Eli Biham
Amr M. Youssef (Eds.)

LNCS 4356

Selected Areas in Cryptography

13th International Workshop, SAC 2006
Montreal, Canada, August 2006
Revised Selected Papers

Springer

Lecture Notes in Computer Science 4356

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

Editorial Board

David Hutchison
Lancaster University, UK
Takeo Kanade
Carnegie Mellon University, Pittsburgh, PA, USA
Josef Kittler
University of Surrey, Guildford, UK
Jon M. Kleinberg
Cornell University, Ithaca, NY, USA
Friedemann Mattern
ETH Zurich, Switzerland
John C. Mitchell
Stanford University, CA, USA
Moni Naor
Weizmann Institute of Science, Rehovot, Israel
Oscar Nierstrasz
University of Bern, Switzerland
C. Pandu Rangan
Indian Institute of Technology, Madras, India
Bernhard Steffen
University of Dortmund, Germany
Madhu Sudan
Massachusetts Institute of Technology, MA, USA
Demetri Terzopoulos
University of California, Los Angeles, CA, USA
Doug Tygar
University of California, Berkeley, CA, USA
Moshe Y. Vardi
Rice University, Houston, TX, USA
Gerhard Weikum
Max-Planck Institute of Computer Science, Saarbruecken, Germany

Eli Biham   Amr M. Youssef (Eds.)

Selected Areas in Cryptography

13th International Workshop, SAC 2006
Montreal, Canada, August 17-18, 2006
Revised Selected Papers

Springer

Volume Editors

Eli Biham
Technion - Israel Institute of Technology
Computer Science Department
Haifa 32000, Israel
E-mail: biham@cs.technion.ac.il

Amr M. Youssef
Concordia University
Concordia Institute for Information Systems Engineering
1425 René Lévesque Blvd. West, Montreal, Quebec, H3G 1M8, Canada
E-mail: youssef@ciise.concordia.ca

Library of Congress Control Number: 2007935809

CR Subject Classification (1998): E.3, D.4.6, K.6.5, F.2.1-2, C.2, H.4.3

LNCS Sublibrary: SL 4 – Security and Cryptology

ISSN 0302-9743
ISBN-10 3-540-74461-4 Springer Berlin Heidelberg New York
ISBN-13 978-3-540-74461-0 Springer Berlin Heidelberg New York

This work is subject to copyright. All rights are reserved, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, re-use of illustrations, recitation, broadcasting, reproduction on microfilms or in any other way, and storage in data banks. Duplication of this publication or parts thereof is permitted only under the provisions of the German Copyright Law of September 9, 1965, in its current version, and permission for use must always be obtained from Springer. Violations are liable to prosecution under the German Copyright Law.

Springer is a part of Springer Science+Business Media

springer.com

© Springer-Verlag Berlin Heidelberg 2007
Printed in Germany

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 12111787 06/3180 5 4 3 2 1 0

Preface

These are the proceedings of SAC 2006, the thirteenth annual workshop on Selected Areas in Cryptography. The workshop was sponsored by the Concordia Institute for Information Systems Engineering, in cooperation with the IACR, the International Association of Cryptologic Research, `www.iacr.org`. This year's themes for SAC were:

1. Design and analysis of symmetric key cryptosystems
2. Primitives for symmetric key cryptography, including block and stream ciphers, hash functions, and MAC algorithms
3. Efficient implementations of symmetric and public key algorithms
4. Side-channel analysis (DPA, DFA, Cache analysis, etc.)

A total of 25 papers were accepted for presentation at the workshop, out of 86 papers submitted (of which one was withdrawn by the authors shortly after the submission deadline). These proceedings contain revised versions of the accepted papers. In addition two invited talks were given: Adi Shamir gave the Stafford Tavares Lecture, entitled "A Top View of Side Channels". The second invited talk was given by Serge Vaudenay entitled "When Stream Cipher Analysis Meets Public-Key Cryptography" (his paper on this topic is enclosed in these proceedings).

The reviewing process was a challenging task, and many good submissions had to be rejected. Each paper was reviewed by at least three members of the Program Committee, and papers co-authored by a member of the Program Committee were reviewed by at least five (other) members. The reviews were then followed by deep discussions on the papers, which contributed a lot to the quality of the final selection. In most cases, extensive comments were sent to the authors. A total of about 300 reviews were written by the committee and external reviewers for the 86 papers, of which 92 reviews were made by 65 external reviewers. Over 240 discussion comments were made by committee members (with up to 30 comments per member). Several papers had deep discussions with 17–19 discussion comments each. In addition, the Co-chairs wrote over 200 additional discussion comments.

It was a pleasure for us to work with the Program Committee, whose members worked very hard during the review process. We are also very grateful to the external referees, who contributed with their special expertise to the selection process. Their work is highly appreciated.

The submission and review process was done using an electronic submission and review software written by Thomas Baignères and Matthieu Finiasz. Thomas and Matthieu also modified and improved their system especially for SAC 2006, with many new features. Their response was very quick and timely, and in many cases features were added or changes were made within less than an hour. We wish to thank them very much for all this work.

We would also like to acknowledge Sheryl Tablan and Sheila Anderson for their great help in the local organization.

Finally, but most importantly, we would like to thank all the authors from all over the world who submitted papers to the workshop, and to all the participants at the workshop.

October 2006

Eli Biham
Amr Youssef

SAC 2006

August 17–18, 2006, Montréal, Canada

Sponsored by the
Concordia Institute for Information Systems Engineering

In cooperation with the
International Association of Cryptologic Research (IACR)

Workshop Co-chairs

Eli Biham, Computer Science Department, Technion – Israel Institute of Technology, Technion City, Haifa 32000, Israel

Amr M. Youssef, Concordia Institute for Information Systems Engineering, Concordia University, 1425 René Lévesque Blvd. West, Montréal, Quebec, H3G 1T7, Canada

Program Committee

Carlisle Adams	University of Ottawa, Canada
Alex Biryukov	University of Luxembourg, Luxembourg
Nicolas Courtois	Axalto, France
Orr Dunkelman	Technion, Israel
Helena Handschuh	Spansion, EMEA, France
Thomas Johansson	Lund, Sweden
Antoine Joux	Université de Versailles St-Quentin-en-Yvelines, France
Pascal Junod	Nagravision, Switzerland
Lars Knudsen	DTU, Denmark
Stefan Lucks	University of Mannheim, Germany
Bart Preneel	Katholieke Universiteit Leuven, Belgium
Matt Robshaw	France Telecom, France
Doug Stinson	University of Waterloo, Canada
Stafford Tavares	Queen's University, Canada
Eran Tromer	Weizmann Institute of Science, Israel
Xiaoyun Wang	Tsinghua University and Shandong University, China
Michael Wiener	Cryptographic Clarity, Canada

External Referees

Frederik Armknecht
Thomas Baignères
Elad Barkan
Lejla Batina
Aurélie Bauer
Come Berbain
Johannes Bloemer
Colin Boyd
Anne Canteaut
Rafi Chen
Carlos Cid
Jeremy Clark
Scott Contini
Ivan Damgaard
Blandine Debraize
Håkan Englund
Aleks Essex
Matthieu Finiasz
Ewan Fleischmann
Guillaume Fumaroli
Henri Gilbert
Martin Hell
Matt Henricksen
Jonathan J. Hoch
Tetsu Iwata
Ulrich Kühn
Nathan Keller
Matthias Krause
Simon Künzli
Tanja Lange
Joe Lano
Stefan Mangard
Alexander Maximov
Alexander May
Alfred Menezes
Nele Mentens
Brad Metz
Marine Minier
Jean Monnerat
James Muir
Sean Murphy
Mridul Nandi
Gregory Neven
Dag Arne Osvik
Pascal Paillier
Souradyuti Paul
Jan Pelzl
Gilles Piret
Axel Poschmann
Soren S. Thomsen
Kazuo Sakiyama
Kai Schramm
Jean-Pierre Seifert
Nigel Smart
Heiko Stamer
François-Xavier Standaert
Dirk Stegemann
Emin Tatli
Nicolas Theriault
Boaz Tsaban
Ingrid Verbauwhede
Frederik Vercauteren
Charlotte Vikkelsoe
Christopher Wolf
Robert Zuccherato

Table of Contents

Improved DST Cryptanalysis of IDEA

Eyüp Serdar Ayaz and Ali Aydın Selçuk

Department of Computer Engineering
Bilkent University
Ankara, 06800, Turkey
{serdara,selcuk}@cs.bilkent.edu.tr

Abstract. In this paper, we show how the Demirci-Selcuk-Ture attack, which is currently the deepest penetrating attack on the IDEA block cipher, can be improved significantly in performance. The improvements presented reduce the attack's plaintext, memory, precomputation time, and key search time complexities. These improvements also make a practical implementation of the attack on reduced versions of IDEA possible, enabling the first experimental verifications of the DST attack.

1 Introduction

International Data Encryption Algorithm (IDEA) is one of the most popular block ciphers today, commonly used in popular software applications such as PGP. IDEA is known to be extremely secure too: Despite its relatively long history and numerous attempts to analyze it [1, 2, 3, 4, 5, 6, 8, 9, 10, 13, 14, 15], most known attacks on IDEA, which is an 8.5-round cipher, apply to no more than the cipher reduced to 4 rounds. The most effective attack currently known is due to Demirci, Selçuk, and Türe (DST) [7], which is a chosen plaintext attack effective on IDEA up to 5 rounds.

In this paper, we study the ways of enhancing the DST attack and improving its performance. The improvements discussed include shortening the variable part of the plaintexts, reducing the sieving set size, and utilizing previously unused elimination power of the sieving set. The improvements result in a reduction in the plaintext, memory, precomputation time, and key search time complexities of the attack and show that the DST attack can be conducted significantly more efficiently than it was originally thought.

The rest of this paper is organized as follows: In Section 2, we briefly describe the IDEA block cipher. In Section 3, we give an overview of the DST attack. In Section 4, we present several key observations on the DST attack and how to optimize the attack accordingly. In Section 5, we analyze the success probability of the attack according to these optimizations. In Section 6, we present our experimental results and compare them with our theoretical expectations. In Section 7, we calculate the total complexity of the revised attack. Finally in Section 8, we conclude with an overall assessment of the work presented.

© Springer-Verlag Berlin Heidelberg 2007

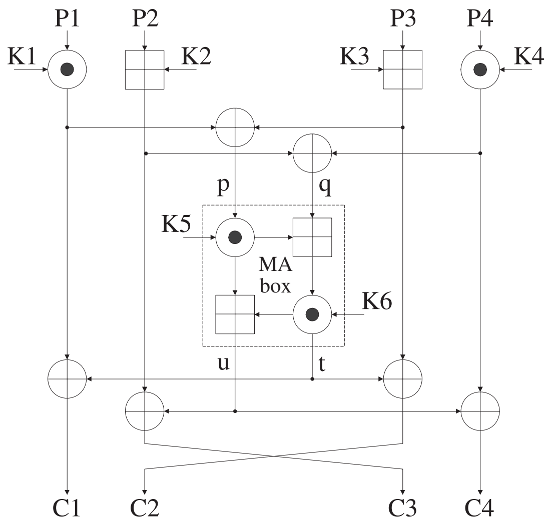


Fig. 1. One round of IDEA

1.1 Notation

We use the following notation in this paper: For modular addition and modular subtraction we use the symbols $\boxplus$ and $\boxminus$ respectively. Bitwise exclusive-or (XOR) is denoted by $\oplus$ and the IDEA multiplication is denoted by $\odot$. The plaintext is shown as (P_1, P_2, P_3, P_4) which is a concatenation of four 16-bit subblocks. Similarly the ciphertext is shown as (C_1, C_2, C_3, C_4). The superscripts in parenthesis denote the round numbers. There are six round-key subblocks for each round which are denoted by $K_1, K_2, K_3, K_4, K_5, K_6$. The inputs of the MA-box are denoted by p and q and the outputs are denoted by u and t.

The least significant bit of a variable x is denoted by $\text{lsb}(x)$, the ith least significant bit is denoted by $\text{lsb}_i(x)$, and the least significant i bits are denoted by $\text{lsbs}_i(x)$. Similarly, the most-significant counterparts of these operators are respectively denoted by $\text{msb}(x)$, $\text{msb}_i(x)$, and $\text{msbs}_i(x)$. Concatenation of two variables x, y is denoted by $(x|y)$. Finally, an inclusive bit interval between the mth and nth bits of a round-key subblock $K_j^{(i)}$ is denoted by $K_j^{(i)}[m \ldots n]$.

2 IDEA Block Cipher

The IDEA block cipher is a modified version of the PES block cipher [11, 12]. IDEA has 64-bit blocks and takes 128-bit keys. The blocks are divided into four 16-bit words and all the operations are on these words. Three different "incompatible" group operations are performed on these words: Bitwise XOR, modular addition, and the *IDEA multiplication*, which is multiplication modulo $2^{16} + 1$ where 0 represents 2^{16}.

There are two parts in an IDEA round. The first is the transformation part:

$$T : (P_1, P_2, P_3, P_4) \rightarrow (P_1 \odot K_1, P_2 \boxplus K_2, P_3 \boxplus K_3, P_4 \odot K_4).$$

In the second part, two inputs of the MA-box are calculated as $p = (P_1 \odot K_1) \oplus (P_3 \boxplus K_3)$ and $q = (P_2 \boxplus K_2) \oplus (P_4 \odot K_4)$. The outputs of the MA-box are $t = ((p \odot K_5) \boxplus q) \odot K_6$ and $u = (p \odot K_5) \boxplus t$. After these calculations t is XORed with the first and third output of the transformation part and u is XORed with the second and fourth. Finally, the ciphertext is formed by taking the outer blocks directly and exchanging the inner blocks.

$$\begin{aligned} C_1 &= (P_1 \odot K_1) \oplus t, \\ C_2 &= (P_3 \boxplus K_3) \oplus t, \\ C_3 &= (P_2 \boxplus K_2) \oplus u, \\ C_4 &= (P_4 \odot K_4) \oplus u. \end{aligned}$$

IDEA consists of eight full rounds and an additional half round, which consists of one transformation part.

The key schedule creates 16-bit round subkeys from a 128-bit master key by taking 16 bits for a subkey and shifting the master key 25 bits after every 8th round key.

Decryption can be done using the encryption algorithm with the multiplicative and additive inverses of the round key subblocks in the transformation part and the same key subblocks in the MA-box.

3 The DST Attack

In this section, we give a brief overview of the DST attack with the relevant properties of the IDEA cipher.

3.1 Some Properties of IDEA

The following are some key observations of Demirci et al. [7] on the IDEA cipher which are fundamental to the DST attack. Proofs can be found in the original paper [7].

Theorem 1. *Let $\mathcal{P} = \{(P_1, P_2, P_3, P_4)\}$ be a set of 256 plaintexts such that*

- P_1*,* P_3*,* $\mathrm{lsbs}_8(P_2)$ *are fixed,*
- $\mathrm{msbs}_8(P_2)$ *takes all possible values over* $0, 1, \ldots, 255$*,*
- P_4 *varies according to* P_2 *such that* $q = (P_2 \boxplus K_2^{(1)}) \oplus (P_4 \odot K_4^{(1)})$ *is fixed.*

For $p^{(2)}$ denoting the first input of the MA-box in the second round, the following properties will hold in the encryption of the set $\mathcal{P}$:

- $\mathrm{lsbs}_8(p^{(2)})$ *is fixed,*
- $\mathrm{msbs}_8(p^{(2)})$ *takes all possible values over* $0, 1, \ldots, 255$.

Moreover, the $p^{(2)}$ values, when ordered according to the plaintext's $\mathrm{msbs}_8(P_2)$ *beginning with* $\mathrm{msbs}_8(P_2) = 0$, *will be of the form*

$$(y_0|z), (y_1|z), \ldots, (y_{255}|z)$$

for some fixed, 8-bit z, and $y_i = (((i \boxplus a) \oplus b) \boxplus c) \oplus d$, for $0 \leq i \leq 255$ and fixed, 8-bit a, b, c, d.

Theorem 2. *In the encryption of the plaintext set $\mathcal{P}$ defined in Theorem 1,* $\mathrm{lsb}(K_5^{(2)} \odot p^{(2)})$ *equals either* $\mathrm{lsb}(C_2^{(2)} \oplus C_3^{(2)})$ *or* $\mathrm{lsb}(C_2^{(2)} \oplus C_3^{(2)}) \oplus 1$ *for all the 256 plaintexts in $\mathcal{P}$.*

Lemma 1. *In the IDEA round function, the following property is satisfied:*

$$\mathrm{lsb}(t \oplus u) = \mathrm{lsb}(p \odot K_5).$$

Corollary 1. $\mathrm{lsb}(C_2^{(i)} \oplus C_3^{(i)} \oplus (K_5^{(i)} \odot (C_1^{(i)} \oplus C_2^{(i)}))) = \mathrm{lsb}(C_2^{(i-1)} \oplus C_3^{(i-1)} \oplus K_2^{(i)} \oplus K_3^{(i)})$.

Corollary 2. $\mathrm{lsb}(C_2^{(i)} \oplus C_3^{(i)} \oplus (K_5^{(i)} \odot (C_1^{(i)} \oplus C_2^{(i)}))) \oplus (K_5^{(i-1)} \odot (C_1^{(i-1)} \oplus C_2^{(i-1)}))) = \mathrm{lsb}(C_2^{(i-2)} \oplus C_3^{(i-2)} \oplus K_2^{(i)} \oplus K_3^{(i)} \oplus K_2^{(i-1)} \oplus K_3^{(i-1)})$.

3.2 Attack on 3-Round IDEA

The DST attack starts with a precomputation phase where a "sieving set" is prepared which consists of 2^{56} elements of 256-bit strings

$$S = \{f(a, b, c, d, z, K_5^{(2)}) : \ 0 \leq a, b, c, d, z < 2^8, \ 0 \leq K_5^{(2)} < 2^{16}\}.$$

computed bitwise as

$$f(a, b, c, d, z, K_5^{(2)})[i] = \mathrm{lsb}(K_5^{(2)} \odot (y_i|z))$$

for $0 \leq i < 255$, where $y_i = (((i \boxplus a) \oplus b) \boxplus c) \oplus d$.

Once preparation of the sieving set is completed, the main phase of the attack follows. Below is a description of the basic attack on the 3-round IDEA:

1. The attacker takes a chosen plaintext set $\mathcal{R} = \{(P_1, P_2, P_3, P_4)\}$, where P_1, P_3, and $\mathrm{lsbs}_8(P_2)$ are fixed at an arbitrary value, and $\mathrm{msbs}_8(P_2)$ and P_4 take all possible values. All elements of $\mathcal{R}$ are encrypted with the 3-round IDEA.
2. For each value of $K_2^{(1)}$ and $K_4^{(1)}$, take a subset $\mathcal{P}$ of 256 plaintexts from $\mathcal{R}$ such that $\mathrm{msbs}_8(P_2)$ varies from 0 to 255 and P_4 is chosen to make $(P_2 \boxplus K_2^{(1)}) \oplus (P_4 \odot K_4^{(1)})$ constant.
3. For each value of $K_5^{(3)}$, a 256-bit string is formed by computing

$$\mathrm{lsb}(C_2^{(3)} \oplus C_3^{(3)} \oplus (K_5^{(3)} \odot (C_1^{(3)} \oplus C_2^{(3)})))$$

   for each of the plaintexts in $\mathcal{P}$, ordered by $\mathrm{msbs}_8(P_2)$. If the current $(K_2^{(1)}, K_4^{(1)}, K_5^{(3)})$ triple is correct, this 256-bit string must be found in the sieving set. If it cannot be found, the key triple is eliminated.

4. If many key candidates survive this test, steps 1–3 can be repeated with a different plaintext set $\mathcal{R}$ until a single triple remains. We call one execution of steps 1–3 an *elimination round*.

This attack finds $K_2^{(1)}$, $K_4^{(1)}$, $K_5^{(3)}$ directly by exhaustive search. We can also find $K_5^{(2)}$ indirectly by storing the corresponding $K_5^{(2)}$ value along with each sieving set entry and returning its value in case of a sieving set hit.

3.3 Attack on 3.5-Round IDEA

The 3.5-round attack works similar to the 3-round attack. To find $\mathrm{lsb}(C_2^{(3)} \oplus C_3^{(3)} \oplus (K_5^{(3)} \odot (C_1^{(3)} \oplus C_2^{(3)})))$ we encrypt $\mathcal{P}$ with 3.5-round IDEA and decrypt $C_1^{(3.5)}$ and $C_2^{(3.5)}$ for a half-round by exhaustive search on $K_1^{(4)}$ and $K_2^{(4)}$. It is not necessary to find $C_3^{(3)}$ since $C_2^{(3)} \oplus C_3^{(3)}$ is equal to $C_2^{(3.5)} \oplus C_3^{(3.5)}$ or $C_2^{(3.5)} \oplus C_3^{(3.5)} \oplus 1$ for all 256 ciphertexts.

3.4 Attacks on Higher Number of Rounds

The attack on higher-round IDEA versions utilizes Corollary 2 to find $\mathrm{lsb}(C_2^{(2)} \oplus C_3(2))$ or its complement by computing $\mathrm{lsb}(C_2^{(4)} \oplus C_3^{(4)} \oplus (K_5^{(4)} \odot (C_1^{(4)} \oplus C_2^{(4)}))) \oplus (K_5^{(3)} \odot (C_1^{(3)} \oplus C_2^{(3)}))$.

In the 4-round attack, it is necessary to try exhaustively all possible values of $K_1^{(4)}$, $K_2^{(4)}$, $K_5^{(4)}$, and $K_6^{(4)}$ to find $C_1^{(3)} \oplus C_2^{(3)}$. For the 4.5-round attack, we need to search over $K_1^{(5)}$, $K_2^{(5)}$, $K_3^{(5)}$, $K_4^{(5)}$ to obtain the 4th round outputs. For the 5-round attack, $K_5^{(5)}$, and $K_6^{(5)}$ are also searched.

3.5 Complexity of the DST Attack

In these attacks, the space complexity and precomputation time are independent of the number of rounds while the key search time varies depending on the number of rounds attacked.

Memory required for the attack is determined by the size of the sieving set, which consists of 2^{56} elements of 256-bit strings.

Precomputation time is the time that is needed to prepare the sieving set. We need to calculate the f function once for each bit of the sieving set. There are 2^{56} elements of 256-bit strings, therefore the precomputation time complexity is 2^{64} f computations.

Complexity of the main phase of the attack, the key search time, is different in the 3-, 3.5-, 4-, 4.5- and 5-round attacks depending on the number of key bits searched. In each of these attacks, a lookup string is computed over 256 ciphertexts for each key candidate, contributing a complexity factor of 2^8. In the 3-round attack, the key searched is 34 bits, making the key search time complexity 2^{42} partial decryptions. The 3.5-round attack searches 32 more bits, making the time complexity 2^{74}. The 4-round attack needs 16 more key bits which raises the time complexity to 2^{90}. We search 114 key bits for the 4.5-round attack and 119 bits for the 5-round attack, with the complexities of 2^{122} and 2^{127} partial decryptions respectively.

4 The Improved DST Attack

In this section we describe the improvements we have made on the DST attack which reduce the precomputation time, key search time, space, and plaintext complexities of the attack.

4.1 Shortening the Variable Parts

The original DST attack partitioned P_2 into 8-bit fixed and 8-bit variable parts, where the variable part took all possible 2^8 values over the chosen plaintext set $\mathcal{P}$. One can observe that in fact it is not necessary to have a balanced partition of P_2 and the attack works just as fine with an imbalanced partition. Accordingly, one can obtain significant savings in the attack by reducing the size of the variable part. For v denoting the number of most significant bits in the variable part of P_2, the sieving set for the attack becomes,

$$S = \{f(a, b, c, d, z, K_5^{(2)}) : \ 0 \le a, b, c, d < 2^v, \ 0 \le z < 2^{16-v}, \ 0 \le K_5^{(2)} < 2^{16}\}.$$

Note that shortening the variable part of P_2 narrows the sieving set both vertically and horizontally. With a v-bit variable part, the sieving set entries will be 2^v bits each instead of 256 bits. Furthermore, the number of entries in the sieving set will be reduced by a factor of $2^{3(8-v)}$. This change also decreases the key search time by 2^{8-v}, since for each candidate key, we encrypt 2^v plaintexts to form the bit string to be searched in the sieving set instead of 256. We will see in Section 5 that having five variable bits is enough for an effective elimination. Therefore by an imbalanced partition of P_2, we obtain an improvement by a factor of 2^9 in precomputation time, 2^3 in key search time and 2^{12} in space.

4.2 Size of the Sieving Set

Another reduction in the size of the sieving set comes from the identical entries yielded by different (a, b, c, d) quadruples, i.e., the *collisions*. In the DST attack all the elements of the sieving set were thought to be distinct [7]. We have found that actually a significant number of collisions exist among the sieving set entries. Some of these collisions were found analytically and some were observed empirically. The analytical findings were obtained according to the y_i values:

Definition 1. *We call two (a, b, c, d) quadruples, $0 \le a, b, c, d < 2^v$, equivalent if they give the same $y_i = (((i \boxplus a) \oplus b) \boxplus c) \oplus d$ value for all $0 \le i < 2^v$.*

Lemma 2. *For any quadruple (a, b, c, d), complementing the most significant bit of any two or four of a, b, c, d yields an equivalent quadruple.*

Proof. We are working in modulo 2^v, so there is no carry bit for addition on the most significant bit. This means changing the most significant bit of a variable in the addition operation changes only the most significant bit of the result. Exclusive-or has the same effect on all bits. So, in an expression of addition and exclusive-or operations, changing one of the variables' most significant bit flips the most significant bit of the result. Changing the most significant bit of an even number of the variables leaves the result unchanged. □

This property gives $\binom{4}{0}+\binom{4}{2}+\binom{4}{4}=8$ equivalent (a,b,c,d) quadruples. Another equivalence is related to the complement operation:

Lemma 3. *(a,b,c,d) is equivalent to $(a,\overline{b},\overline{c}\boxplus 1,\overline{d})$ for $0 \le a,b,c,d < 2^v$.*

Proof.

$$\begin{aligned}(((i\boxplus a)\oplus \overline{b})\boxplus \overline{c}\boxplus 1)\oplus \overline{d} &= (\overline{((i\boxplus a)\oplus b)}\boxplus \overline{c}\boxplus 1)\oplus \overline{d}\\ &= ((2^v-1-((i\boxplus a)\oplus b))\boxplus(2^v-c))\oplus \overline{d}\\ &= (2^{v+1}-1-(((i\boxplus a)\oplus b)\boxplus c))\oplus \overline{d}\\ &= \overline{(((i\boxplus a)\oplus b)\boxplus c)}\oplus \overline{d}\\ &= (((i\boxplus a)\oplus b)\boxplus c)\oplus d\end{aligned}$$ □

This relation can be applied to the 8 equivalent quadruples found in Lemma 1 yielding 16 equivalent quadruples.

The third equivalence is related to the second most significant bit:

Lemma 4. *(a,b,c,d) is equivalent to $(a\boxplus 2^{v-2},b,c\boxplus 2^{v-2},d)$ if $\mathrm{msb}_2(b)=1$, and to $(a\boxplus 2^{v-2},b,c\boxminus 2^{v-2},d)$ if $\mathrm{msb}_2(b)=0$.*

Proof. Assume $\mathrm{msb}_2(b)=1$ and consider $((((i\boxplus a)\boxplus 2^{v-2})\oplus b)\boxplus 2^{v-2})\boxplus c$. Obviously $\mathrm{msb}_2((i\boxplus(a\boxplus 2^{v-2}))\oplus b)=\mathrm{msb}_2(i\boxplus a)$. As for the most significant two bits, if there is a carry in the outer addition of $(i\boxplus a)\boxplus 2^{v-2}$, there will also be a carry on the outmost addition of $(((i\boxplus a)\boxplus 2^{v-2})\oplus b)\boxplus 2^{v-2}$ since $\mathrm{msb}_2(b)=1$. Similarly, if there is no carry in the outer addition of $(i\boxplus a)\boxplus 2^{v-2}$, there will also be no carry on the outmost addition of $(((i\boxplus a)\boxplus 2^{v-2})\oplus b)\boxplus 2^{v-2}$. So the most significant bit of the result is not changed. The second most significant bit is complemented twice, so it also remains same. Hence in both cases $((i\boxplus(a\boxplus 2^{v-2}))\oplus b)\boxplus(c\boxplus 2^{v-2})=((i\boxplus a)\oplus b)\boxplus c$.

Now, assume $\mathrm{msb}_2(b)=0$ and consider $((((i\boxplus a)\boxplus 2^{v-2})\oplus b)\boxminus 2^{v-2})\boxplus c$. Obviously $\mathrm{msb}_2((i\boxplus(a\boxplus 2^{v-2}))\oplus b)=\mathrm{msb}_2(\overline{i\boxplus a})$. As for the most significant two bits, if there is a carry in the outer addition of $(i\boxplus a)\boxplus 2^{v-2}$, then there will be no carry on the outmost addition of $(((i\boxplus a)\boxplus 2^{v-2})\oplus b)\boxplus 2^{v-2}$ since $\mathrm{msb}_2(b)=0$. Similarly, if there is no carry in the outer addition of $(i\boxplus a)\boxplus 2^{v-2}$, then there will be a carry on the outmost addition of $(((i\boxplus a)\boxplus 2^{v-2})\oplus b)\boxplus 2^{v-2}$. So the most significant bit of the result is changed in the operation $((((i\boxplus a)\boxplus 2^{v-2})\oplus b)\boxminus 2^{v-2})$. Adding 2^{v-1} will neutralize this, so the most significant bit of the result will remain the same. The second most significant bit is complemented twice, so it will be unchanged. Hence in both cases $((i\boxplus(a\boxplus 2^{v-2}))\oplus b)\boxplus(c\boxplus 2^{v-2}\boxplus 2^{v-1})$ $=((i\boxplus(a\boxplus 2^{v-2}))\oplus b)\boxplus(c\boxminus 2^{v-2})=((i\boxplus a)\oplus b)\boxplus c$. □

When Lemma 4 is applied to all 16 equivalent quadruples, the size of the equivalence class is doubled, yielding 32 equivalent quadruples.

If we discard the two most significant bits of a and one most significant bit of b, c, d, we will find exactly one of these 32 equivalent quadruples, since the equivalent quadruples take all possible values over these five bits. Therefore, in the sieving set formation phase we do not have to search all combinations of (a, b, c, d); conducting the search on $\mathrm{lsbs}_{v-2}(a)$, $\mathrm{lsbs}_{v-1}(b)$, $\mathrm{lsbs}_{v-1}(c)$, $\mathrm{lsbs}_{v-1}(d)$ suffices. This reduction decreases both the precomputation time and the sieving set size by a factor of 2^5.

The collisions we dealt with in this section are exclusively based on equivalent (a, b, c, d) quadruples. As the experimental results in Section 6 show, there are other collisions as well and the actual collision rate can safely be assumed to be 2^6 or higher.

4.3 Indirect Elimination Power from the Sieving Set

The effectiveness of the DST attack can be improved significantly by using previously unutilized elimination power from the sieving set. When a lookup string is matched with a sieving set entry, we can do a further correctness test on the key by checking whether the key values used in obtaining the set entry matched are consistent with the round keys used in obtaining the lookup string.

First, we can check the $K_5^{(2)}$ found in a sieving set hit for consistency with the keys used in the partial decryption. The 3-round attack searches $K_2^{(1)}[17 \dots 32]$, $K_4^{(1)}[49 \dots 64]$, $K_5^{(3)}[51 \dots 66]$, which intersects with $K_5^{(2)}[58 \dots 73]$ on 9 bits over $[58 \dots 66]$. If we store the values of these nine bits of $K_5^{(2)}$ for each sieving set entry and compare them to the corresponding bits of the key candidate used in the partial decryption in case of a hit, a wrong key's chances of passing the sieving test will be reduced by a factor of 2^9.

The keys found in further round attacks—$K_1^{(4)}$, $K_2^{(4)}$ for 3.5-round attack, $K_5^{(4)}$, $K_6^{(4)}$ for 4-round attack, $K_1^{(5)}$, $K_2^{(5)}$, $K_3^{(5)}$, $K_4^{(5)}$ for 4.5-round attack and $K_5^{(5)}$, $K_6^{(5)}$—do not bring us any more bits intersecting with $K_5^{(2)}$.

The seven bits of $K_5^{(2)}$ that do not intersect with the searched round keys can be utilized to deduce the corresponding seven bits of the master key. Moreover, in attacks that use multiple elimination rounds, a check on these bits can be carried out to test the consistency of the sieving set hits across different elimination rounds. Either way, these seven bits can be used to reduce the set of key candidates by a factor of 2^7 per elimination round.

A similar consistency check can be applied also on the a values of the sieving set entries. Note that the 32 equivalent quadruples found in Section 4.2 have the same $\mathrm{lsbs}_{v-2}(a)$ value. Hence, in case of a sieving set hit, the a value of the sieving set entry matched can be compared on the $v - 2$ low order bits to the a value of the partial decryption,

$$a \;=\; \mathrm{msbs}_v(K_2^{(1)}) + \mathrm{carry}(\mathrm{lsbs}_{16-v}(P_2) \boxplus \mathrm{lsbs}_{16-v}(K_2^{(1)})),$$

which is fixed and known over the plaintext set $\mathcal{P}$. This extension brings an extra elimination power of 2^{v-2} to the attack while costing $v-2$ bits of storage per sieving set entry.

A similar check can be carried out over the c values. The 32 equivalent quadruples are equal to $\pm c \bmod 2^{v-2}$ over $\mathrm{lsbs}_{v-2}(c)$ while $\mathrm{msbs}_2(c)$ takes all possible four values. Moreover, for every value of c there are two possible values of $\mathrm{msbs}_v(K_3^{(2)})$ since

$$\begin{aligned} c \;=\; & \mathrm{msbs}_v(K_3^{(2)}) \;+ \\ & \mathrm{carry}(((\mathrm{lsbs}_{16-v}(P_2) \boxplus \mathrm{lsbs}_{16-v}(K_2^{(1)})) \oplus \mathrm{lsbs}_{16-v}(u^{(1)})) \boxplus \mathrm{lsbs}_{16-v}(K_3^{(2)})) \end{aligned}$$

where the carry bit is an unknown. The key bits $\mathrm{msbs}_v(K_3^{(2)})$ are covered completely by $K_2^{(1)}$ for $v \leq 7$ which is the case in our attacks. Therefore, by conducting a consistency check between the key candidate tried and the c value of the sieving set entry matched, we can reduce the number of keys by an additional factor of 2^{v-4}. As in the case of a, this check on c costs an extra $v-2$ bits of storage per sieving set entry.

5 The Success Probability

As discussed in Section 4, we have found the actual size of the sieving set to be about 2^6 times smaller than what was thought previously, due to the collisions among the set entries. Hence, with a v-bit variable part of P_2, the expected size of the sieving set is about 2^{26+3v}. When a wrong key is checked against the sieving set, the probability of two random 2^v-bit strings matching by chance is 2^{-2^v}. With the indirect elimination power from $K_5^{(2)}$, $\mathrm{lsbs}_{v-2}(a)$, and $\mathrm{lsbs}_{v-2}(c)$, the probability of a random match between the lookup string and a particular sieving set entry is further reduced to $2^{-(2^v+2v+3)}$. Hence, the probability of a wrong key's passing the test (i.e., matching at least one entry in the sieving set) is now reduced to

$$1 - \left(1 - \frac{1}{2^{2^v+2v+3}}\right)^{(2^{26+3v})} \;\approx\; 2^{-2^v+v+23}$$

for a given v. Accordingly, $v=5$ is the smallest value of v that gives a non-negligible elimination power, where a wrong key's probability of passing the test is 2^{-4}. This probability drops substantially by increasing v: For $v=6$, it becomes 2^{-33}; for $v=7$ it is 2^{-95}, and for $v=8$ it is 2^{-221}.

The probability of elimination discussed above is for attacks with one elimination round (i.e., one pass of Steps 1–3 of the attack algorithm). In attacks that use several elimination rounds, a consistency check on $K_5^{(2)}[67\ldots73]$ is also possible in the elimination rounds after the first one. In this case, the probability of a wrong key's having a consistent match with a sieving set entry is further

Table 1. The actual sieving set sizes for 32-bit IDEA ($w = 8$) with $v = 5$. Each column shows the results for a particular combination of LS, $K_5^{(2)}$, a, c, included in the set entries. As more information is included, the collision rate approaches to the theoretical expectation given in the last column.

	LS	LS, $K_5^{(2)}$	LS, $K_5^{(2)}, a$	LS, $K_5^{(2)}, a, c$	$2^{2w+3v-6}$
$v = 5$	$2^{22.3}$	$2^{23.6}$	$2^{24.5}$	$2^{24.7}$	2^{25}

reduced to $2^{-(2^v+2v+10)}$. Hence, the probability of a wrong key's passing such an elimination round is

$$1 - \left(1 - \frac{1}{2^{2^v+2v+10}}\right)^{(2^{26+3v})} \approx 2^{-2^v+v+16}.$$

The probability of a wrong key's passing an elimination test with r rounds is therefore

$$2^{(-2^v+v+23)+(r-1)(-2^v+v+16)} = 2^{r(-2^v+v+16)+7}.$$

To successfully conclude an attack, we will need to run as many elimination rounds as needed to reduce the number of surviving key candidates to one. In the 3-round attack, 34 key bits are searched giving 2^{34} candidates in total. For $v = 5$, the probability of a wrong key's not being eliminated after r iterations is 2^{-11r+7}. Hence, four elimination rounds would suffice to eliminate virtually all wrong keys while keeping $v = 5$ in the 3-round attack. Similarly, two elimination rounds would suffice for $v = 6$ and one elimination round for $v = 7$.

6 Experimental Results

The improvements obtained have made a practical implementation of the DST attack possible on reduced versions of IDEA. We tested the attack on IDEA reduced to 3 rounds with a block size of 32 bits (i.e., word size $w = 8$). The key size is reduced accordingly to 64 bits; the key schedule rotates the master key 11 bits after every 8th subkey produced. The attack is tested with $v = 5$, since $v \geq 6$ is still beyond our limits of feasibility, and $v \leq 4$ does not produce a meaningful attack as the lookup string length, 2^v, is too short to give any significant elimination.

First we tested the size of the sieving set in comparison to our theoretical expectation $2^{2w+3v-6}$. The results, summarized in Table 1, show that the actual sieving set size is somewhat further smaller than our expectation due to unaccounted collisions, by a factor of 8 to 1.5, depending on the amount of extra information included—$K_5^{(2)}$, a,or c.[1]

[1] Tests were carried out for other combinations of $K_5^{(2)}$, a, and c not listed in Table 1 as well. Due to space limitations, only the most essential ones are listed here, according to their order of significance.

Table 2. The experimental results for the DST attack with $v = 5$. The results in the table are the ratio of wrong keys passing the sieving set test uneliminated, obtained over 1000 runs of the attack, each containing 2^{18} keys tested. The theoretical results are the calculations in Section 5 according to the actual sieving set sizes in Table 1.

	LS	LS, $K_5^{(2)}$	LS, $K_5^{(2)}, a$	LS,$K_5^{(2)}, a, c$
Theoretical	$2^{-9.7}$	$2^{-13.4}$	$2^{-15.5}$	$2^{-16.3}$
Empirical	$2^{-9.6}$	$2^{-12.3}$	$2^{-13.2}$	$2^{-13.9}$

We implemented the DST attack with $v = 5$ to see its actual success. Table 2 summarizes the result of these tests, where the wrong keys are eliminated according to the lookup string (LS), $K_5^{(2)}$, a, and c; and the ratio of the uneliminated ones are listed. The test results are compared to the theoretical results calculated in Section 5.

An analysis of the experimental results reveals several key points. First and foremost, the DST attack works as expected. Especially when only LS is used in elimination, the expected and the actual results are almost identical. When $K_5^{(2)}$, a, and c are also included in the process, the power of the attack is significantly boosted. There appears to be a slight deviation from the expectations however, which probably results from some subtle correlations involved. Accordingly, there may be a few wrong keys left at the end of the attack, which can easily be removed by an extra elimination round or by exhaustive search.

7 Complexity of the Attack

The optimizations discussed in this paper provide significant reductions in the space, precomputation time, and key search time complexities of the DST attack. Space complexity of the attack is mainly the size of the sieving set. Each sieving set entry contains a 2^v-bit lookup string. Additionally we need to store the $K_5^{(2)}$, $\mathrm{lsbs}_{v-2}(a)$, and $\mathrm{lsbs}_{v-2}(c)$ values to have the extra elimination power, which costs us an extra $12+2v$ bits per entry. The number of entries in the set is about 2^{3v+26}. Thus the overall space requirement of the sieving set is $2^{3v+26} \cdot (2^v + 2v + 12)$ bits. In terms of the IDEA block size, this is less than 2^{41} IDEA blocks for $v = 5$.

Precomputation time complexity is the time required to calculate the sieving set. We need to compute the f function 2^v times for each sieving set entry. The number of entries calculated for the sieving set is $2^{3v+32-5}$ since the most significant bits of a, b, c, d and the second most significant bit of a need not to be searched. Hence the precomputation time complexity is 2^{4v+27} f computations which is roughly equivalent to 2^{4v+26} IDEA rounds. The precomputation time is the dominant time complexity only for the 3-round attack.

Key search time complexity depends on both the number of rounds attacked and the number of variable bits in P_2. For each candidate key set, we take 2^v values of $\mathrm{msbs}_v(P_2)$ and calculate the lookup string by partial decryptions. This procedure may need to be repeated several times if the attack requires multiple elimination rounds.

Table 3. A comparison of the complexities of the basic DST attack and the optimized version. The space complexity figures are in terms of one IDEA block (64 bits). The unit of precomputation time complexity is one computation of the f function. The key search complexities are compared in terms of the number of partial decryptions to be executed. The optimized attack figures are given for $v = 5, 6, 7$ which yield the best results.

	DST	$v = 5$	$v = 6$	$v = 7$
Space complexity	2^{58}	2^{41}	2^{45}	2^{49}
Precomputation	2^{64}	2^{47}	2^{51}	2^{55}
Key search, 3-round	2^{42}	2^{39}	2^{40}	2^{41}
3.5-round	2^{74}	2^{71}	2^{72}	2^{73}
4-round	2^{90}	2^{87}	2^{88}	2^{89}
4.5-round	2^{122}	2^{119}	2^{120}	2^{121}
5-round	2^{127}	2^{124}	2^{125}	2^{126}

Table 4. Plaintext complexities of the DST attack for different v. The improvements over the original attack ($v = 8$) in this respect, although non-trivial, is relatively less significant compared to the other improvements.

Attack	$v = 5$	$v = 6$	$v = 7$	$v = 8$
3-round	2^{23}	2^{22}	2^{23}	2^{24}
3.5-round	$2^{23.6}$	2^{23}	2^{23}	2^{24}
4-round	2^{24}	2^{23}	2^{23}	2^{24}
4.5-round	$2^{24.6}$	$2^{23.6}$	2^{23}	2^{24}
5-round	$2^{24.6}$	$2^{23.6}$	2^{23}	2^{24}

The effect of multiple elimination rounds on the attack's complexity is two fold. First, a different plaintext set $\mathcal{R}$ would be needed for each elimination round, making the total plaintext complexity of the attack $r \cdot 2^{16+v}$ for r denoting the number of elimination rounds to be applied. Second, the complexity of the key search phase would increase due to multiple repetitions of the elimination procedure. However, this increase can be expected to be relatively marginal, since the extra elimination rounds will be applied only to the keys that have passed the previous tests. Given that each elimination round will remove the vast majority of the wrong keys, the additional time complexity from the extra elimination rounds will be negligible.

The space and time complexities of the optimized DST attack in comparison to the basic attack are summarized in Table 3; the plaintext complexities are given in Table 4.

8 Conclusion

In this paper, we described several improvements on the DST attack [7] on IDEA and showed how the attack can be made significantly more efficient. The improvements reduce the plaintext, memory, precomputation, and the time complexity of the attack. The new attack becomes the most efficient attack on all these four accounts on the 4.5- and 5-round IDEA, and the most efficient in plaintext complexity on the 4-round cipher along with [10].

With the current improvements, a practical implementation of the attack has also become feasible and we provided the first experimental verifications of the DST attack.

An even more significant improvement on the DST attack would be to extend it beyond 5 rounds of IDEA. Unfortunately, the round keys that need to be tried exhaustively in the partial decryption phase covers all the 128 key bits in the 5.5-round or higher round versions of the attack. Hence, no matter how much improvement is achieved on the core section of the attack, the overall attack cannot be made perform faster than exhaustive search on 5.5 or more rounds. We leave it as an open research problem to make the fundamental ideas of the DST attack work effectively on 5.5 or more rounds of the IDEA cipher.

Acknowledgments

We would like to thank Hüseyin Demirci for several helpful suggestions and comments on this paper.

References

[1] 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)
[2] Biryukov, A., Nakahara Jr., J., Preneel, B., Vandewalle, J.: New Weak-Key Classes of IDEA. In: Deng, R.H., Qing, S., Bao, F., Zhou, J. (eds.) ICICS 2002. LNCS, vol. 2513, pp. 315–326. Springer, Heidelberg (2002)
[3] Borst, J., Knudsen, L.R., Rijmen, V.: Two Attacks on Reduced IDEA (extended abstract). In: Fumy, W. (ed.) EUROCRYPT 1997. LNCS, vol. 1233, pp. 1–13. Springer, Heidelberg (1997)
[4] Daemen, J., Govaerts, R., Vandewalle, J.: Cryptanalysis of 2.5 round of IDEA (extended abstract), Technical Report ESAC-COSIC Technical Report 93/1, Department Of Electrical Engineering, Katholieke Universiteit Leuven (March 1993)
[5] Daemen, J., Govaerts, R., Vandewalle, J.: Weak Keys of IDEA. In: Stinson, D.R. (ed.) CRYPTO 1993. LNCS, vol. 773, pp. 224–231. Springer, Heidelberg (1994)
[6] Demirci, H.: Square-like Attacks on Reduced Rounds of IDEA. In: Nyberg, K., Heys, H.M. (eds.) SAC 2002. LNCS, vol. 2595, pp. 147–159. Springer, Heidelberg (2003)
[7] Demirci, H., Selçuk, A.A., Türe, E.: A New Meet-in-the-Middle Attack on the IDEA Block Cipher. In: Matsui, M., Zuccherato, R.J. (eds.) SAC 2003. LNCS, vol. 3006, pp. 117–129. Springer, Heidelberg (2004)
[8] Hawkes, P.: Differential-Linear Weak Key Classes of IDEA. In: Nyberg, K. (ed.) EUROCRYPT 1998. LNCS, vol. 1403, pp. 112–126. Springer, Heidelberg (1998)
[9] Hawkes, P., O'Connor, L.: On Applying Linear Cryptanalysis to IDEA. In: Kim, K.-c., Matsumoto, T. (eds.) ASIACRYPT 1996. LNCS, vol. 1163, pp. 105–115. Springer, Heidelberg (1996)

[10] Junod, P.: New attacks against reduced-round versions of IDEA. In: Gilbert, H., Handschuh, H. (eds.) FSE 2005. LNCS, vol. 3557, pp. 384–397. Springer, Heidelberg (2005)
[11] Lai, X., Massey, J.L.: A Proposal for a New Block Encryption Standard. In: Damgård, I.B. (ed.) EUROCRYPT 1990. LNCS, vol. 473, pp. 389–404. Springer, Heidelberg (1991)
[12] Lai, X., Massey, J.L., Murphy, S.: Markov Ciphers and Differential Cryptanalysis. In: Davies, D.W. (ed.) EUROCRYPT 1991. LNCS, vol. 547, pp. 17–38. Springer, Heidelberg (1991)
[13] Meier, W.: On the Security of the IDEA Block Cipher. In: Helleseth, T. (ed.) EUROCRYPT 1993. LNCS, vol. 765, pp. 371–385. Springer, Heidelberg (1994)
[14] Nakahara Jr., J., Barreto, P.S.L.M., Preneel, B., Vandewalle, J., Kim, H.Y.: Square Attacks Against Reduced-Round PES and IDEA Block Ciphers. In: 23rd Symposium on Information Theory in the Benelux. Louvain-la-Neuve, pp. 187–195 (2002)
[15] Nakahara, J., Preneel, B., Vandewalle, J.: The Biryukov-Demirci attack on reduced-round versions of IDEA and MESH block ciphers. In: Wang, H., Pieprzyk, J., Varadharajan, V. (eds.) ACISP 2004. LNCS, vol. 3108, pp. 98–109. Springer, Heidelberg (2004)

Improved Related-Key Impossible Differential Attacks on Reduced-Round AES-192

Wentao Zhang[1], Wenling Wu[2], Lei Zhang[2], and Dengguo Feng[2]

[1] State Key Laboratory of Information Security,
Graduate University of Chinese Academy of Sciences, Beijing 100049, P.R. China
zhangwt@gucas.ac.cn

[2] State Key Laboratory of Information Security,
Institute of Software, Chinese Academy of Sciences, Beijing 100080, P.R. China
{wwl,zhanglei1015,feng}@is.iscas.ac.cn

Abstract. In this paper, we present several new related-key impossible differential attacks on 7- and 8-round AES-192, following the work of Eli Biham et al. [6] and Jakimoski et al. [10]. We choose another relation of the related keys, start attacks from the very beginning(instead of the third round in [6]) so that the data and time complexities are improved largely, and only two related keys are needed instead of 32 in the attacks of [6]. Furthermore, we point out and correct an error in [6] when they attacked 8-round AES-192, then present our revised attacks. Finally, we give a new related-key differential attack on 7-round AES-192, which mainly uses a property of MixColumns operation of AES.

Keywords: AES, cryptanalysis, related-key differentials, impossible differentials.

1 Introduction

AES [12] supports 128-bit block size with three different key lengths (128,192,and 256 bits). Because of its importance, it's very necessary to constantly reevaluate the security of AES under various cryptanalytic techniques. In this paper, we study the security of 192-bit key version of AES(AES-192) against the related-key impossible differential attack.

Related-key attacks [2] allow an attacker to obtain plaintext-ciphertext pairs by using related(but unknown) keys. The attacker first searches for possible weaknesses of the encryption and key schedule algorithms, then choose appropriate relation between keys and make two encryptions using the related keys expecting to derive the unknown key information. Differential cryptanalysis [1] analyzes the evolvement of the difference between a pair of plaintexts in the following round outputs in an iterated cipher. Related-key differential attack [11] combines the above two cryptanalytic techniques together, and it studies the development of differences in two encryptions under two related keys. Furthermore, impossible differential attacks [3] use differentials that hold with probability 0(or non-existing differentials) to eliminate wrong key material and leave the right key candidate. In this case, the combined attack is called related-key impossible differential attack.

© Springer-Verlag Berlin Heidelberg 2007

There are several impossible differential attacks on AES [4,7,8]. The best impossible differential attack on AES-192 is on 7 rounds [7].

If we view the expanded keys as a sequence of words, then the key schedule of AES-192 applies a non-linear transformation once every six words, whereas the key schedules of AES-128 and AES-256 apply non-linear transformations once every four words. This property brings better and longer related-key differentials of AES-192, so directly make AES-192 more susceptible to related-key attacks than AES-128 and AES-256. In the last few years, the security of AES-192 against related-key attacks has drawn much attention from cryptology researchers [5,6,9,10]. In [10], Jakimoski et al. presented related-key impossible differential attacks on 7- and 8-round AES-192. Following the work of [10], Biham et al.[6] gave several new related-key impossible differential attacks also on 7- and 8-round AES-192, which substantially improved the data and time complexity of those in [10]. Both in [5] and [9], the security of AES-192 against the related-key boomerang attack were studied. The best known related-key attack on AES-192 hitherto is due to Biham et al.[5], and it is applicable to a 9-round variant of AES-192.

Table 1. Comparison of Some Previous Attacks with Our New Attacks

Source	Number of Rounds	Data Complexity	Time Complexity	Number of Keys	Attack Type
Ref.[5]	9	2^{86} RK-CP	2^{125}	256	RK Rectangle
Ref.[7]	7	2^{92}CP	2^{186}	1	Imp.Diff
Ref.[10]	7	2^{111}RK-CP	2^{116}	2	RK Imp.Diff
	8	2^{88}RK-CP	2^{183}	2	
Ref.[6]	7	2^{56}RK-CP	2^{94}	32	RK Imp.Diff
	8	$2^{68.5}$RK-CP	$*2^{184}$	32	
	8	2^{92}RK-CP	$*2^{159}$	32	
	8	2^{116}RK-CP	$*2^{134}$	32	
This paper	7	2^{52}RK-CP	2^{80}	2	RK Imp.Diff
	8	$2^{64.5}$RK-CP	2^{177}	2	
	8	2^{88}RK-CP	2^{153}	2	
	8	2^{112}RK-CP	2^{136}	2	
This paper	7	2^{37}RK-CP	2^{145}	2	RK Diff

RK – Related-key, CP – Chosen plaintext,
Time complexity is measured in encryption units.

In this paper, we present several new related-key impossible differential attacks, following the work of [6] and [10]. In [6], the authors expressed: “We note that due to the special structure of the key schedule, the best round to start the attack with is round 2 of the original AES”. However, we can choose another key difference of the two related keys, and start the attacks from the very beginning, so greatly improve the data and time complexities of their attacks, and only two related keys are needed instead of 32 in [6]. Furthermore, we point out an error in [6] when they attacked 8-round AES-192, and then present our attacks. Lastly, we present a new related-key differential attack on 7-round AES-192, which can be regarded as a byproduct during the preparation of this paper, and it utilizes another property, ie., the specific property of Mixcolumns operation of AES.

Amongst our results, we reduce the data complexity of our attack by a factor of 2^4 and time complexity by a factor of 2^{14} compared with that in [6] for 7-round AES-192. The results are also improved in various degrees for several attacks on 8-round AES-192. Finally, a new related-key differential attack on 7-round AES-192 is presented, it needs more time, but the data complexity is reduced greatly. In all the attacks, only two related keys are needed. We summarize our results along with some previously known ones against AES-192 in Table 1. Note that there is an error in [6](which will be explained later), so the evaluated time complexities on 8 rounds are not right in [6], we mark them with "$*$".

Here is the outline of this paper. In Section 2, we give a brief description of AES. In Section 3, we choose another key difference of the two related keys, and present the corresponding subkeys difference of AES-192. Then a new 5.5-round related-key impossible differential is gained. Using this impossible differential, Section 4 presents an attack on 7-round AES-192; Section 5 firstly describes an error in [6], then presents three variants of our attacks on 8-round AES-192. Section 6 presents a new related-key differential attack on 7-round AES-192. Finally, Section 7 summarizes this paper.

2 Description of AES

The AES algorithm encrypts or decrypts data blocks of 128 bits by using keys of 128, 192 or 256 bits. The 128-bit plaintexts and the intermediate state are treated as byte matrices of size 4×4. Each round is composed of four operations:

- SubBytes(SB): applyinging the S-box on each byte.
- ShiftRows(SR): cyclically shifting each row (the i'th row is shifted by i bytes to the left, $i = 0, 1, 2, 3$).
- MixColumns(MC): multiplication of each column by a constant 4×4 matrix M over the field $GF(2^8)$, where M is

$$\begin{pmatrix} 02\ 03\ 01\ 01 \\ 01\ 02\ 03\ 01 \\ 01\ 01\ 02\ 03 \\ 03\ 01\ 01\ 02 \end{pmatrix}$$

  And the inverse of M is

$$\begin{pmatrix} 0e\ 0b\ 0d\ 09 \\ 09\ 0e\ 0b\ 0d \\ 0d\ 09\ 0e\ 0b \\ 0b\ 0d\ 09\ 0e \end{pmatrix}$$

- AddRoundKey(ARK): XORing the state and a 128-bit subkey.

The MixColumns operation is omitted in the last round, and an additional AddRoundKey operation is performed before the first round. We also assume that the MixColumns operation is omitted in the last round of the reduced-round variants.

The number of rounds is dependent on the key size, 10 rounds for 128-bit keys, 12 for 192-bit keys and 14 for 256-bit keys.

The key schedule of AES-192 takes the 192-bit secret key and expands it to thirteen 128-bit subkeys. The expanded key is a linear array of 4-byte words and is denoted by $G[4 \times 13]$. Firstly, the 192-bit secret key is divided into 6 words $G[0], G[1] \dots G[5]$. Then, perform the following:

For $i = 6, \dots 51$, do
If $(i \equiv 0 \mod 6)$, then $G[i] = G[i-6] \oplus SB(G[i-1] \lll 8) \oplus RCON[i/6]$
Else $G[i] = G[i-6] \oplus G[i-1]$

where $RCON[\cdot]$ is an array of predetermined constants, $\lll$ denotes rotation of a word to the left by 8 bits.

2.1 Notations

In the rest of this paper, we will use the following notations: x_i^I denotes the input of the i'th round, while x_i^S, x_i^R, x_i^M and x_i^O respectively denote the intermediate values after the application of SubBytes, ShiftRows, MixColumns and AddRoundKey operations of the i'th round. Obviously, $x_{i-1}^O = x_i^I$ holds.

Let k_i denote the subkey in the i'th round, and the initial whitening subkey is k_0. In some cases, the order of the MixColumns and the AddRoundKey operation in the same round is changed, which is done by replacing the subkey k_i with an equivalent subkey w_i, where $w_i = MC^{-1}(k_i)$.

Let $(x_i)_{Col(l)}$ denote the l'th column of x_i, where $l = 0, 1, 2, 3$. And $(x_i)_j$ the j'th byte of $x_i (j = 0, 1, \dots 15)$, here Column(0) includes byte 0,1,2 and 3, Column(1) includes byte 4,5,6 and 7, etc.

3 A 5.5-Round Related-Key Impossible Differential of AES-192

We choose a new difference between two related keys as follows:

$((a,0,0,0),(0,0,0,0),(a,0,0,0),(0,0,0,0),(0,0,0,0),(0,0,0,0))$.

Hence, the subkey differences in the first 8 rounds are as presented in Table 2, which will be used in our attacks later.

Throughout the attacks in our paper, we assume that the subkey differences are as presented in Table 2. A 5.5-round related-key impossible differential can be built like in [6] and [10]. Firstly, a 4.5-round related-key differential with probability 1 in the forward direction, then a 1-round related-key differential with probability 1 in the reverse direction, where the intermediate differences contradict each other. The 5.5-round related-key impossible differential is:

$$\Delta x_1^M = ((0,0,0,0),(0,0,0,0),(a,0,0,0),(a,0,0,0)) \xrightarrow{\text{5.5-round}}$$
$$\Delta x_6^O = ((?,?,?,?),(?,?,?,?),(?,?,?,?),(0,0,0,b))$$

Table 2. Subkey Differences Required for the Attacks in this paper

Round(i)	$\Delta k_{i,Col(0)}$	$\Delta k_{i,Col(1)}$	$\Delta k_{i,Col(2)}$	$\Delta k_{i,Col(3)}$
0	$(a,0,0,0)$	$(0,0,0,0)$	$(a,0,0,0)$	$(0,0,0,0)$
1	$(0,0,0,0)$	$(0,0,0,0)$	$(a,0,0,0)$	$(a,0,0,0)$
2	$(0,0,0,0)$	$(0,0,0,0)$	$(0,0,0,0)$	$(0,0,0,0)$
3	$(a,0,0,0)$	$(0,0,0,0)$	$(0,0,0,0)$	$(0,0,0,0)$
4	$(0,0,0,0)$	$(0,0,0,0)$	$(a,0,0,0)$	$(a,0,0,0)$
5	$(a,0,0,0)$	$(a,0,0,0)$	$(a,0,0,0)$	$(a,0,0,0)$
6	$(a,0,0,b)$	$(0,0,0,b)$	$(a,0,0,b)$	$(0,0,0,b)$
7	$(a,0,0,b)$	$(0,0,0,b)$	$(a,0,c,b)$	$(a,0,c,0)$
8	$(0,0,c,b)$	$(0,0,c,0)$	$(a,0,c,b)$	$(a,0,c,0)$

a, b and c are non-zero byte differences.

The above differential holds with probability 0, where a and b are non-zero values, "?" denotes any value.

The first 4.5-round differential is obtained as follows: the input difference Δx_1^M is canceled by the subkey difference of the first round. The zero difference Δx_2^I is preserved through all the operations until the AddRoundKey operation of the third round, as the subkey difference of the second round is zero. Thus, we can get $\Delta x_4^I = \Delta k_3 = ((a,0,0,0),(0,0,0,0),(0,0,0,0),(0,0,0,0))$, where only one byte is active. Then the next three operations in the fourth round will convert the active byte to a complete column of active bytes, and after the AddRoundKey operation with k_4, we will get $\Delta x_4^O = ((N,N,N,N),(0,0,0,0),(a,0,0,0),(a,0,0,0))$, where N denotes a non-zero byte(possibly distinct). Applying the SubBytes and ShiftRows operations of the 5'th round, Δx_4^O will evolve into $\Delta x_5^R = ((N,0,0,0),(0,0,0,N),(N,0,N,0),(N,N,0,0))$, where only one byte is active both in Column 0 and Column 1. Hence, $\Delta x_5^M = ((N,N,N,N),(N,N,N,N),(?,?,?,?),(?,?,?,?))$. Finally, after the key addition with k_5, we can get $\Delta x_5^O = ((?,N,N,N),(?,N,N,N),(?,?,?,?),(?,?,?,?))$.

The second differential ends after the 6'th round with output difference $\Delta x_6^O = ((?,?,?,?),(?,?,?,?),(?,?,?,?),(0,0,0,b))$. When rolling back this difference through the AddRoundKey and MixColumn operations, we get the difference in the last column of Δx_6^R is zero. Hence, $\Delta x_6^I = ((?,0,?,?),(?,?,0,?),(?,?,?,0),(0,?,?,?))$. It's obvious that $\Delta x_6^I = \Delta x_5^O$ with probability 1. However, we can see that $(\Delta x_5^O)_1$ is a non-zero byte in the first 4.5-round differential, while $(\Delta x_6^I)_1$ is a zero byte in the second differential, this is a contradiction.

Using the above 5.5-round impossible differential, we can start our attacks from the very beginning. Hence, compared with the attacks in [6], there is no unknown bytes in the key difference, whereas one unknown byte in the key difference in [6]. Therefore, our attacks can proceed with one less byte guessing, which makes the time complexity reduced at least by a factor of 2^7. Moreover, only two related keys are needed, which makes the data complexity reduced by a factor of 2^4 immediately.

4 A 7-Round Related-Key Impossible Differential Attack

Using the above impossible differential, we can attack a 7-round variant of AES-192.

At first, we assume that the values of a, b, c are all known, ie., we have two related keys K_1 and K_2 with the required subkey differences listed in Table 2. We will deal with the conditions on the related keys to achieve these subkey differences at the end of this section.

The attack procedure is quite similar to that in [6]. However, the attack complexity will be reduced significantly.

4.1 The Attack Procedure

Precomputation: For all the 2^{64} possible pairs of values of the last two columns of x_1^M(ie.,$(x_1^M)_{Col(2)}$ and $(x_1^M)_{Col(3)}$) both with difference $(a, 0, 0, 0)$, compute the 8 byte values in bytes 1, 2, 6, 7, 8, 11, 12, and 13 of plaintext P. Store the pairs of 8-byte values in a hash table H_p indexed by the XOR differences in these bytes.

The algorithm is as follows:

1. Generate two pools S_1 and S_2 of m plaintexts each, such that for each plaintext pair $P_1 \in S_1$ and $P_2 \in S_2$, $P_1 \oplus P_2 = ((a, ?, ?, 0), (0, 0, ?, ?), (?, 0, 0, ?), (?, ?, 0, 0))$, where ? denotes any byte value.
2. Ask for the encryption of the pool S_1 under K_1, and of the pool S_2 under K_2. Denote the ciphertexts of the pool S_1 by T_1, and the encrypted ciphertexts of the pool S_2 by T_2.
3. For all ciphertexts $C_2 \in T_2$, compute $C_2^* = C_2 \oplus ((0, 0, 0, 0), (0, 0, 0, 0), (0, 0, 0, 0), (a, 0, 0, 0))$.
4. Insert all the ciphertexts $C_1 \in T_1$ and the values $\{C_2^* | C_2 \in T_2\}$ into a hash table indexed by bytes 6,9,and 12.
5. Guess the value of the subkey byte $(k_7)_3$ and perform the followings:
   (a) Initialize a list A of the 2^{64} possible values of the bytes 1, 2, 6, 7, 8, 11, 12, and 13 of k_0.
   (b) Decrypt the byte $(x_7^O)_3$ in all the ciphertexts to get the intermediate values before the subkey addition in the 6'th round.
   (c) For every pair C_1, C_2^* in the same bin of the hash table, check whether the corresponding intermediate values calculated in Step 5(b) are equal. If no, discard the pair.
   (d) For every remaining pair C_1, C_2^*, consider the corresponding plaintext pair and compute $P_1 \oplus P_2$ in the eight bytes 1, 2, 6, 7, 8, 11, 12, and 13. Denote the resulting value by P'.
   (e) If the bin P' in H_p is nonempty, access this bin. For each pair (x, y) in that bin remove from the list A the values $P_1 \oplus x$ and $P_1 \oplus y$, where P_1 is restricted to eight bytes (plaintext bytes 1, 2, 6, 7, 8, 11, 12, and 13).
   (f) If A is not empty, output the values in A along with the guess of $(k_7)_3$.

4.2 Analysis of the Attack Complexity

From the two pools of m plaintexts each, m^2 possible ciphertexts pairs (C_1, C_2^*) can be derived. After the filtering in step 4, there remains about $2^{-24}m^2$ pairs in each bin of the hash table. In Step 5, we have an additional 8-bit filtering for every possible value of $(k_7)_3$ separately, so about $2^{-32}m^2$ pairs will remain for a given subkey guess of $(k_7)_3$. Each pair deletes one subkey candidate on average, and there are 2^{64} subkey candidates in all, so the expected number of remaining subkeys is $2^{64}(1-1/2^{64})^{m'}$ in step 5(f). If $m' = 2^{70}$, the expected number is about $e^{-20} = 2^{-28.85}$, and we can expect that only the right subkey will remain. Hence, we get the value of $64 + 8 = 72$ subkey bits. In order to derive $m' = 2^{70}$, we need $m = 2^{51}$ chosen plaintexts in each of the two pools. So the data complexity of the attack is 2^{52} chosen plaintexts.

The time complexity is dominated by Step 5(e). In this step, $m' = 2^{70}$ pairs are analyzed, leading to one memory access on average to H_p and one memory access to A. This step is repeated 2^8 times (once for one guess of $(k_7)_3$). Therefore, the time complexity is 2^{79} memory accesses, which is equivalent to about 2^{73} encryptions. The precomputation requires about 2^{62} encryptions and the required memory is about 2^{69} bytes.

In the above attack, we assumed that the values of a, b and c are known. Here,the value a can be chosen by the attacker. The value b is the result of application of SubBytes operation, so there are 127 possible values of b given the value of a. And the attack can proceed without knowing the value of c. Hence,we only need to repeat the attack for all the possible values of b. Therefore,the total time complexity is multiplied by 2^7, the data and memory complexity remain unchanged.

To sum up, the total complexity of the above attack is as follows: The data complexity is 2^{52} chosen plaintexts, the time complexity is 2^{80} encryptions, and the required memory is 2^{69} bytes.

5 Three 8-Round Related-Key Impossible Differential Attacks

In this section, we point out an error in [6] when they attacked 8-round AES-192. Then, present our own attacks.

5.1 An Error in the 8-Round Attacks of [6]

In Section 4 of [6], the authors presented three variant attacks on 8-round AES-192. In the first version, 13 key bytes are guessed: bytes 0,2,3,5,6,7,8,9,10,12,13 and 15 of k_9, and byte 11 of w_8(ie., $(w_8)_{3,2}$ in [6]). For peeling off the last round, only those ciphertext pairs which have zero difference (before the subkey addition with k_9) in the remaining four bytes 1,4,11 and 14 are treated. Then rolling back through the ShiftRows and SubBytes operations, difference in the four bytes of Column 1 are all zero, ie.,$(\Delta x_9^I)_{Col(1)} = (0, 0, 0, 0)$. This property still holds

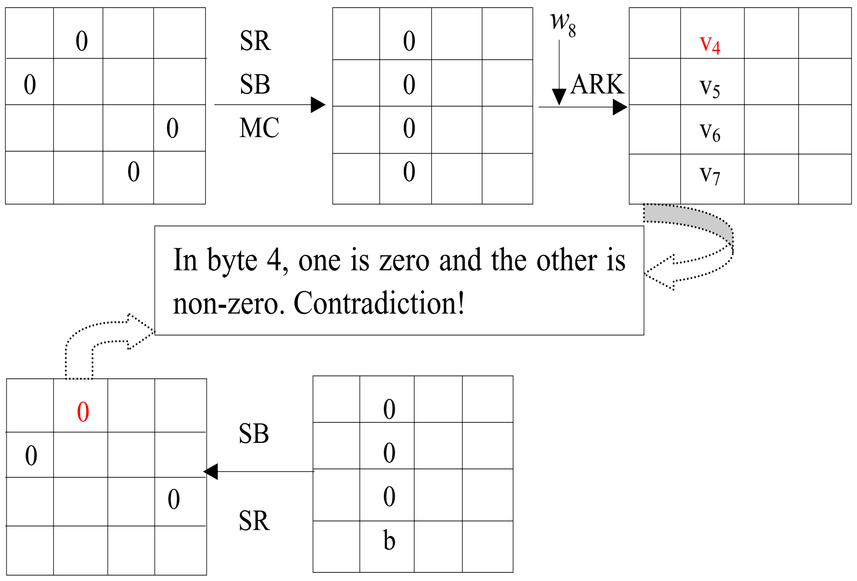

SR
SB
MC
w_8
ARK
v_4
v_5
v_6
v_7
In byte 4, one is zero and the other is non-zero. Contradiction!
SB
SR
0
b

Fig. 1. An Error in Ref.[6]

when applying the MixColumns operation in round 8. Then applying the subkey addition with w_8 in round 8, difference in the four bytes of Column 1 will equal to the corresponding byte of Δw_8. Especially, we can get $(\Delta x_8^R)_4 = (\Delta w_8)_4$. Note that $(\Delta w_8)_4$ is determined by $(\Delta k_8)_{Col(1)}$ which have one non-zero byte and three zero bytes, thus applying the inverse operation of MixColumns to $(\Delta k_8)_{Col(1)}$, we can deduce that $(\Delta x_8^R)_4 = (\Delta w_8)_4$ is a non-zero byte. Then we can conclude that $(\Delta x_8^I)_4$ is a non-zero byte immediately.

However, when applying the impossible differential from round 2 to round 7 in [6], the attacker must filter a certain amount of plaintext pairs to satisfy the requirement of the differential, especially $(\Delta x_8^I)_4 = (\Delta x_7^O)_4$ equals to zero. This is a contradiction, which is emphasized in Figure 1, in which the first pane denotes the intermediate state before the subkey addition with k_9, the last pane denotes the input state of round 8, and the other panes denote the intermediate state between them. For simplicity, let v_4, v_5, v_6, v_7 denote byte 4,5,6,7 of w_8 respectively. Here, we will use x_8^W to denote the intermediate value after the application of AddRoundKey operation with w_8 in round 8. From the above analysis, in order to correctly use the 5.5-round differential, we must filter a certain amount of pairs which satisfy that $(\Delta x_8^W)_4 = (\Delta w_8)_4$ to make $(\Delta x_8^I)_4 = 0$. Then, after the Mixcolumns operation in round 8, this byte has relation with all the four bytes in Column 1. Thus, it seems that four more key bytes of k_9 should be guessed in order to calculate the byte difference $(\Delta x_8^W)_4$ from ciphertext pairs. However, we can deal with this problem by guessing only one more key byte, and treat only ciphertext pairs that have zero difference in the other three bytes.

For the second and third version of the attacks on 8-round AES-192 in [6], there exists similar errors. And we will adopt the same technique to reduce the amount of key bytes guess.

In the following, we will present our attacks on 8-round AES-192.

5.2 Our Attacks on 8-Round AES-192

In the following, we will give three variants of attacks on 8-round AES-192, which are all based on the 7-round attack in Section 4. As in [6], the main difference between them is a data-time trade-off. In all the 8-round attacks, we guess part of the last round subkey k_8, peel off the last round and apply the 7-round attack. In order to reduce the amount of key material guess, we also change the order of the MixColumns and the AddRoundKey operations in the 7'th round, this is done by replacing the subkey k_7 with an equivalent subkey w_7. And we use x_7^W to denote the intermediate value after the application of AddRoundKey operation with w_7 in the 7'th round.

If a pair of ciphertexts satisfy the condition that $(\Delta x_6^O)_{Col(3)} = (0, 0, 0, b)$, then after the SubBytes and ShiftRows operations of the 7'th round, bytes 6,9 and 12 of Δx_7^R must be zero. Next, applying the key addition with w_7, we can get $(\Delta x_7^W)_6 = (\Delta w_7)_6$,$(\Delta x_7^W)_9 = (\Delta w_7)_9$, and $(\Delta x_7^W)_{12} = (\Delta w_7)_{12}$.

In order to satisfy the above conditions and guess less subkey material, we treat only ciphertext pairs that have certain properties. Take for example, to make $(\Delta x_7^W)_6 = (\Delta w_7)_6$, we only choose ciphertext pairs that satisfy $(\Delta x_7^O)_{Col(1)} = (z_4, 0, 0, 0)$, where z_4 is uniquely determined by $(\Delta w_7)_6$ to make the above condition hold, ie., $MC^{-1}(z_4, 0, 0, 0) = (?, ?, (\Delta w_7)_6, ?)$. Similarly, we can decide the values of bytes z_8(or z_{11}) and z_{12}, which make $(\Delta x_7^W)_9 = (\Delta w_7)_9$ and $(\Delta x_7^W)_{12} = (\Delta w_7)_{12}$ respectively.

The attack can be performed in one out of three possible ways.

The First Attack. Guess bytes 0,1,2,4,5,7,8,10,11,12,13,14, and 15 of k_8, partially decrypt these bytes in the last round. These subkey bytes allow us to partially decrypt the last round in Columns 0,1 and 2. And we only treat ciphertext pairs that have zero difference in the remaining 3 bytes(before the key addition with k_8, the same below). This condition allows us to use 2^{-24} of the possible ciphertext pairs. Then the difference Δx_8^I is known, we first check whether the difference in byte 12 equals to z_{12}. This filtering is done using a 8-bit condition, which makes the remaining ciphertext pairs satisfy the condition that byte 12 in Δx_6^O is zero. Next, applying the inverse of MixColumns operation to Column 1 and Column 2 of Δx_8^I, calculate byte 6 and 9 of Δx_7^W and check whether they are equal to $(w_7)_6$ and $(w_7)_9$ respectively. This filtering thus makes bytes 13 and 14 of Δx_6^O equal to zero too, and uses a 16-bit condition. Then, guess byte 3 of w_7 and continue partial decryption to find out whether $(\Delta x_6^O)_{15} = b$ holds, which is done using a 8-bit condition. After this filtering, the remaining ciphertext pairs can be used to discard wrong subkey guesses like in the 7-round attack.

In this variant of the attack, we guess a total of 112 subkey bits. And a portion of $2^{-24-8-16-8} = 2^{-56}$ of the pairs can be used in the attack to discard the wrong subkey guesses.

Here we can use the differential properties of the key schedule algorithm. The value of b can be determined by a and $(k_5)_{12} = (k_8)_4 \oplus (k_8)_{12}$, the value of c can be determined by b and $(k_7)_7 = (k_8)_{11} \oplus (k_8)_{15}$.

About $2^{63.5}$ plaintexts in each pool are needed to derive about $2^{63.5+63.5-56} = 2^{71}$ data pairs for every guess of the 112-bit key material guess in the last two rounds. Each pair discards one possible value for the eight bytes guess of subkey k_0 on average. Therefore, the probability that some wrong subkey guess remains is at most $2^{64}e^{-128} \approx 2^{-120}$, and the expected number of subkey suggestions is approximately $2^{-120}2^{112} = 2^{-8}$. Hence, with a high probability only the right value will remain. The data complexity of this attack is about $2^{64.5}$ chosen plaintexts. The time complexity is about $2^{71} \times 2^{112}/2^6 = 2^{177}$ and the required memory is about 2^{69} bytes.

The Second Attack. Guess bytes 0,1,4,7,10,11,12,13,14, and 15 of k_8. And treat only ciphertext pairs that have zero difference in the remaining 6 bytes. This condition allows us to use only 2^{-48} of the possible ciphertext pairs. Then the difference Δx_8^I is known, we first check whether the difference in bytes 11 and 12 are z_{11} and z_{12} respectively. This filtering is done using a 16-bit condition, which makes the remaining ciphertext pairs satisfy the conditon that bytes 12 and 13 in Δx_6^O are all zero. Next, calculate the difference in byte 6 of Δx_7^W and check whether it equals to $(w_7)_6$. This filtering uses a 8-bit condition, and makes the remaining pairs also satisfy that byte 14 of Δx_6^O equals to zero. Then, guess byte 3 of w_7 and continue partial decryption to find out whether $(\Delta x_6^O)_{15} = b$ holds. This is done using a 8-bit condition. After this filtering, the remaining ciphertext pairs can be used to discard wrong subkey guesses.

In this variant of the attack, we guess a total of 88 subkey bits. But only a portion of 2^{-80} of the pairs can be used in the attack to discard wrong subkey guesses.

As in the first attack, the value of b and c can also be determined by a and subkey guess of k_8.

Choose a pool of 2^{64} plaintexts which differ only at the eight bytes 1, 2, 6, 7, 8, 11, 12 and 13, and having all possible values in these bytes. Choose two such pools S_1 and S_2, such that for each plaintext pair $P_1 \in S_1$ and $P_2 \in S_2$, $P_1 \oplus P_2 = ((a, ?, ?, 0), (0, 0, ?, ?), (?, 0, 0, ?), (?, ?, 0, 0))$. Encrypt S_1 and S_2 under the two related keys each. Then, we can derive $2^{64} \times 2^{64} = 2^{128}$ pairs of plaintexts using 2^{65} chosen plaintexts, call such two pools a structure.

About 2^{23} structures are needed to get about 2^{71} data pairs which can be used to delete wrong subkey guesses. Hence, the data complexity of this attack is about 2^{88} chosen plaintexts. The time complexity is about $2^{71} \times 2^{88}/2^6 = 2^{153}$.

The Third Attack. Guess bytes 0,7,10,13,4,8, and 12 of k_8, partially decrypt these bytes in the last round. And treat only ciphertext pairs that have zero difference in the remaining 9 bytes. This condition allows us to use only 2^{-72}

of the possible ciphertext pairs. Then the difference Δx_8^I is known, we check whether the difference in bytes 4,8 and 12 are z_4, z_8 and z_{12} respectively. This filtering is done using a 24-bit condition. Thus, the remaining ciphertext pairs all satisfy that bytes 12,13,14 in Δx_6^O are all zero. Then, guess byte 3 of w_7 and continue partial decryption to find out whether $(\Delta x_6^O)_{15} = b$ holds. This is done using a 8-bit condition. After this filtering, the remaining ciphertext pairs can be used to discard wrong subkey guesses.

In this attack variant, we guess only 64 subkey bits. But only a portion of 2^{-104} of the pairs can be used in the attack. This leads to a relatively high data complexity, but to a lower time complexity.

The value of b can be determined by a and subkey guess of k_8. Hence, we only need to repeat the attack for all the 127 possible values of c.

About 2^{47} structures are needed to get about 2^{71} data pairs which can be used to delete wrong subkey guesses. Hence, the data complexity of this attack is about 2^{112} chosen plaintexts. The time complexity is about $2^7 \times 2^{71} \times 2^{64}/2^6 = 2^{136}$.

6 A New Related-Key Differential Attack on 7-Round AES-192

In this section, we present a new related-key differential attack on 7-round AES-192, which mainly uses the specific property of Mixcolumns operation of AES.

The subkey differences are also as presented in Table 2, and we will change the order of the MixColumns and the AddRoundKey operations in the 6'th round. Submit two plaintexts P_1 and P_2 for encryption under the two related keys respectively. Similar to the analysis in Section 3, if$\Delta x_1^M = ((0,0,0,0),(0,0,0,0),$ $(a,0,0,0),(a,0,0,0))$, then we can conclude that Δx_5^R must have the form of $((N,0,0,0),(0,0,0,N)$,$(N,0,N,0),(N,N,0,0))$, where only one byte is active both in Column 0 and Column 1. Considering Column 1, we have $(\Delta x_5^R)_{Col(1)} =$ $(0,0,0,N)$, then after the following MixColumns operation, we can get that $(\Delta x_5^M)_4$ =$(\Delta x_5^M)_5$ holds with probability 1 because of the specific MixColumns operation of AES.

In order to calculate the values of bytes 4 and 5 in Δx_5^M, we need to guess 10 subkey bytes: bytes 0,1,4,7,10,11,13, and 14 of k_7, and bytes 1,4 of w_6.

The attack procedure has many similarities to the above attacks, including the precomputation, the initial plaintexts selection and encryption, and the initialization of the list A. The difference only consists in the filtering condition of data. For each guess of the 10 key bytes, we will calculate the two values of $(x_5^M)_4$ and $(x_5^M)_5$ for each plaintext, then check whether $(\Delta x_5^M)_4$ =$(\Delta x_5^M)_5$ for each data pairs. If not, we can use it to delete the corresponding key guess from A as in the above attacks.

From the two pools of m plaintexts each, m^2 possible ciphertext pairs can be derived. For every possible guess of the 10 key bytes, about $m' = (1 - 2^{-8})m^2$ pairs can be used to delete the wrong subkey guesses of k_0. Each pair deletes one subkey candidate on average, so the probability that some wrong subkey guess remains is at most $2^{64}(1-1/2^{64})^{m'}$. If $m' = 2^{71}$, the expected number is about 2^{-120}, and we can expect that only the right subkey will remain. Hence, we get the value of $80 + 64 = 144$ subkey bits. In order to derive $m' = 2^{71}$, we need about $m = 2^{36}$ chosen plaintexts in each of the two pools. So the data complexity of the attack is about 2^{37} chosen plaintexts.

The time complexity is about $2^{71} \times 2^{80}/2^{6} = 2^{145}$. And the required memory is also about 2^{69} bytes.

Here, b can be calculated from a and $(k_5)_{12}$=$(k_7)_0 \oplus (k_7)_4$, and c can be calculated from b and $(k_7)_7$.

To sum up, the total complexity of the above attack is as follows: The data complexity is 2^{37} chosen plaintexts, the time complexity is 2^{145} encryptions, and the required memory is 2^{69} bytes.

Compared with the 7-round attack in Section 4, the data complexity is decreased, but the time complexity is increased greatly. Nevertheless, the attack uses another different point of AES, ie., the specific property of the MixColumns operation.

7 Summary

Up to now, better results are achieved against reduced-round AES-192 using related-key cryptanalysis in contrast to other non-related-key cryptanalysis approaches. This fact reflects some weaknesses of the key schedule algorithm of AES-192.

In this paper, we improved the attack results presented in [6] and [10] through choosing a new difference of the related keys. Furthermore, we detected an error in [6] when they attacked 8-round AES-192, then presented our revised attacks. The new chosen related-key difference made our attack start from the very beginning instead of the third round as in [6]. Hence, the number of unknown bytes in the subkey differences become less, which makes the attack complexity improved largely in this paper. The comparison of our attack results and those in [6] and [10] can be found in Table 1.

We also present a new related-key differential attack on 7-round AES-192, which mainly utilizes the property of MixColumns operation, but it is a pity that we can't extend the attack to 8-round at present, we wish that this point may be used in further attacks on AES.

Acknowledgment

We would like to thank anonymous referees for their helpful comments and suggestions. The research presented in this paper is supported by the National Natural Science Foundation of China(No.60373047, 90604036); and the National Basic Research 973 Program of China No.2004CB318004.

References

1. Biham, E., Shamir, A.: Differential cryptanalysis of DES-like cryptosystems. Journal of Cryptology 4(1), 3–72 (1991)
2. Biham, E.: New Types of Cryptanalytic Attacks Using Related Keys. Journal of Cryptology 7(4), 229–246 (1994)
3. Biham, E., Biryukov, A., Shamir, A.: Cryptanalysis of Skipjack Reduced to 31 Rounds. In: Stern, J. (ed.) EUROCRYPT 1999. LNCS, vol. 1592, pp. 12–23. Springer, Heidelberg (1999)
4. Biham, E., Keller, N.: Cryptanalysis of Reduced Variants of Rijndael, in Official public comment for Round 2 of the AES development effort (2000) Available at `http://csrc.nist.gov/encryption/aes/round2/conf3/aes3papers.html`
5. Biham, E., Dunkelman, O., Keller, N.: Related-Key Boomerang and Rectangle Attacks. In: Cramer, R.J.F. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 507–525. Springer, Heidelberg (2005)
6. Biham, E., Dunkelman, O., Keller, N.: Related-Key Impossible Differential Attacks on 8-Round AES-192. In: Pointcheval, D. (ed.) CT-RSA 2006. LNCS, vol. 3860, pp. 21–33. Springer, Heidelberg (2006)
7. Phan, R.C.-W.: Impossible Differential Cryptanalysis of 7-round Advanced Encryption Standard (AES). Information Processing Letters 91(1), 33–38 (2004)
8. Cheon, J.H., Kim, M., Kim, K., Lee, J.-Y., Kang, S.: Improved Impossible Differential Cryptanalysis of Rijndael and Crypton. In: Kim, K.-c. (ed.) ICISC 2001. LNCS, vol. 2288, pp. 39–49. Springer, Heidelberg (2002)
9. Hong, S., Kim, J., Kim, G., Lee, S., Preneel, B.: Related-Key Rectangle Attacks on Reduced Versions of SHACAL-1 and AES-192. In: Gilbert, H., Handschuh, H. (eds.) FSE 2005. LNCS, vol. 3557, pp. 368–383. Springer, Heidelberg (2005)
10. Jakimoski, G., Desmedt, Y.: Related-Key Differential Cryptanalysis of 192-bit Key AES Variants. In: Matsui, M., Zuccherato, R.J. (eds.) SAC 2003. LNCS, vol. 3006, pp. 208–221. Springer, Heidelberg (2004)
11. Kelsey, J., Schneier, B., Wagner, D.: Related-Key Cryptanalysis of 3-WAY, Biham-DES, CAST, DES-X, NewDES, RC2, and TEA. In: Han, Y., Quing, S. (eds.) ICICS 1997. LNCS, vol. 1334, pp. 233–246. Springer, Heidelberg (1997)
12. National Institute of Standards and Technology. Advanced Encryption Standard (AES), FIPS Publication 197 (November 26, 2001) Available at `http://csrc.nist.gov/encryption/aes`

Related-Key Rectangle Attack on the Full SHACAL-1

Orr Dunkelman[1,⋆], Nathan Keller[2,⋆⋆], and Jongsung Kim[3,4,⋆⋆⋆]

[1] Computer Science Department, Technion.
Haifa 32000, Israel
orrd@cs.technion.ac.il
[2] Einstein Institute of Mathematics, Hebrew University.
Jerusalem 91904, Israel
nkeller@math.huji.ac.il
[3] ESAT/SCD-COSIC, Katholieke Universiteit Leuven
Kasteelpark Arenberg 10, B-3001 Leuven-Heverlee, Belgium
Kim.Jongsung@esat.kuleuven.be
[4] Center for Information Security Technologies(CIST), Korea University
Anam Dong, Sungbuk Gu, Seoul, Korea
joshep@cist.korea.ac.kr

Abstract. SHACAL-1 is a 160-bit block cipher with variable key length of up to 512-bit key based on the hash function SHA-1. It was submitted to the NESSIE project and was accepted as a finalist for the 2nd phase of the evaluation.

In this paper we devise the first known attack on the full 80-round SHACAL-1 faster than exhaustive key search. The related-key differentials used in the attack are based on transformation of the collision-producing differentials of SHA-1 presented by Wang et al.

1 Introduction

In 1993, NIST has issued a standard hash function called Secure Hash Algorithm (FIPS-180) [25]. Later this version was named SHA-0, as NIST published a small tweak to this standard called SHA-1 in 1995. Both SHA-0 and SHA-1 are based on padding the message and dividing it to blocks of 512 bits, and then iteratively compressing those blocks into a 160-bit digest. Recently, NIST has published three more standard hash functions as part of FIPS-180: SHA-256, SHA-384 and SHA-512. Each of the new hash functions has a digest size corresponding

⋆ The author was supported by the Clore scholarship programme and by the Israel MOD Research and Technology Unit.

⋆⋆ The author was supported by the Adams fellowship.

⋆⋆⋆ This author was financed by a Ph.D grant of the Katholieke Universiteit Leuven and by the Korea Research Foundation Grant funded by the Korean Government (MOEHRD) (KRF-2005-213-D00077) and supported by the Concerted Research Action (GOA) Ambiorics 2005/11 of the Flemish Government and by the European Commission through the IST Programme under Contract IST2002507932 ECRYPT.

© Springer-Verlag Berlin Heidelberg 2007

to its number, i.e., SHA-256 has a 256-bit digest, etc. After the publication of these hash functions, NIST has issued another hash function SHA-224 that has a digest size of 224 bits.

Both SHA-0 and SHA-1 were subjected to a great deal of analysis [11]. In the last two years there was a major progress in the attacks on both of the hash functions. This progress included finding a collision in SHA-0, and devising an algorithm that can find a collision in SHA-1 in less than 2^{63} SHA-1 applications [3,4,30,32,28]. The new techniques are based on finding good differentials of the compressing function of SHA-1 and combining them with some novel plaintext modification techniques.

It was suggested to use the compression function of SHA-1 as a block cipher [13]. Later this suggestion was named SHACAL-1 and submitted to the NESSIE project [14]. SHACAL-1 is a 160-bit block cipher with variable key length (0–512 bits) and 80 rounds based on the compression function of SHA-1. The cipher, was selected as a NESSIE finalist, but was not selected for the NESSIE portfolio [22].

Due to the structure of SHACAL-1, differentials of SHA-1 correspond to related-key differentials of SHACAL-1. Hence, it seems natural that some of the techniques used in the new attacks on SHA-1 can be converted into a related-key attack on SHACAL-1. We show that this is indeed the case. The differentials found in the attacks devised in [30] can be converted into high probability related-key differentials of SHACAL-1.

After transforming the collision producing differentials into related-key differentials, we use them in a related-key rectangle attack [8,15,18]. The resulting attack succeeds to attack the full 80-round SHACAL-1 using four related-keys faster than exhaustive key search.

The related-key rectangle technique was used in previously published attacks on SHACAL-1 [15,18] and was by far the most successful technique to attack the cipher. The best previously known attack on the cipher based on this technique was applicable up to 70 rounds of SHACAL-1. Our results extend these previously known results by using improved differentials and improved attack techniques.

We note that the best known attack on SHACAL-1 that does not use related-keys is a rectangle attack on 49-round SHACAL-1 [7]. A comparison of the known attacks along with our new results on SHACAL-1 is presented in Table 1.

This paper is organized as follows: In Section 2 we describe the block cipher SHACAL-1. In Section 3 we describe the previously known results on SHACAL-1 and the relevant results on SHA-1. In Section 4 we give a short description of the related-key rectangle attack. In Section 5 we present the new attacks on the full SHACAL-1. Section 6 explores the differences between our attacks on SHACAL-1 and other works on SHA-1. The appendix contains the differentials used in the attack. Finally, Section 7 summarizes the paper.

2 Description of SHACAL-1

SHACAL-1 [14] is a 160-bit block cipher supporting variable key lengths (0–512 bits). It is based on the compression function of the hash function SHA-1 [25].

Table 1. Summary of Our Results and Previously Known Results on SHACAL-1

Attack & Source	Number of Keys	Number of Rounds	Rounds	Complexity Data	Complexity Time
Differential [20]	1	41	0–40	2^{141} CP	2^{491}
Amplified Boomerang [20]	1	47	0–46	$2^{158.5}$ CP	$2^{508.4}$
Rectangle [7]	1	47	0–46	$2^{151.9}$ CP	$2^{482.6}$
Rectangle [7]	1	49	29–77	$2^{151.9}$ CC	$2^{508.5}$
Related-Key Rectangle [18]	2	59	0–58	$2^{149.7}$ RK-CP	$2^{498.3}$
Related-Key Rectangle [15]	4	70	0–69	$2^{151.8}$ RK-CP	$2^{500.1}$
Related-Key Rectangle (Section 5.2)	4	80	0–79	$2^{159.8}$ RK-CP	$2^{420.0}$
Related-Key Rectangle (Section 5.2)	4	80	0–79	$2^{153.8}$ RK-CP	$2^{501.2}$

Complexity is measured in encryption units.
CP — Chosen Plaintexts, CC — Chosen Ciphertexts, RK — Related-Key.

The cipher has 80 rounds (also referred as steps) grouped into four types of 20 rounds each.[1]

The 160-bit plaintext is divided into five 32-bit words – A, B, C, D and E. We denote by X_i the value of word X before the ith round, i.e., the plaintext P is divided into A_0, B_0, C_0, D_0 and E_0, and the ciphertext is composed of $A_{80}, B_{80}, C_{80}, D_{80}$ and E_{80}.

In each round the words are updated according to the following rule:

$$\begin{aligned}
A_{i+1} &= W_i + ROTL_5(A_i) + f_i(B_i, C_i, D_i) + E_i + K_i \\
B_{i+1} &= A_i \\
C_{i+1} &= ROTL_{30}(B_i) \\
D_{i+1} &= C_i \\
E_{i+1} &= D_i
\end{aligned}$$

where + denotes addition modulo 2^{32}, $ROTL_j(X)$ represents rotation to the left by j bits, W_i is the round subkey, and K_i is the round constant.[2] There are three different functions f_i, selected according to the round number:

$$\begin{array}{llll}
f_i(X,Y,Z) = f_{if} = & (X\&Y)|(\neg X\&Z) & & 0 \le i \le 19 \\
f_i(X,Y,Z) = f_{xor} = & (X \oplus Y \oplus Z) & & 20 \le i \le 39, 60 \le i \le 79 \\
f_i(X,Y,Z) = f_{maj} = & ((X\&Y)|(X\&Z)|(Y\&Z)) & & 40 \le i \le 59
\end{array}$$

In [14] it is strongly advised to use keys of at least 128 bits, even though shorter keys are supported. The first step in the key schedule algorithm is to pad the supplied key into a 512-bit key. Then, the 512-bit key is expanded into eighty 32-bit subkeys (or a total of 2560 bits of subkey material). The

[1] To avoid confusion, we adopt the common notations for rounds. In [14] the notation step stands for round, where round is used for a group of 20 steps.

[2] This time we adopt the notations of [14], and alert the reader of the somewhat confusing notations.

expansion is done in a linear manner using a linear feedback shift register (over $GF(2^{32})$).

The key schedule is as follows: Let $M_0, \ldots, M_{15}$ be the 16 key words (32 bits each). Then the round subkeys $W_0, \ldots, W_{79}$ are computed by the following algorithm:

$$W_i = \begin{cases} M_i & 0 \leq i \leq 15 \\ (W_{i-3} \oplus W_{i-8} \oplus W_{i-14} \oplus W_{i-16}) \lll 1 & 16 \leq i \leq 79. \end{cases}$$

3 Previous Results

A preliminary differential and linear analysis of the properties of the compression function of SHA-1 as a block cipher is presented in [13]. The found differentials are relatively short (10 rounds) and have probabilities varying between 2^{-13} and 2^{-26} (depending on the round functions).

In [26] these differentials are improved, and 20-round differentials with probability 2^{-41} are presented. In [20] another set of differentials of SHACAL-1 is presented, including a 30-round differential with probability 2^{-130}.

In [24] an algorithm for identifying whether two SHACAL-1 encryptions use a pair of related keys is presented. The attack is based on finding slid pairs. Once a slid pair is encountered, the attacker can determine whether the two encryptions have related keys. The attack requires about 2^{96} encryptions under each of the two keys to find a slid pair.

In [20] a 21-round differential for rounds 0–20 and a 15-round differential for rounds 21–35 are combined to devise an amplified boomerang distinguisher [16] for 36-round SHACAL-1. This distinguisher is used to attack 39-round SHACAL-1 using $2^{158.5}$ chosen plaintexts and about $2^{250.8}$ 39-round SHACAL-1 encryptions. The attack is based on guessing (or trying) the subkeys of the three additional rounds, and then checking whether the distinguisher succeeds. This approach is further extended to attack 47-round SHACAL-1 before exhaustive key search becomes faster than this attack. Another attack presented in [20] is a differential attack on 41-round SHACAL-1. The success of these attacks was questioned and resolved in [7].

Besides resolving the problems with previous attacks, in [7] a rectangle attack on 49-round SHACAL-1 is presented. The attack requires $2^{151.9}$ chosen plaintexts, and has a running time equivalent to $2^{508.5}$ 49-round SHACAL-1 encryptions.

In [18] a related-key rectangle attack with two keys is presented against 59-round SHACAL-1. This attack has a data complexity of $2^{149.7}$ related-key chosen plaintexts and has a time complexity of $2^{498.3}$ 59-round SHACAL-1 encryptions. This attack is improved in [15] to a related-key rectangle attack with four keys on 70-round SHACAL-1. The improved attack has a data complexity of $2^{151.8}$ related-key chosen plaintexts, and a time complexity of $2^{500.1}$ 70-round SHACAL-1 encryptions.

4 Related-Key Boomerang and Related-Key Rectangle Attacks

In this section we briefly describe the related-key rectangle attack. First, we outline the boomerang and the rectangle attacks and describe related-key differentials. Then, we describe the combination that forms into the related-key rectangle attack.

4.1 The Rectangle Attack

The rectangle attack [5] is an improved variant of the amplified boomerang attack [16] that has evolved from the boomerang attack presented in [27]. We first describe the boomerang attack, and then show the transformation into amplified boomerang/rectangle attacks.

The main idea behind the boomerang attack is to use two short differentials with high probabilities instead of one long differential with a low probability. We assume that a block cipher $E:\{0,1\}^n \times \{0,1\}^k \rightarrow \{0,1\}^n$ can be described as a cascade $E = E_1 \circ E_0$, such that for E_0 there exists a differential $\alpha \rightarrow \beta$ with probability p, and for E_1 there exists a differential $\gamma \rightarrow \delta$ with probability q.

The distinguisher is based on the following boomerang process:

- Ask for the encryption of a pair of plaintexts (P_1, P_2) such that $P_1 \oplus P_2 = \alpha$ and denote the corresponding ciphertexts by (C_1, C_2).
- Calculate $C_3 = C_1 \oplus \delta$ and $C_4 = C_2 \oplus \delta$, and ask for the decryption of the pair (C_3, C_4). Denote the corresponding plaintexts by (P_3, P_4).
- Check whether $P_3 \oplus P_4 = \alpha$.

The boomerang attack uses the first characteristic $(\alpha \rightarrow \beta)$ for E_0 with respect to the pairs (P_1, P_2) and (P_3, P_4), and uses the second characteristic $(\gamma \rightarrow \delta)$ for E_1 with respect to the pairs (C_1, C_3) and (C_2, C_4).

For a random permutation the probability that the last condition is satisfied is 2^{-n}. For E, the probability that the pair (P_1, P_2) is a right pair with respect to the first differential $(\alpha \rightarrow \beta)$ is p. The probability that both pairs (C_1, C_3) and (C_2, C_4) are right pairs with respect to the second differential is q^2. If all these are right pairs, then $E_1^{-1}(C_3) \oplus E_1^{-1}(C_4) = \beta = E_0(P_3) \oplus E_0(P_4)$. Thus, with probability p, $P_3 \oplus P_4 = \alpha$. The total probability of this quartet of plaintexts and ciphertexts to satisfy the boomerang conditions is $(pq)^2$.

The attack can be mounted for all possible β's and γ's simultaneously (as long as $\beta \neq \gamma$). Therefore, a right quartet for E is encountered with probability no less than $(\hat{p}\hat{q})^2$, where:

$$\hat{p} = \sqrt{\sum_{\beta} \Pr{}^2[\alpha \rightarrow \beta]}, \qquad \text{and} \qquad \hat{q} = \sqrt{\sum_{\gamma} \Pr{}^2[\gamma \rightarrow \delta]}.$$

The complete analysis is given in [27,5,6].

As the boomerang attack requires adaptive chosen plaintexts and ciphertexts, many of the techniques that were developed for using distinguishers in key recovery attacks can not be combined with the boomerang attack. This led to the introduction of chosen plaintext variants of the boomerang attack called the *amplified boomerang attack* [16] and the *rectangle attack* [5]. The transformation of the boomerang attack into a chosen plaintext attack is quite standard, and is achieved by birthday-paradox arguments. The key idea behind the transformation is to encrypt many plaintext pairs with input difference α, and to look for quartets that conform to the requirements of the boomerang process.

The rectangle (or the amplified boomerang) process is as follows:

– Ask for the encryption of many pairs of plaintexts $(P, P \oplus \alpha)$.
– Search two pairs of plaintexts $(P_1, P_2), (P_3, P_4)$, and their corresponding ciphertexts (C_1, C_2) and (C_3, C_4), respectively, satisfying:
  - $P_1 \oplus P_2 = P_3 \oplus P_4 = \alpha$
  - $C_1 \oplus C_3 = C_2 \oplus C_4 = \delta$

Given the same decomposition of E as before, and the same basic differentials, the analysis in [5] shows that out of N plaintext pairs, the number of right quartets is expected to be $N^2 2^{-n} \hat{p}^2 \hat{q}^2$. We note, that the main reduction in the probability follows from the fact that unlike the boomerang attack, in the rectangle attack the event $E_0(P_1) \oplus E_0(P_3) = \gamma$ occurs with probability 2^{-n} even when all the differentials hold.

4.2 Related-Key Differentials

Related-key differentials [17] were used for cryptanalysis several times in the past. Recall, that a regular differential deals with some plaintext difference ΔP and a ciphertext difference ΔC such that

$$\Pr_{P,K}[E_K(P) \oplus E_K(P \oplus \Delta P) = \Delta C]$$

is high enough (or zero [2]).

A related-key differential is a triplet of a plaintext difference ΔP, a ciphertext difference ΔC, and a key difference ΔK, such that

$$\Pr_{P,K}[E_K(P) \oplus E_{K \oplus \Delta K}(P \oplus \Delta P) = \Delta C]$$

is useful (high enough or zero).

4.3 Related-Key Rectangle Attack

The related-key rectangle attack was introduced in [18,15], and independently in [8].

Let us assume that we have a related-key differential $\alpha \to \beta$ of E_0 under a key difference ΔK_{ab} with probability p. Assume also that we have another related-key differential $\gamma \to \delta$ for E_1 under a key difference ΔK_{ac} with probability q.

The related-key rectangle process involves four different unknown (but related) keys — K_a, $K_b = K_a \oplus \Delta K_{ab}$, $K_c = K_a \oplus \Delta K_{ac}$, and $K_d = K_a \oplus \Delta K_{ab} \oplus \Delta K_{ac}$. The attack is performed by the following algorithm:

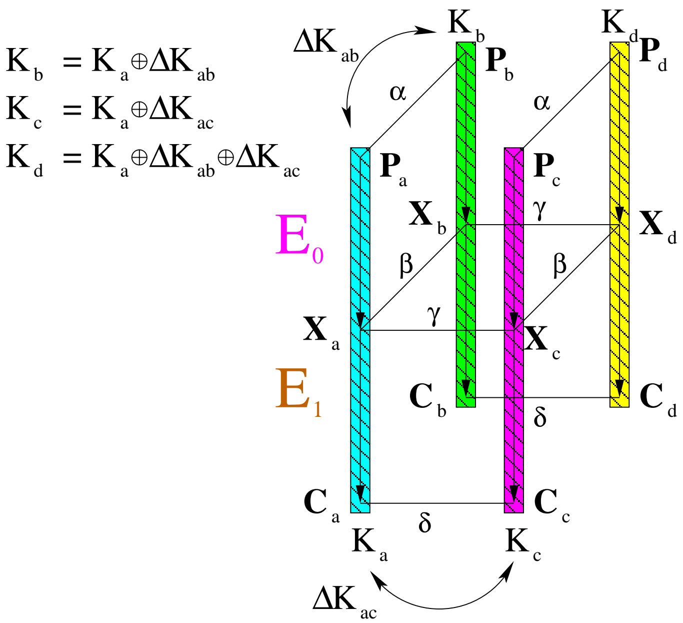


Fig. 1. A Related-Key Rectangle Quartet

- Choose N plaintext pairs $(P_a, P_b = P_a \oplus \alpha)$ at random and ask for the encryption of P_a under K_a and of P_b under K_b. Denote the set of these pairs by S.
- Choose N plaintext pairs $(P_c, P_d = P_c \oplus \alpha)$ at random and ask for the encryption of P_c under K_c and P_d under K_d. Denote the set of these pairs by T.
- Search a pair of plaintexts $(P_a, P_b) \in S$ and a pair of plaintexts $(P_c, P_d) \in T$, and their corresponding ciphertexts (C_a, C_b) and (C_c, C_d), respectively, satisfying:
  - $P_a \oplus P_b = P_c \oplus P_d = \alpha$
  - $C_a \oplus C_c = C_b \oplus C_d = \delta$

See Figure 1 for an outline of such a quartet.

The attack can use many differentials for E_0 and E_1 simultaneously (just like in a regular rectangle attack) as long as all related-key differentials used in E_0 have the same key difference ΔK_{ab} and the same input difference α and as long as all related-key differentials used in E_1 have the same key difference ΔK_{ac} and the same output difference δ.

The analysis of the related-key rectangle attack is similar to the one of the rectangle attack. Starting with N plaintext pairs in S and N plaintext pairs in T, we expect to find $N^2 2^{-n} (\hat{p}\hat{q})^2$ right quartets in $S \times T$. For a random permutation the number of "right quartets" is about $N^2 2^{-2n}$, so as long as $\hat{p}\hat{q} > 2^{-n/2}$ we can use the related-key rectangle attack to distinguish between a random permutation and the attacked cipher. This distinguisher can be later used for a key recovery attack.

We note that a related-key boomerang attack can be constructed similarly to the related-key rectangle attack. The full analysis can be found in [8]. The related-key boomerang and rectangle techniques were used to attack reduced round variants of AES, IDEA, SHACAL-1, and SHACAL-2 and the full KASUMI and COCONUT98 [8,9,15,19,18].

In the case of SHACAL-1, the key schedule algorithm is linear. Therefore, given a key difference, all subkey differences are known, and can be easily used in the related-key model.

5 Related-Key Rectangle Attack on the Full SHACAL-1

Our attack on SHACAL-1 is based on a 69-round related-key distinguisher. In the attack on the full SHACAL-1, we try all the possible subkeys of the remaining 11 rounds, and decrypt all the ciphertexts. Then, the 69-round distinguisher is applied. We improve the time complexity of the attack by partially decrypting only 8 rounds, and then use the early abort approach to reduce the number of values that are decrypted through the remaining three more rounds, before the attack is applied. It is expected that for the right guess of the subkey of the last 11 rounds, the distinguisher would be more successful than for a wrong guess. Thus, we can use this distinguisher to identify (to some extent) the right subkey.

5.1 69-Round Related-Key Distinguisher

We decompose 69-round SHACAL-1 into two sub-ciphers: E_0 that contains the first 34 rounds of SHACAL-1 (rounds 0–33), and E_1 that contains the remaining 35 rounds (rounds 34–68).

We have transformed the collision producing differentials of SHA-1 presented in [30] into related-key differentials for each of the two sub-ciphers. The first related-key differential (for E_0) has probability 2^{-41}, and by fixing two bits of the plaintexts and using several differentials simultaneously for E_0 we obtain $\hat{p} = 2^{-38.5}$. The second related-key differential (for E_1) has probability 2^{-39}, and by using several differentials simultaneously for E_1 we obtain $\hat{q} = 2^{-38.3}$. The differentials are presented in Appendix A.

Combining these two differentials together leads to a 69-round related-key rectangle distinguisher with probability $2^{-80} \cdot \hat{p}\hat{q} = 2^{-156.8}$, i.e., given N related-key chosen plaintext pairs, we expect $N^2 \cdot 2^{-160} \cdot (\hat{p}\hat{q})^2$ right quartets. Hence, given $2^{157.8}$ related-key chosen plaintext pairs, we expect four right rectangle quartets, while for a random cipher only $2^{-4.4}$ are expected.

5.2 The Key Recovery Attack

The basic approach for a key recovery attack is to guess the subkey of the last 11 rounds, partially decrypt all ciphertexts, and apply the distinguisher for the remaining 69 rounds. Such an approach can be improved using the fact that in every round, only a small part of the intermediate value is substantially changed, while most of the value is only shifted. The attack is based on the *early abort* technique which is widely used [11,12]. In this technique, once a pair/quartet does not satisfy the required differences/properties it is excluded from further analysis.

In the description of the attack algorithm we use the following notations: X_A denotes the value of word A in X. Similarly, $Y_{D,E}$ denotes words D and E of Y, etc. Let ΔX_i denote the difference in word X before round i, i.e., ΔA_{70} is the difference in word A before round 70, and after round 69. Also, let e_i be the 32-bit word composed of 31 0's and 1 in the ith place. We use $e_{i,j}$ to denote $e_i \oplus e_j$ and $e_{i,j,k} = e_{i,j} \oplus e_k$, etc. We also denote the set of possible values of ΔA_{70} given that the second differential is satisfied by S'.

We observe that even if we partially decrypt only 8 rounds, we still have a filtering condition on the quartets: Since $\Delta D_{72} = ROTL_{30}(\Delta A_{69})$ and $\Delta E_{72} = ROTL_{30}(\Delta B_{69})$, we can check whether the difference in these words corresponds to the output difference in words A and B of the second differential. In addition, we observe that we can extend the second differential by a truncated differential of one additional round. There are only $324 = 2^{8.3}$ possible ΔA_{70} values in S', hence, there are only 324 possible values for ΔC_{72} in case the second differential holds.

Using these observations, we can get a filtering of $64 + 23.7 = 87.7$ bits for every pair in the end of round 71, or a filtering of 175.4 bits in total. Since the attack starts with $2^{315.6}$ quartets, we expect that $2^{140.3}$ quartets pass the filtering for any given subkey guess of rounds 72–79. We then guess the subkey of round 71 and compute ΔE_{71} that is equal to ΔC_{69} if the differential holds to obtain an additional 64-bit filtering on the remaining quartets. After this filtering only $2^{76.3}$ quartets remain for each subkey guess. Then we continue by guessing the subkeys of rounds 70 and 69. As a result, the time complexity of the attack drops rapidly, while the data complexity remains unchanged.

The algorithm of the attack is as follows:

1. **Data Collection Phase**
   (a) Ask for the encryption of $2^{157.8}$ pairs of plaintexts (P_a, P_b), where $P_b = P_a \oplus \alpha$, where P_a and P_b satisfy the restrictions described in Appendix A, and where P_a is encrypted under K_a and P_b is encrypted under K_b.
   (b) Ask for the encryption of $2^{157.8}$ pairs of plaintexts (P_c, P_d), where $P_c = P_d \oplus \alpha$, where P_c and P_d satisfy the restrictions described in Appendix A, and where P_c is encrypted under K_c and P_d is encrypted under K_d.
2. **Partial Decryption**
   (a) For each guess of the subkey of rounds 72–79:
      i. Partially decrypt all ciphertexts (under the corresponding keys).
      ii. Find all pairs of partially decrypted ciphertexts (C_a, C_c), such that $C_{a_{C,D,E}} \oplus C_{c_{C,D,E}} \in S$, where C_a is encrypted under K_a, C_c

is encrypted under K_c and $S = \{(x, y, z) : ROTL_{30}(x) \in S', ROTL_{30}(y) = \delta_A = 0, ROTL_{30}(z) = \delta_B = e_2\}$.

iii. For each such pair (C_a, C_c), let P_a and P_c be the corresponding plaintexts. Let $P_b = P_a \oplus \alpha$ and $P_d = P_c \oplus \alpha$, and let C_b and C_d be the corresponding ciphertexts, respectively.

iv. If $C_{b_{C,D,E}} \oplus C_{d_{C,D,E}} \in S$ pass the quartet (P_a, P_b, P_c, P_d) for a further analysis.

(b) **Partial Decryption of Round 71:** For each guess of the subkey of round 71:

i. Partially decrypt all the remaining quartets (under the corresponding keys) and denote the resulting intermediate values by (C'_a, C'_b, C'_c, C'_d).

ii. For each of the remaining quartets, check whether $C'_{a_E} \oplus C'_{c_E} = \delta_C = 0$ and discard all the quartets that do not satisfy the equation.

iii. For each of the remaining quartets, check whether $C'_{b_E} \oplus C'_{d_E} = \delta_C = 0$ and discard all the quartets that do not satisfy the equation.

(c) **Partial Decryption of Round 70:** For each guess of the subkey of round 70:

i. Partially decrypt all the remaining quartets (under the corresponding keys) and denote the resulting intermediate values by $(C''_a, C''_b, C''_c, C''_d)$.

ii. For each of the remaining quartets, check whether $C''_{a_E} \oplus C''_{c_E} = \delta_D = 0$ and discard all the quartets that do not satisfy the equation.

iii. For each of the remaining quartets, check whether $C'_{b_E} \oplus C'_{d_E} = \delta_D = 0$ and discard all the quartets that do not satisfy the equation.

(d) **Partial Decryption of Round 69:** For each guess of the subkey of round 69:

i. Partially decrypt all the remaining quartets (under the corresponding keys) and denote the resulting intermediate values by $(C'''_a, C'''_b, C'''_c, C'''_d)$.

ii. For each of the remaining quartets, check whether $C'''_{a_E} \oplus C'''_{c_E} = \delta_E = e_1$ and discard all the quartets that do not satisfy the equation.

iii. For each of the remaining quartets, check whether $C'''_{b_E} \oplus C'''_{d_E} = \delta_E = e_1$ and discard all the quartets that do not satisfy the equation.

iv. Pass all the remaining quartets to further analysis.

(e) **Further Analysis:** If for this subkey guess only one quartet is suggested (or no quartets are suggested) discard the subkey guess. If the subkey is not discarded, exhaustively search all possible values for the remaining 160 subkey bits for the correct key.

5.3 Analysis of the Key Recovery Attack

The time complexity of Step 1 is $2^{159.8}$ encryptions. The time complexity of Step 2(a) is $\frac{8}{80} \cdot 2^{256} \cdot 2^{159.8} = 2^{412.5}$ SHACAL-1 encryptions. Steps 2(b)–2(e) are repeated for each subkey guess, i.e., 2^{256} times. For a given subkey guess, Step 2(b) consists of $2^{141.3} \cdot 2^{32}$ partial decryptions of one SHACAL-1 round. This is equivalent to $2^{141.3} \cdot 2^{32} \cdot \frac{1}{80} = 2^{167.0}$ full SHACAL-1 encryptions. Thus, the total time complexity of Step 2(b) is about $2^{256} \cdot 2^{167.0} = 2^{423.0}$ SHACAL-1 encryptions.

There is an improvement of the time complexity by a factor of 8 based on the observation that the difference in the most significant bit is not affected by the actual key value. Thus, it is possible to guess in Step 2(a) the entire subkey of rounds 74–79, and all but most significant bits of the subkeys of rounds 72–73. This does not affect the ability to compute the difference in the most significant bits of the words D^{72} and E^{72}. Similarly, in Step 2(b) is is sufficient to guess the 31 least significant bits of K_{71} in order to find the difference in the three words: C^{71}, D^{71}, and E^{71}.

In Step 2(c) there is again no need to guess the entire subkey to deduce the difference in the most significant bit. However, in order for the partial decryption to be done correctly, the real value of the B^{70} has to be computed. Thus, in this step we guess the most significant bit of K^{73} along with the 31 least significant bits of K^{70}. The same is done also in Step 2(d), where the most significant bit of K^{72} is guessed along with the 31 least significant bits of K^{69}. We note that the improved variant guess $11 \cdot 32 - 3 = 349$ subkey bits during the entire attack.

The time complexities of the other steps are relatively smaller. Hence, the total data complexity of the attack is $2^{159.8}$ related-key chosen plaintexts encrypted under four keys, and the time complexity is $2^{420.0}$ SHACAL-1 encryptions. The memory requirement of the attack is about $2^{159.8}$ memory blocks of 320 bits, required for storing the large amount of data.

We note that a different approach may be used in our attack. We can remove the last three rounds of the second differential to increase its probability by a factor of 2^6, resulting in a 66-round related-key rectangle distinguisher with probability $2^{-80} \cdot \hat{p} \cdot \hat{q} = 2^{-150.8}$. The resultant distinguisher requires $2^{151.8}$ related-key chosen plaintext pairs (P_a, P_b) and (P_c, P_d) each to produce four right plaintext quartets (while for a random cipher about $2^{-16.4}$ quartets that satisfy the rectangle conditions are expected). Then, we apply partial decryptions of rounds 69-79, 68, 67 and 66 in Steps 2(a), 2(b), 2(c) and 2(d), respectively, and then run the final exhaustive search for the remaining 64-bit keys in Step 2(e).

The time complexity of Step 2(a) in this case is $2^{152.8+352} \cdot (11/80) = 2^{501.9}$ SHACAL-1 encryptions. In this attack we can derive the set S in Step 2-(a) for the filtering of quartets, which has $2^{70.8}$ elements, and thus the number of remaining quartets after this step is about $(2^{151.9} \cdot 2^{-160+70.8})^2 = 2^{125.2}$. It follows that Step 2(b) takes about $2^{126.2} \cdot 2^{352+32} \cdot (1/80) = 2^{503.9}$ SHACAL-1 encryptions. Compared to Steps 2(a) and 2(b), the followed steps have quite small time complexities. Hence, this full-round attack on SHACAL-1 works with a data complexity of $2^{153.8}$ related-key chosen plaintexts encrypted under four related keys and with a time complexity of $2^{501.9} + 2^{503.9} = 2^{504.2}$ SHACAL-1 encryptions. Again, a factor 8 in the time complexity can be improved using the observation about the most significant bits, i.e., the attack's time complexity is $2^{501.2}$ SHACAL-1 encryptions.

6 Differences Between Attacking SHA-1 and SHACAL-1

While it may seem that any attack on SHA-1 can be easily transformed into an attack on SHACAL-1, and vice versa, this is not exactly the case. Investigating the recent attacks on SHA-1 in [3,4,30], it seems that these attacks heavily rely on the fact that the attacker can control some of the bits that enter the nonlinear operations. This way, the collision-producing differentials have much higher probability than the respective related-key differentials we use. We can impose conditions on the keys (increasing the probabilities of the related-key differentials), but then our attack would be applicable only for such keys, i.e., a weak key class.

Another difference between the attacks on SHA-1 and our attack is the fact that the collision attacks can iteratively fix the values they use, i.e., using message modification techniques or neutral bits. This enables the collision producing attacks to use shorter differential than ours (as these attacks actually start the probabilistic process in a much later step).

There is another difference between the two cases. While in the case of encryption (SHACAL-1), we have to deal with each block of message independently, collision attacks on the hash function can use multiple blocks. For example, the attacker can treat messages that detoured the differential in a very late step, by respective changes to the second block of the message. This fact allows the collision search to use shorter differentials (this time from the end point), thus, increasing the success probability.

Another problem our attack faces is the dual representation of XOR and additive differentials. As we have less control on the encryption process than the collision attacks have on the compression process, it is less useful for us to consider the differentials using the dual representation. Again, for exploiting the advantage of the additive differentials in the related-key differentials, we must fix some of the key bits, resulting again in a weak key class.

7 Summary and Conclusions

In this paper we converted the differentials of the compression function of SHA-1 presented by Wang et al. to related-key differentials of the block cipher SHACAL-1. Then we used the related-key rectangle technique to devise the first known attack on the full 80-round SHACAL-1.

We also discussed the possibility of converting other techniques used in the attacks on SHA-1 to attack SHACAL-1, and concluded that such conversion will result in an attack applicable only to a weak key class of SHACAL-1.

Our attack improves by far the previously known results, that were able to attack up to 70 rounds of the cipher, and demonstrates the power of the related-key rectangle technique. However, the result is still highly theoretical and a practical attack on the full SHACAL-1 seems out of reach at this stage. We note that keys shorter than 420 bits can still be considered secure, as for these keys the time complexity of our attack is greater than exhaustive key search.

References

1. Biham, E.: New Types of Cryptanalytic Attacks Using Related Keys. Journal of Cryptology 7(4), 229–246 (1994)
2. 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)
3. Biham, E., Chen, R.: Near-Collisions of SHA-0. In: Franklin, M. (ed.) CRYPTO 2004. LNCS, vol. 3152, pp. 290–305. Springer, Heidelberg (2004)
4. Biham, E., Chen, R., Joux, A., Carribault, P., Lemuet, C., Jalby, W.: Collisions of SHA-0 and Reduced SHA-1. In: Cramer, R.J.F. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 36–57. Springer, Heidelberg (2005)
5. Biham, E., Dunkelman, O., Keller, N.: The Rectangle Attack – Rectangling the Serpent. In: Pfitzmann, B. (ed.) EUROCRYPT 2001. LNCS, vol. 2045, pp. 340–357. Springer, Heidelberg (2001)
6. Biham, E., Dunkelman, O., Keller, N.: New Results on Boomerang and Rectangle Attacks. In: Daemen, J., Rijmen, V. (eds.) FSE 2002. LNCS, vol. 2365, pp. 1–16. Springer, Heidelberg (2002)
7. Biham, E., Dunkelman, O., Keller, N.: Rectangle Attacks on 49-Round SHACAL-1. In: Johansson, T. (ed.) FSE 2003. LNCS, vol. 2887, pp. 22–35. Springer, Heidelberg (2003)
8. Biham, E., Dunkelman, O., Keller, N.: Related-Key Boomerang and Rectangle Attacks. In: Cramer, R.J.F. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 507–525. Springer, Heidelberg (2005)
9. 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)
10. Biham, E., Shamir, A.: Differential Cryptanalysis of the Data Encryption Standard. Springer, Heidelberg (1993)
11. Chabaud, F., Joux, A.: Differential Collisions in SHA-0. In: Krawczyk, H. (ed.) CRYPTO 1998. LNCS, vol. 1462, pp. 56–71. Springer, Heidelberg (1998)
12. Ferguson, N., Kelsey, J., Lucks, S., Schneier, B., Stay, M., Wagner, D., Whiting, D.: Improved Cryptanalysis of Rijndael. In: Schneier, B. (ed.) FSE 2000. LNCS, vol. 1978, pp. 213–230. Springer, Heidelberg (2001)
13. Handschuh, H., Knudsen, L.R., Robshaw, M.J.: Analysis of SHA-1 in Encryption Mode. In: Naccache, D. (ed.) CT-RSA 2001. LNCS, vol. 2020, pp. 70–83. Springer, Heidelberg (2001)
14. Handschuh, H., Naccache, D.: SHACAL. In: preproceedings of NESSIE first workshop, Leuven (2000)
15. Hong, S., Kim, J., Kim, G., Lee, S., Preneel, B.: Related-Key Rectangle Attacks on Reduced Versions of SHACAL-1 and AES-192. In: Gilbert, H., Handschuh, H. (eds.) FSE 2005. LNCS, vol. 3557, pp. 368–383. Springer, Heidelberg (2005)

16. Kelsey, J., Kohno, T., Schneier, B.: Amplified Boomerang Attacks Against Reduced-Round MARS and Serpent. In: Schneier, B. (ed.) FSE 2000. LNCS, vol. 1978, pp. 75–93. Springer, Heidelberg (2001)
17. Kelsey, J., Schneier, B., Wagner, D.: Key-Schedule Cryptoanalysis of IDEA, G-DES, GOST, SAFER, and Triple-DES. In: Koblitz, N. (ed.) CRYPTO 1996. LNCS, vol. 1109, pp. 237–251. Springer, Heidelberg (1996)
18. Kim, J., Kim, G., Hong, S., Lee, S., Hong, D.: The Related-Key Rectangle Attack — Application to SHACAL-1. In: Wang, H., Pieprzyk, J., Varadharajan, V. (eds.) ACISP 2004. LNCS, vol. 3108, pp. 123–136. Springer, Heidelberg (2004)
19. Kim, J., Kim, G., Lee, S., Lim, J., Song, J.: Related-Key Attacks on Reduced Rounds of SHACAL-2. In: Canteaut, A., Viswanathan, K. (eds.) INDOCRYPT 2004. LNCS, vol. 3348, pp. 175–189. Springer, Heidelberg (2004)
20. Kim, J., Moon, D., Lee, W., Hong, S., Lee, S., Jung, S.: Amplified Boomerang Attack against Reduced-Round SHACAL. In: Zheng, Y. (ed.) ASIACRYPT 2002. LNCS, vol. 2501, pp. 243–253. Springer, Heidelberg (2002)
21. NESSIE – New European Schemes for Signatures, Integrity and Encryption, `http://www.nessie.eu.org/nessie`
22. NESSIE, Portfolio of recommended cryptographic primitives
23. NESSIE, Performance of Optimized Implementations of the NESSIE Primitives, NES/DOC/TEC/WP6/D21/2
24. Saarinen, M.-J.O.: Cryptanalysis of Block Ciphers Based on SHA-1 and MD5. In: Johansson, T. (ed.) FSE 2003. LNCS, vol. 2887, pp. 36–44. Springer, Heidelberg (2003)
25. National, U.S.: Bureau of Standards, Secure Hash Standard, Federal Information Processing Standards Publications No. 180-2 (2002)
26. Bogeart, E.V.D., Rijmen, V.: Differential Analysis of SHACAL, NESSIE internal report NES/DOC/KUL/WP3/009/a (2001)
27. Wagner, D.: The Boomerang Attack. In: Knudsen, L.R. (ed.) FSE 1999. LNCS, vol. 1636, pp. 156–170. Springer, Heidelberg (1999)
28. Wang, X., Yao, A.C., Yao, F.: Cryptanalysis on SHA-1. In: Cryptographic Hash Workshop, NIST, Gaithersburg (2005)
29. Wang, X., Lai, X., Feng, D., Chen, H., Yu, X.: Cryptanalysis of the Hash Functions MD4 and RIPEMD. In: Cramer, R.J.F. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 1–18. Springer, Heidelberg (2005)
30. Wang, X., Yin, Y.L., Yu, H.: Finding Collisions in the Full SHA-1. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 17–36. Springer, Heidelberg (2005)
31. Wang, X., Yu, H.: How to Break MD5 and Other Hash Functions. In: Cramer, R.J.F. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 19–35. Springer, Heidelberg (2005)
32. Wang, X., Yu, H., Yin, Y.L.: Efficient Collision Search Attacks on SHA-0. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 1–16. Springer, Heidelberg (2005)

A Related-Key Differentials of SHACAL-1

In this appendix we describe the differentials used for our related-key rectangle attacks on SHACAL-1. These differentials are based on the collision producing differentials presented in [30].

Table 2. Related-Key Differential for Rounds 0–33 of SHACAL-1

Round (i)	ΔK	ΔA_i	ΔB_i	ΔC_i	ΔD_i	ΔE_i	Probability
Input 0	e_1	0	0	e_{31}	e_{31}	e_{31}	2^{-1}
$1^\dagger$	e_6	e_1	0	0	e_{31}	e_{31}	2^{-2}
$2^\dagger$	$e_{1,31}$	0	e_1	0	0	e_{31}	2^{-2}
3	e_{31}	0	0	e_{31}	0	0	2^{-1}
4	$e_{1,31}$	0	0	0	e_{31}	0	2^{-2}
5	$e_{6,31}$	e_1	0	0	0	e_{31}	2^{-1}
6	0	0	e_1	0	0	0	2^{-2}
7	$e_{6,31}$	e_1	0	e_{31}	0	0	2^{-2}
8	e_{31}	0	e_1	0	e_{31}	0	2^{-3}
9	e_6	e_1	0	e_{31}	0	e_{31}	2^{-2}
10	e_{31}	0	e_1	0	e_{31}	0	2^{-3}
11	e_6	e_1	0	e_{31}	0	e_{31}	2^{-2}
12	$e_{1,31}$	0	e_1	0	e_{31}	0	2^{-3}
13	0	0	0	e_{31}	0	e_{31}	2^{-1}
14	e_{31}	0	0	0	e_{31}	0	2^{-1}
15	e_{31}	0	0	0	0	e_{31}	1
16	0	0	0	0	0	0	1
17	0	0	0	0	0	0	1
18	0	0	0	0	0	0	1
19	0	0	0	0	0	0	1
20	0	0	0	0	0	0	1
21	0	0	0	0	0	0	1
22	0	0	0	0	0	0	1
23	0	0	0	0	0	0	1
24	0	0	0	0	0	0	1
25	0	0	0	0	0	0	1
26	e_2	0	0	0	0	0	2^{-1}
27	e_7	e_2	0	0	0	0	2^{-1}
28	e_2	0	e_2	0	0	0	2^{-1}
29	$e_{0,3}$	0	0	e_0	0	0	2^{-2}
30	$e_{0,8}$	e_3	0	0	e_0	0	2^{-2}
31	$e_{0,3}$	0	e_3	0	0	e_0	2^{-2}
32	$e_{1,4}$	0	0	e_1	0	0	2^{-2}
33	$e_{1,9}$	e_4	0	0	e_1	0	2^{-2}
Output (34)		0	e_4	0	0	e_1	

† — The probability of this round can be improved by a factor of 2.
Differences are presented before the round, i.e., ΔA_0 is the input difference.

The first related-key differential is for rounds 0–33 and is presented in Table 2. The probability of the differential is 2^{-41}. This probability can be increased by a factor of 4 by fixing the equivalent to two bits in each of the plaintexts of the pair. If we set the most significant bit of A to be zero, the probability of the second round of the differential is increased by a factor of 2. By setting bit 3 of A to differ from bit 3 of B, the probability of the third round of the differential is also increased by a factor of 2.

Table 3. Related-Key Differential for Rounds 34–68 of SHACAL-1

Round (i)	ΔK	ΔA_i	ΔB_i	ΔC_i	ΔD_i	ΔE_i	Probability
Input 34	$e_{1,30}$	0	e_1	e_{31}	0	$e_{30,31}$	2^{-2}
35	e_1	0	0	e_{31}	e_{31}	0	2^{-1}
36	e_6	e_1	0	0	e_{31}	e_{31}	2^{-1}
37	$e_{1,31}$	0	e_1	0	0	e_{31}	2^{-1}
38	e_{31}	0	0	e_{31}	0	0	1
39	$e_{1,31}$	0	0	0	e_{31}	0	2^{-1}
40	$e_{6,31}$	e_1	0	0	0	e_{31}	2^{-1}
41	0	0	e_1	0	0	0	2^{-2}
42	$e_{6,31}$	e_1	0	e_{31}	0	0	2^{-2}
43	e_{31}	0	e_1	0	e_{31}	0	2^{-3}
44	e_6	e_1	0	e_{31}	0	e_{31}	2^{-2}
45	e_{31}	0	e_1	0	e_{31}	0	2^{-3}
46	e_6	e_1	0	e_{31}	0	e_{31}	2^{-2}
47	$e_{1,31}$	0	e_1	0	e_{31}	0	2^{-3}
48	0	0	0	e_{31}	0	e_{31}	2^{-1}
49	e_{31}	0	0	0	e_{31}	0	2^{-1}
50	e_{31}	0	0	0	0	e_{31}	1
51	0	0	0	0	0	0	1
52	0	0	0	0	0	0	1
53	0	0	0	0	0	0	1
54	0	0	0	0	0	0	1
55	0	0	0	0	0	0	1
56	0	0	0	0	0	0	1
57	0	0	0	0	0	0	1
58	0	0	0	0	0	0	1
59	0	0	0	0	0	0	1
60	0	0	0	0	0	0	1
61	e_2	0	0	0	0	0	2^{-1}
62	e_7	e_2	0	0	0	0	2^{-1}
63	e_2	0	e_2	0	0	0	2^{-1}
64	$e_{0,3}$	0	0	e_0	0	0	2^{-2}
65	$e_{0,8}$	e_3	0	0	e_0	0	2^{-2}
66	$e_{0,3}$	0	e_3	0	0	e_0	2^{-2}
67	$e_{1,4}$	0	0	e_1	0	0	2^{-2}
68	$e_{1,9}$	e_4	0	0	e_1	0	2^{-2}
Output 69		0	e_4	0	0	e_1	

Differences are presented before the round, i.e., ΔA_{34} is the input difference.

We use the notation e_i to represent the 32-bit word composed of 31 0's and 1 in the ith place. We use $e_{i,j}$ to denote $e_i \oplus e_j$ and $e_{i,j,k} = e_{i,j} \oplus e_k$, etc.

Due to the nature of the rectangle attack, we can improve the probability by counting over several differentials. We have counted over differentials which have the same first 33 rounds as the differential presented in Table 2. The resulting probability is $\hat{p} = 2^{-38.5}$ (when fixing the respective bits of the plaintext).

The second related-key differential for rounds 34–68 is presented in Table 3. This differential is also based on the collision producing differentials of [30]. The probability of this differential is 2^{-39}.

Again, due to the nature of the rectangle attack, we can improve the probability by counting over several differentials. We count over various similar characteristics, by changing the first round of this differential. The resulting probability is $\hat{q} = 2^{-38.3}$.

Cryptanalysis of Achterbahn-Version 2

Martin Hell and Thomas Johansson

Dept. of Information Technology, Lund University,
P.O. Box 118, 221 00 Lund, Sweden
{martin,thomas}@it.lth.se

Abstract. Achterbahn is one of the stream cipher proposals in the eSTREAM project. After the first version had been successfully cryptanalyzed, the second version, denoted Achterbahn-Version 2, was proposed. This paper demonstrates an attack on this second version. In the attack, a quadratic approximation of the output function is considered. The attack uses less keystream bits than the upper limit given by the designers and the computational complexity is significantly less than exhaustive key search.

Keywords: Achterbahn, cryptanalysis, stream ciphers, key recovery attack.

1 Introduction

The Achterbahn stream cipher is one of many candidates submitted to the eSTREAM [1] project. It is to be considered as a hardware efficient cipher, using a key size of 80 bits. There have been some successful attacks on Achterbahn [6,5]. As a response to these attacks, the cipher was updated to a more secure version, denoted Achterbahn-Version 2 [4]. Recently, eSTREAM moved into the second phase of the evaluation process and based on the design of Achterbahn-Version 2, the cipher qualified as one of the phase 2 ciphers. After receiving a preliminary version of this paper, the designers tweaked the cipher one more time and the version that is to be considered for the second phase of eSTREAM is the third version, denoted Achterbahn-128/80. This third version will not be considered in this paper.

The design of Achterbahn is based on the idea of a nonlinear combiner, but using nonlinear feedback shift registers instead of registers with linear feedback. When Achterbahn was tweaked, the designers focused on improving the cipher such that approximations of the output function was not a threat. In this paper, we show that the tweak was not enough, it is still possible to attack the cipher using approximations of the output function. This is the first attack on Achterbahn-Version 2.

The paper is outlined as follows. Section 2 will discuss some background theory. Section 3 gives a description of the Achterbahn stream cipher. In Section 4 we give the previous results on Achterbahn that are important to our analysis, which is then given in Section 5. Section 6 will conclude the paper.

© Springer-Verlag Berlin Heidelberg 2007

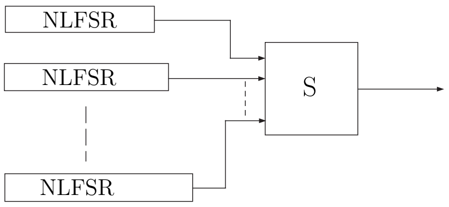


Fig. 1. Overview of the Achterbahn design idea

2 Preliminaries

In this paper we will repeatedly refer to the bias of an approximation. The bias ϵ of an approximation A of a Boolean function P is usually defined in one of two ways.

1. $\Pr(P = A) = 1/2 + \epsilon$. In this case, when n independent bits are xored the bias of the sum is given by $2^{n-1}\epsilon^n$.
2. $\Pr(P = A) = 1/2(1+\epsilon)$. In this case, when n independent bits are xored the bias of the sum is given by ϵ^n. This bias is also commonly referred to as the imbalance.

The bias in the first case will always be half of the bias in the second case. Nevertheless, it is common to approximate the number of keystream bits needed in a distinguisher as

$$\#\text{ samples needed} = \frac{1}{\epsilon^2} \tag{1}$$

regardless which definition of the bias that has been used. The error probability of the distinguisher decreases exponentially with a constant factor multiplied with the number of samples given in (1). Following the notation used in all previous papers on Achterbahn, we will adopt the second case in this paper. Thus, $\epsilon = 2\Pr(P = A) - 1$. Obviously, the sign of ϵ is irrelevant in the theoretical analysis. In the following, when P equals A with probability α, we will write this as $P \overset{\alpha}{=} A$.

3 Description of Achterbahn

The Achterbahn stream cipher was first proposed in [2] and later tweaked in [4]. This section will describe both versions of Achterbahn.

Achterbahn supports a key of size 80 bits. The size of the IV is variable and all multiples of 8 between 0 and 64 bits are supported. The cipher consists of a set of nonlinear feedback shift registers and an output function, see Fig. 1. All registers are primitive, which in this context means that the period of register R_i is $2^{N_i} - 1$, where N_i is the length of register R_i. We denote this period by T_i. Hence,

$$T_i = 2^{N_i} - 1, \quad \forall i.$$

The output function is a Boolean function that takes one input bit from each shift register and outputs a keystream bit. The input bit to the Boolean function from register R_i at time t will be denoted $x_i(t)$ and if the time instance t is fixed the simplified notation x_i will sometimes be used.

Achterbahn comes in two variants, denoted reduced Achterbahn and full Achterbahn. In reduced Achterbahn the input bit to the Boolean function from shift register R_i is simply the output bit of R_i. In full Achterbahn the bit used in the Boolean function is a key dependent linear combination of a few bits in R_i. Achterbahn-Version 1 uses 8 shift registers. Their size ranges from 22 to 31 bits. The keystream bit, denoted z, is produced by the Boolean function

$$R(x_1,\ldots,x_8) = x_1 + x_2 + x_3 + x_4 + x_5x_7 + x_6x_7 + x_6x_8 + x_5x_6x_7 + x_6x_7x_8.$$

Achterbahn-Version 2 uses two extra shift registers, hence, it consists of 10 nonlinear feedback shift registers of size ranging from 19 to 32 bits. Their sizes are $N = 19, 22, 23, 25, 26, 27, 28, 29, 31$ and 32. The Boolean output function in Achterbahn-Version 2 is much larger than the function used in Version 1. It is defined as

$$\begin{aligned} S(x_1,\ldots,x_{10}) &= x_1 + x_2 + x_3 + x_9 + G(x_4,x_5,x_6,x_7,x_{10}) \\ &\quad + (x_8 + x_9)(G(x_4,x_5,x_6,x_7,x_{10}) + H(x_1,x_2,x_3,x_4,x_5,x_6,x_7,x_{10})), \end{aligned}$$

where

$$\begin{aligned} G(x_4,x_5,x_6,x_7,x_{10}) &= x_4(x_5 \vee x_{10}) + x_5(x_6 \vee x_7) + x_6(x_4 \vee x_{10}) \\ &\quad + x_7(x_4 \vee x_6) + x_{10}(x_5 \vee x_7) \end{aligned}$$

and

$$\begin{aligned} H(x_1,x_2,x_3,x_4,x_5,x_6,x_7,x_{10}) &= x_2 + x_5 + x_7 + x_{10} + (x_3 + x_4)\overline{x}_6 \\ &\quad + (x_1 + x_2)(x_3\overline{x}_6 + x_6(x_4 + x_5)). \end{aligned}$$

The function S has resiliency 5 and nonlinearity 448. This means that any biased linear or nonlinear approximation has to consider at least 6 variables.

3.1 Initialization

The initialization of Achterbahn is simple. It is divided into 5 or 6 steps depending on if the reduced or the full variant is used. These steps are the following.

1. All registers R_i are loaded in parallel with the first N_i bits of the key.
2. The registers are updated and the remaining key bits are xored with the input to the registers.
3. The registers are updated and the IV bits are xored with the input to the registers.
4. The least significant bit of each NLFSR is set to 1. This prevents the NLFSRs to be initialized with all zeros.

5. Warm up phase. The registers are clocked such that the total number of clocks for each register in the initialization phase is $112 + |IV|$, where $|IV|$ is the chosen length of the IV.
6. A key and IV dependent vector is prepared, defining which of the positions in the registers that are to be xored to form the input to the Boolean combining function. (This step is only used in the full variant of Achterbahn.)

Because of step 4 above, the entropy of the content of register R_i at the beginning of step 5 is only $N_i - 1$. No secrets (key bits) are used in phase 5 and thus, exhaustively searching register R_i requires 2^{N_i-1} tries. Note that in previous papers on Achterbahn the entropy loss in step 4 has not been taken into account and when we discuss the previous analysis the factor 2^{N_i} will be used as was done in the original papers.

4 Previous Analysis of Achterbahn

There are several papers analyzing the Achterbahn stream cipher. In this section we take a closer look at them and give the results that are relevant to the new attack given in Section 5.

4.1 Analysis of Achterbahn-Version 1

Achterbahn-Version 1 was first cryptanalyzed in [5], taking advantage of weaknesses found in the Boolean output function. The designers answered by giving two alternative combining functions R' and R'' in [3]. In [6], which is an extended and published version of [5], the authors show that the cipher is weak even if the new combining functions are used. An important observation in [5] is the following. Assume that $x_5 = x_6 = 0$ in $R(x_1, \ldots, x_8)$. Then $R(x_1, \ldots, x_8)$ is a purely linear function. The linear complexity of the resulting function $l(t) = x_1(t) \oplus x_2(t) \oplus x_3(t) \oplus x_4(t)$ is then bounded by the sum of the linear complexities of the registers R_1, R_2, R_3 and R_4, which is approximately 2^{26}. Hence assuming that $x_5 = x_6 = 0$, there are parity checks involving at most 2^{26} consecutive bits. This parity check equation could be found by noting that $l(t) \oplus l(t + T_i)$ does not depend on the variable x_i. Doing this for $i = 1, 2, 3$ and 4, a parity check equation involving 16 terms within a time interval of $2^{26.75}$ keystream bits can be found. By knowing for which initial states of R_5 and R_6 these 16 terms will be zero, i.e., when the parity check was valid with probability 1, the key could be recovered.

The method of finding a parity check was nicely refined and generalized in [4]. They note that the sequence generated by R_i has characteristic polynomial $x^{T_i} - 1$. Furthermore,

$$g(x) = (x^{T_1} - 1)(x^{T_2} - 1)(x^{T_3} - 1)(x^{T_4} - 1)$$

is a characteristic polynomial of $l(t)$. Even if all variables do not appear linearly in the ANF of a Boolean function, a sparse parity check can easily be found. For instance, the sequence produced by the function $F(t) = x_1(t)x_2(t) \oplus x_1(t)x_2(t)x_3(t)$ has characteristic polynomial

$$g(x) = (x^{T_1 T_2} - 1)(x^{T_1 T_2 T_3} - 1),$$

giving a parity check equation involving only 4 terms.

In [6], the authors also demonstrated that it is possible to break Achterbahn by considering biased linear approximations of the output function. The approximation

$$z(t) \overset{\alpha}{=} x_1(t) \oplus x_2(t) \oplus x_3(t) \oplus x_4(t) \oplus x_6(t)$$

holds with probability $\alpha = 0.75$, i.e., it has a bias $\epsilon = 0.5$. Since there are 32 terms in the corresponding parity check equation, the total bias is 2^{-32} and a distinguishing attack using 2^{64} bits exists. Furthermore, they note that by guessing the state of register R_1, the parity check will only involve 16 terms and the distinguisher will only need 2^{32} bits. Additionally, the computational complexity will increase by a factor of 2^{23}. Now the attack is a key recovery attack with computational complexity 2^{55} using 2^{32} bits of keystream. This is the best known attack on reduced Achterbahn. The same attack is possible on the full version, but the computational complexity is then 2^{61} instead.

4.2 Analysis of Achterbahn-Version 2

In [4], the designers of Achterbahn demonstrate that the attacks mentioned above will not work when applied to Version 2. This is mostly due to the fact that the combining function $S(x_1, \ldots, x_{10})$ is 5-resilient, thus any biased linear approximation has at least 6 terms and the corresponding parity check will have 64 terms. By guessing the state of the first two registers, the number of terms in the parity check will be 16, but even then the computational complexity and the keystream needed will be far above exhaustive key search.

Further, the designers also considered quadratic and cubic approximations of $S(x_1, \ldots, x_{10})$. In this section we give a description of the cubic case since the result of this analysis gives a very important prerequisite for Achterbahn-Version 2. Our attack will use a quadratic approximation. The cubic approximation that is considered to be most threatening is given by

$$C(x_1, \ldots, x_{10}) = x_4 + x_6 x_9 + x_1 x_2 x_3.$$

This approximation will agree with $S(x_1, \ldots, x_{10})$ with probability

$$\frac{63}{128} = \frac{1}{2}\left(1 - \frac{1}{64}\right),$$

implying that $\epsilon = 2^{-6}$. We can guess the content of register R_4 with $N_4 = 25$. The characteristic polynomial of the sequence generated by the two nonlinear terms is

$$g(x) = (x^{T_6 T_9} - 1)(x^{T_1 T_2 T_3} - 1).$$

This will give a parity check equation with 4 terms and bias $\epsilon^4 = (2^{-6})^4 = 2^{-24}$ assuming that the variables are independent. The distance between the first and the last bit in the parity check is $T_1T_2T_3 + T_6T_9 \approx 2^{64}$ bits. The time complexity of this attack is $2^{48}2^{N_4} = 2^{73}$. This is less than exhaustive key search and consequently *the designers restrict the frame length of Achterbahn-Version 2 to* 2^{63} *bits.*

Note that the previously described attack is *impossible* when the keystream length is limited to 2^{63} since then we cannot create any biased samples at all. In most distinguishing attacks on stream ciphers, you can usually create biased samples even if the keystream length is limited, it is just the case that you cannot collect enough samples to detect the bias for sure.

5 Cryptanalysis of Achterbahn-Version 2

Since there is an attack requiring approximately 2^{64} keystream bits, and the frame length is restricted to 2^{63} bits, a new attack has to require less than 2^{63} keystream bits in order to be regarded as successful. A danger of restricting the amount of keystream to some number due to the existence of an attack is that someone might find an improvement of the attack. This would render the cipher insecure. In this section we demonstrate exactly that. A straightforward approach of our attack is given first and in Section 5.4 an improved variant is given, reducing the computational complexity significantly.

5.1 Attack on the Reduced Variant

The complexities given in this subsection will be based on the reduced variant of the cipher, i.e., the input to the Boolean combining function will be the rightmost bit in each NLFSR.

The attack will consider the quadratic approximation

$$Q(x_1, \ldots, x_{10}) = x_1 + x_2 + x_3x_8 + x_4x_6.$$

This approximation will agree with S with probability

$$\frac{33}{64} = \frac{1}{2}\left(1 + \frac{1}{32}\right),$$

implying that $\epsilon = 2^{-5}$. Denote the sequence produced by Q by $z'(t)$. Using this approximation, we can use the characteristic polynomial

$$g(x) = (x^{T_3T_8} - 1)(x^{T_4T_6} - 1).$$

which gives a parity check equation involving 4 terms. Looking at the sequence generated by Achterbahn-Version 2, we know that if we consider the sequence

$$d(t) = z(t) \oplus z(t + T_3T_8) \oplus z(t + T_4T_6) \oplus z(t + T_3T_8 + T_4T_6) \qquad (2)$$

then $d(t)$ will not depend on the quadratic terms in $Q(x_1, \ldots, x_{10})$. Under the assumption that the 4 keystream bits in (2) are independent, the bias of (2) is $\epsilon^4 = 2^{-20}$. If these keystream bits are not independent the bias will be larger, so we are considering a worst case scenario. The dependency of the keystream bits will be further examined in a separate paper. Hence, with probability $\alpha = 1/2(1 + 2^{-20})$, the sequence $d(t)$ will equal

$$\begin{aligned} d(t) &\overset{\alpha}{=} z'(t) \oplus z'(t + T_3T_8) \oplus z'(t + T_4T_6) \oplus z'(t + T_3T_8 + T_4T_6) \\ &= x_1(t) \oplus x_2(t) \oplus x_1(t + T_3T_8) \oplus x_2(t + T_3T_8) \oplus x_1(t + T_4T_6) \\ &\quad \oplus x_2(t + T_4T_6) \oplus x_1(t + T_3T_8 + T_4T_6) \oplus x_2(t + T_3T_8 + T_4T_6). \end{aligned}$$

At this point we can guess the initial state of the registers R_1 and R_2 as suggested in [4], where they used another approximation. The length of these two registers is $N_1 = 19$ and $N_2 = 22$ respectively. The amount of keystream needed to distinguish the output sequence from random is 2^{40} so the computational complexity would be $2^{18+21+40} = 2^{79}$, which is the same as the expected complexity in an exhaustive search. The distance between the bits in the sum is $T_3T_8 + T_4T_6 \approx 2^{53}$ so this would be the amount of keystream needed.

Instead of taking this approach we note that the length of register R_1 is $N_1 = 19$, hence,

$$x_1(t) = x_1(t + T_1) = x_1(t + 2^{19} - 1).$$

Thus, for all keystream bits, distance $T_1 = 2^{19} - 1$ bits apart, x_1 will always contribute with the same value to the output function. Consequently, instead of taking the sequence $d(t)$ for $t = 0 \ldots 2^{40} - 1$ we can instead take the sequence $d'(t) = d(t(2^{19} - 1))$ for $t = 0 \ldots 2^{40} - 1$, i.e., jump forward T_1 steps for each sample. Hence,

$$\begin{aligned} d'(t) &= z(tT_1) \oplus z(tT_1 + T_3T_8) \oplus z(tT_1 + T_4T_6) \oplus z(tT_1 + T_3T_8 + T_4T_6) \\ &\overset{\alpha}{=} x_2(tT_1) \oplus x_2(tT_1 + T_3T_8) \oplus x_2(tT_1 + T_4T_6) \\ &\quad \oplus x_2(tT_1 + T_3T_8 + T_4T_6) \oplus \gamma(t), \end{aligned}$$

where

$$\gamma(t) = x_1(tT_1) \oplus x_1(tT_1 + T_3T_8) \oplus x_1(tT_1 + T_4T_6) \oplus x_1(tT_1 + T_3T_8 + T_4T_6)$$

is a constant. If the value of $\gamma(t) = 0$, then the probability $\alpha = 1/2(1 + 2^{-20})$. If the value $\gamma(t) = 1$ then $\alpha = 1/2(1 - 2^{-20})$. In any case, the number of samples needed to detect the bias is 2^{40}. The total amount of keystream required in this approach will increase with a factor of 2^{T_1}, i.e.,

$$\#\ \text{keystream bits needed} = 2^{53} + 2^{19}2^{40} = 2^{59.02}.$$

This value is less than the maximum length of a frame. The computational complexity will be $2^{40}2^{21} = 2^{61}$, since now we only need to guess R_2 with $N_1 = 22$, requiring 2^{21} guesses. This will give us the initial state of register R_2 after step 4 in the initialization process.

5.2 Recovering the Key

When one state is known, finding the actual key used can be done using a meet-in-the-middle attack and a time/memory tradeoff approach. First, R_2 is clocked backwards until we reach the state that ended the introduction of the key. We denote this state Δ. Then the key is divided into two parts, k_1 and k_2 bits each and $k_2 = 80 - k_1$. We guess the first k_1 bits of the key and clock the register until after the introduction of this part. All possible 2^{k_1} states are saved in a table. Then the last k_2 bits of the key are guessed, and the state Δ is clocked backwards k_2 times reversing the introduction of the key. Any intersection of the two states reached, gives a possible key candidate. Since R_2 has size $N_2 = 22$ we expect the number of intersections to be $2^{80}2^{-22} = 2^{58}$, i.e., less than the complexity of finding the state of R_2. The step of finding the intersections will require memory 2^{k_1} and time $2^{k_1} + 2^{k_2}$. Appropriate values can be e.g., $k_1 = 30$ and $k_2 = 50$. The total computational complexity of the attack would then be $2^{61} + 2^{58} = 2^{61.17}$.

5.3 Attack on the Full Variant

The full Achterbahn-Version 2 uses a key dependent linear combination of the shift register bits as input to the Boolean combining function. To the best of our knowledge, there is no specification of Version 2 that explicitly gives the amount of bits in each register that is used in the linear combination. However, in the analysis given in [4, Sect. 3.3], the designers imply that for the registers R_1, R_2 and R_3, 3 register bits are used in each. In our attack we are only interested in the amount of bits used from R_1 and R_2 so this information is sufficient. The consequence is that, when attacking the full variant, an extra factor of 2^3 has to be multiplied when finding the state register R_2.

5.4 Improving the Computational Complexity

In the previous subsection, a simple approach for the attack was given resulting in computational complexity $2^{61.17}$ and $2^{59.02}$ keystream bits for the reduced variant. The computational complexity of the attack can be significantly reduced using the fact that the period of the registers are very short. In this subsection we go through each step in the attack and give the computational complexity in each step. It is assumed that the cryptanalyst observes a sequence of $2^{59.02}$ keystream bits.

- **Produce d'(t).** From the observed keystream sequence, the sequence $d'(t)$ of length 2^{40} is computed. This will have computational complexity 2^{42} since each bit in $d'(t)$ is the sum of 4 bits in the keystream. The amount of memory required to save this sequence is 2^{40} bits, i.e., 2^{37} bytes.
- **Build a table from d'(t).** The straightforward approach when $d'(t)$ is available is to compare the bits in $d'(t)$ with the bits produced by

$$x_2(tT_1) \oplus x_2(tT_1 + T_3T_8) \oplus x_2(tT_1 + T_4T_6) \oplus x_2(tT_1 + T_3T_8 + T_4T_6),$$

Position in d'(t)	# Zeros	# Ones
$0 + iT_2$		
$1 + iT_2$		
$2 + iT_2$		
$\vdots$		
$T_2 - 1 + iT_2$		

Fig. 2. Store the number of ones and zeros in a table

$0 \leq t < 2^{40}$, for all possible initial states of R_2. Indeed, this would require a computational complexity of 2^{61} as given in Section 5.1. To speed up the exhaustive search of register R_2 we note that

$$x_2(t) = x_2(t \bmod T_2).$$

This means that all $d'(t + iT_2)$, $\forall i$, will be compared with the same value. In order to take advantage of this, we suggest to build a table with the bits in $d'(t)$. We go through $d'(t)$ and count the number of zeros and ones in $d'(0+iT_2)$, $d'(1+iT_2)$, $d'(2+iT_2)$, etc. These numbers are stored in a table, see Fig 2. This step will have computational complexity 2^{40} and requires about 2^{22} words of memory.

- **Recover $\mathbf{R_2}$**. When the state of register R_2 is to be recovered the table in Fig 2 is used. For each possible initial state of R_2 the sum of the four bits

$$x_2(tT_1) \oplus x_2(tT_1 + T_3T_8) \oplus x_2(tT_1 + T_4T_6) \oplus x_2(tT_1 + T_3T_8 + T_4T_6), \quad (3)$$

  $0 \leq t < T_2$, is found. Note that all positions are taken modulo T_2. The number of occurrences in the precomputed table is then added together where the column used is the value of the sum (3). The bias will be detected for the correct initial state of R_2. Because of the precomputed table, this step will now only have computational complexity $T_2 2^{21} = 2^{43}$ instead of 2^{61}. For full Achterbahn-Version 2, this complexity will be increased to 2^{46}.
- **Recover the key.** To recover the key, the meet-in-the-middle approach given in section 5.1 can be used. In that case 2^{58} keys will be candidates as correct key which is much higher than the computational complexity to find the state of R_2. To reduce this number, we first find the state of R_1. This is easy now since R_2 is known. A similar table can be produced from the sequence $d(t)$ and the initial state of R_1 is found with complexity $T_1 2^{18} = 2^{37}$. When both R_1 and R_2 are known the expected number of key candidates decreases to $2^{80-22-19} = 2^{39}$. All these key candidates can be tested without this step being a computational bottleneck.

It is interesting to note that once we have received $2^{59.02}$ keystream bits, the maximum computational step is only 2^{43} and 2^{46} for the reduced and full variants respectively. This is due to the fact that we only use a fraction of the received keystream and that we can take advantage of the fact that the registers have short period. It is debatable if we can claim that the computational complexity of the attack much lower than the required amount of keystream since producing and receiving the keystream will require at least $2^{59.02}$ clockings of the cipher. On the other hand, if we are given a randomly accessible memory with $2^{59.02}$ keystream bits, then the key is found with much fewer computational steps since not all bits on the memory will be accessed. This could be a possible scenario in the case of future DVD formats with extremely high resolution, though the access time would probably be a bottleneck in that case. Anyway, we will be conservative in our claims and consider the computational complexity to be the same as the amount of keystream needed, i.e., $2^{59.02}$. Consequently, the attack on full and reduced Achterbahn-Version 2 will have the same complexity.

5.5 On the Problem of Finding the Initial State of R_2

When we try to recover the initial state of R_2 we assume that it is enough to consider $1/\epsilon^2$ bits in order to detect the bias and thus identifying the correct initial state. In total 2^{21} different candidate states are tested and while it is true that the bias will be detected for the correct initial state it is very likely that several other states will report a detected bias. This will happen because we can assume that the sum of the positions in the precomputed table will be distributed according to a normal distribution with expected value 2^{39}. In our case, this is not really a problem. The correct initial state can still be found using only $2^{59.02}$ bits and 2^{40} samples to detect the bias. For all states that report a bias, we can do the same thing again, shifting the sequence $d'(t)$ one step. This will give 2^{40} new samples and we can check which of the remaining states will report a detected bias. The total amount of keystream needed in our attack will be increased by only 1 bit and the extra amount of computations will be about 2^{40}, the complexity of building a new table. Since our claimed complexity is $2^{59.02}$ we can do this procedure many times if necessary without exceeding the claimed computational complexity.

6 Conclusion

Achterbahn-Version 2 was designed to resist approximations of the output functions, linear approximations as well as quadratic and cubic approximations. Due to a cubic approximation, the amount of keystream that is allowed to be generated is limited to 2^{63}. In this paper we have shown that it is still possible to find an attack using a quadratic approximation. The amount of keystream needed in the attack is below the given limit. Instead of guessing both R_1 and R_2, as was done in a previous analysis, we guess only one of the registers. The attack on Achterbahn-Version 2 has computational complexity slightly more than 2^{59} and needs slightly more than 2^{59} keystream bits. After receiving the keystream bits the computational step is very fast due to the fact that we do not use all keystream bits and that the periods of the registers are very short. The complexities will be the same for both the full and the reduced variants of the cipher.

Acknowledgement

The work described in this paper has been supported in part by the European Commission through the IST Programme under Contract IST-2002-507932 ECRYPT. The information in this document reflects only the author's views, is provided as is and no guarantee or warranty is given that the information is fit for any particular purpose. The user thereof uses the information at its sole risk and liability.

References

1. ECRYPT. eSTREAM: ECRYPT Stream Cipher Project, IST-2002-507932. Available at `http://www.ecrypt.eu.org/stream/`
2. Gammel, B.M., Göttfert, R., Kniffler, O.: The Achterbahn stream cipher. eSTREAM, ECRYPT Stream Cipher Project, Report 2005/002 (2005), `http://www.ecrypt.eu.org/stream`
3. Gammel, B.M., Göttfert, R., Kniffler, O.: Improved Boolean combining functions for Achterbahn. eSTREAM, ECRYPT Stream Cipher Project, Report 2005/072 (2005), `http://www.ecrypt.eu.org/stream`
4. Gammel, B.M., Göttfert, R., Kniffler, O.: Status of Achterbahn and tweaks. The State of the Art of Stream Ciphers. In: Workshop Record, SASC 2006, Leuven, Belgium (February 2006)
5. Johansson, T., Meier, W., Müller, F.: Cryptanalysis of Achterbahn. eSTREAM, ECRYPT Stream Cipher Project, Report 2005/064 (2005), `http://www.ecrypt.eu.org/stream`
6. Johansson, T., Meier, W., Müller, F.: Cryptanalysis of Achterbahn. In: Robshaw, M. (ed.) FSE 2006. LNCS, vol. 4047, Springer, Heidelberg (2006)

Cryptanalysis of the Stream Cipher ABC v2*

Hongjun Wu and Bart Preneel

Katholieke Universiteit Leuven, ESAT/SCD-COSIC
Kasteelpark Arenberg 10, B-3001 Leuven-Heverlee, Belgium
{wu.hongjun,bart.preneel}@esat.kuleuven.be

Abstract. ABC v2 is a software-efficient stream cipher with 128-bit key. In this paper, we apply a fast correlation attack to break ABC v2 with weak keys. There are about 2^{96} weak keys in ABC v2. The complexity to identify a weak key and to recover the internal state of a weak key is low: identifying one weak key from about 2^{32} random keys requires 6460 keystream bytes and $2^{13.5}$ operations for each random key. Recovering the internal state of a weak key requires about $2^{19.5}$ keystream bytes and $2^{32.8}$ operations. A similar attack can be applied to break ABC v1 with much lower complexity than the previous attack on ABC v1.

Keywords: Fast correlation attack, key-dependent S-box, stream cipher, ABC v2.

1 Introduction

ABC [1] is a stream cipher submitted to the ECRYPT eStream project. It is one of the fastest submissions with encryption speed about 3.5 cycles/byte on the Intel Pentium 4 microprocessor.

ABC v1 was broken by Berbain and Gilbert [3] (later by Khazaei [8]). Their divide-and-conquer attack on ABC exploits the short length (63 bits) of the LFSR in the component A and the non-randomness in the component C: all the possible initial values of the LFSR get tested, and the correct value results in the biased binary stream that matches the non-random output from the component C. The component C is a key-dependent 32-bit-to-32-bit S-box. Vaudenay [12], Murphy and Robshaw [11] have stated that the key-dependent S-boxes may be weak. Berbain and Gilbert's attack on ABC v1 deals with the weak keys that are related to the non-bijective S-box. This type of weak key exists with probability close to 1. Recovering the internal state of a weak key requires about 2^{95} operations and 2^{34} keystream bytes.

In order to resist these attacks, the ABC designers introduced ABC v2 with the improved components A and B. In ABC v2 [2], the length of the LFSR is 127 bits instead of the 63 bits in ABC v1. The increased LFSR length makes it impossible to test all the states of the LFSR, thus the attack on ABC v1

* This work was supported in part by the Concerted Research Action (GOA) Ambiorics 2005/11 of the Flemish Government and in part by the European Commission through the IST Programme under Contract IST-2002-507932 ECRYPT.

© Springer-Verlag Berlin Heidelberg 2007

can no longer be applied to ABC v2. However ABC v2 is still insecure due to the low weight of the LFSR and the non-randomness in the component C (the component C in ABC v1 is the same as in ABC v2).

In this paper, we find a new type of weak key that exists with probability 2^{-32}. This new type of weak key results in a heavily biased output of the component C. Due to the low weight of the LFSR and the strong correlation resulting from the component C, a fast correlation attack can be applied to recover the LFSR. After recovering the LFSR, the internal state of the cipher can be recovered easily. The identification of a weak key from 2^{32} random keys requires 6460 keystream bytes from each key, and $2^{13.5}$ instructions for each keystream. Recovering the internal state of a weak key requires about $2^{27.5}$ keystream bytes and $2^{35.7}$ instructions. Both the ABC v1 and ABC v2 are vulnerable to this attack.

This paper is organized as follows. In Sect. 2, we illustrate the ABC v2. In Sect. 3, we define the weak keys and show how to identify them. Section 4 recovers the internal state of a weak key. Section 5 concludes this paper.

2 The Stream Cipher ABC v2

The stream cipher ABC v2 consists of three components – A, B and C, as shown in Fig. 1. The component A is a regularly clocked LFSR, the component B is a finite state machine (FSM), and the component C is a key-dependent S-box. ABC v1 has the same structure as ABC v2 except that the LFSR in ABC v1 is 63-bit, and the FSM in ABC v1 has less elements than that in ABC v2. The component C in ABC v1 is the same as that in ABC v2.

The component A is based on a linear feedback shift register with primitive polynomial $g(x) = x^{127} + x^{63} + 1$. Denote the register in component A as $(\overline{z}_3, \overline{z}_2, \overline{z}_1, \overline{z}_0)$, where each $\overline{z}_i$ is a 32-bit number. Note that this 128-bit register itself is not a linear feedback shift register. Its initial value depends on the key and IV. At each step of ABC v2, 32 bits of this 128-bit register get updated:

$$\begin{aligned}&\zeta = (\overline{z}_2 \oplus (\overline{z}_1 \ll 31) \oplus (\overline{z}_0 \gg 1)) \bmod 2^{32}\\&\overline{z}_0 = \overline{z}_1\ ,\ \overline{z}_1 = \overline{z}_2\ ,\ \overline{z}_2 = \overline{z}_3\ ,\ \overline{z}_3 = \zeta\ ,\end{aligned}$$

where $\ll$ and $\gg$ indicates left shift and right shift, respectively.

The component B is specified as $B(x) = ((x \oplus d_0) + d_1) \oplus d_2 \bmod 2^{32}$, where x is the 32-bit input, d_0, d_1 and d_2 are key and IV dependent 32-bit numbers, $d_0 \equiv 0 \bmod 4$, $d_1 \equiv 1 \bmod 4$, $d_2 \equiv 0 \bmod 4$. The x is updated as $x = B(x) + \overline{z}_3$.

The component C is specified as $C(x) = S(x) \ggg 16$, where $\ggg$ indicates rotation, x is the 32-bit input, $S(x) = e + \sum_{i=0}^{31}(e_i \times x[i])$, where $x[i]$ denotes the ith least significant bit of x, and e and e_i are key dependent 32-bit random numbers, except that $e_{31} \equiv 2^{16} \bmod 2^{17}$. Note that e and e_i are not related to the initialization vector.

Each 32-bit keystream word is given as $y = C(x) + \overline{z}_0$.

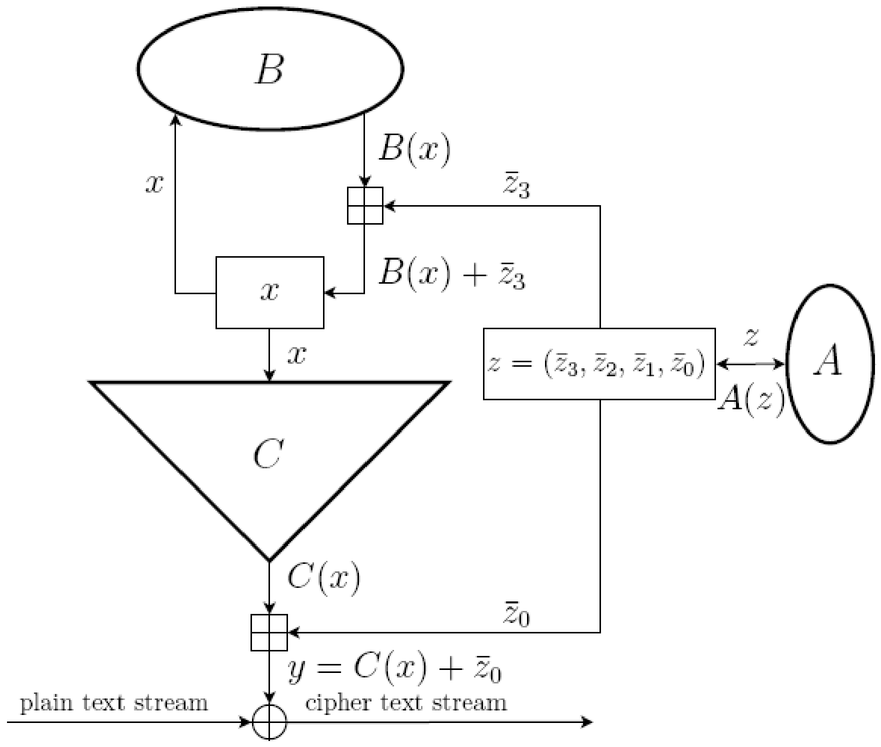


Fig. 1. Keystream generation of ABC v2 [2]

The details of the initialization of ABC v2 are not described here. We are only interested in the generation of the key-dependent S-box in the component C. The above specifications of the component C are sufficient for the illustration of the attacks presented in this paper.

3 The Weak Keys of ABC v2

In Sect. 3.1, we introduce some observation related to the bias of carry bits. Section 3.2 defines the ABC v2 weak keys and gives an attack to identify them.

3.1 The Large Bias of Carry Bits

Carry bits are always biased even if the two addends are random. The bias of the carry bit at the n-th least significant bit position is $\frac{1}{2} + 2^{-n-1}$ ($n \geq 1$). However, this bias is very small for large n. In the following, we look for the large bias of carry bits when the addends are not random.

Table 1. The probability of $c_1 \oplus c_2 \oplus c_3 = 0$ (denote the probability as $\frac{1}{2} + \epsilon$)

n	ϵ	n	ϵ
1	0.125	5	0.071441650390625
2	0.078125	6	0.071430206298828125
3	0.072265625	7	0.071428775787353515625
4	0.071533203125	8	0.071428596973419189453125

Lemma 1. Denote u and v as two random and independent n-bit integers. Let $c_n = (u + v) \gg n$, where c_n denotes the carry bit at the nth least significant bit position. Denote the most significant bit of u as u_{n-1}. Then $\Pr(c_n \oplus u_{n-1} = 0) = \frac{3}{4}$.

Proof. $c_n = (u_{n-1} \cdot v_{n-1}) \oplus ((u_{n-1} \oplus v_{n-1}) \cdot c_{n-1})$. If $c_{n-1} = 0$, then $c_n \oplus u_{n-1} = u_{n-1} \cdot \overline{v}_{n-1}$, where $\overline{v}_{n-1}$ denotes the inverse of v_{n-1}. If $c_{n-1} = 1$, then $c_n \oplus u_{n-1} = \overline{u}_{n-1} \cdot v_{n-1}$. Thus $\Pr(c_n \oplus u_{n-1} = 0) = \frac{3}{4}$.

Lemma 1 implies the following bias.

Theorem 1. Denote a_i, b_i $(1 \leq i \leq 3)$ as n-bit integers. Denote c_i $(1 \leq i \leq 3)$ as binary values satisfying $c_i = (a_i + b_i) \gg n$. Let a_1, a_2, b_1, b_2 and b_3 be random and independent, but $a_3 = a_1 \oplus a_2$. Then $c_1 \oplus c_2 \oplus c_3$ is biased. For $n = 16$, $\Pr(c_1 \oplus c_2 \oplus c_3 = 0) \approx 0.5714$.

If we apply Lemma 1 directly, we obtain that $\Pr(c_1 \oplus c_2 \oplus c_3 = 0) = \frac{1}{2} + \frac{1}{16} = 0.5625$. (The u_{n-1}'s in Lemma 1 are eliminated since they are linearly related in Theorem 1.) The small difference between these two biases (0.5714 and 0.5625) is due to the fact that a_3 is not an independent random number.

We illustrate the validity of Theorem 1 with numerical analysis. For small n, we try all the values of a_1, a_2, b_1, b_2 and b_3 and obtain the following table. From Table 1, we see that the bias ϵ converges to 0.0714 as the value of n increases. For $n = 16$, we performed 2^{32} tests, and the bias is about 0.071424. For $n = 32$, the bias is about 0.071434 with 2^{32} tests. The experimental results show that Theorem 1 is valid. Recently, the complete proof of Theorem 1 is given in [16]. It was shown that $\Pr(c_1 \oplus c_2 \oplus c_3 = 0) = \frac{4}{7} + \frac{3}{7} \times \frac{1}{8^n}$, which confirms the correctness of Theorem 1.

Remarks. In Theorem 1, if a_1, a_2, a_3, b_1, b_2 and b_3 are all random and independent, then $\Pr(c_1 \oplus c_2 \oplus c_3 = 0) = \frac{1}{2} + 2^{-3n-1}$, which is very small for $n = 16$. This small bias cannot be exploited to break ABC v2.

3.2 Identifying the Weak Keys

We start the attack with analyzing the linear feedback shift register used in ABC v2. The register $(\overline{z}_3, \overline{z}_2, \overline{z}_1, \overline{z}_0)$ is updated according to the primitive polynomial $g(x) = x^{127} + x^{63} + 1$. Note that each time the 127-bit LFSR advances

32 steps. To find a linear relation of the 32-bit words, we take the 2^5th power of $g(x)$, and obtain

$$g^{2^5}(x) = x^{127\times 32} + x^{63\times 32} + 1\,. \tag{1}$$

Denote the z_0 at the i-th step as z_0^i, and denote the jth significant bit of z_0^i as $z_{0,j}^i$. Since each time 32 bits get updated, the distance between $\overline{z}_{0,j}^i$ and $\overline{z}_{0,j}^{i+k}$ is $|32\cdot(k-i)|$. According to (1), we obtain the following linear recurrence

$$\overline{z}_0^i \oplus \overline{z}_0^{i+63} \oplus \overline{z}_0^{i+127} = 0\,. \tag{2}$$

The weak keys of ABC v2 are related to the $S(x)$ in the component C. $S(x)$ is defined as $S(x) = e + \sum_{i=0}^{31}(e_i \times x[i])$, where e and e_i are key dependent 32-bit random numbers, except that $e_{31} \equiv 2^{16} \bmod 2^{17}$. **If the least significant bits of e and e_i $(0 \le i < 32)$ are all 0, then the least significant bit of $S(x)$ is always 0, and we consider the key as weak key**. Note that the least significant bit of e_{31} is always 0. Thus a randomly chosen key is weak with probability 2^{-32}.

In the following, we describe how to identify the weak keys. Denote the 32-bit keystream word at the ith step as y_i, the jth significant bit of y_i as $y_{i,j}$. And denote x_i as the input to function S at the i-th step. Then $y_i = (S(x_i) \ggg 16) + \overline{z}_0^i$. Let $c_{i,j}$ denote the carry bit at the j-th least significant bit position of $(S(x_i) \ggg 16) + \overline{z}_0^i$, i.e., $c_{i,j} = (((S(x_i) \ggg 16) \bmod 2^j) + (\overline{z}_0^i \bmod 2^j)) \gg j$. Assume that $((S(x_i) \ggg 16) \bmod 2^{16}$ is random. According to Theorem 1 and (2), we obtain

$$\Pr(c_{i,16} \oplus c_{i+63,16} \oplus c_{i+127,16} = 0) = \frac{1}{2} + 0.0714\,. \tag{3}$$

Due to the rotation of $S(x_i)$, we know that

$$y_{i,16} = S(x_i)_0 \oplus z_{0,16}^i \oplus c_{i,16}\,, \tag{4}$$

where $S(x_i)_0$ denotes the least significant bit of $S(x_i)$. Note that $S(x_i)_0$ is always 0 for a weak key. From (2) and (4), we obtain

$$y_{i,16} \oplus y_{i+63,16} \oplus y_{i+127,16} = c_{i,16} \oplus c_{i+63,16} \oplus c_{i+127,16}\,. \tag{5}$$

From (3) and (5), $y_{i,16}$ is biased as

$$\Pr(y_{i,16} \oplus y_{i+63,16} \oplus y_{i+127,16} = 0) = \frac{1}{2} + 0.0714\,. \tag{6}$$

We use (6) to identify the weak keys. Approximate the binomial distribution with the normal distribution. Denote the total number of samples as N, the mean as μ, and the standard deviation as σ. For the binomial distribution, $p = \frac{1}{2}$, $\mu = Np$ and $\sigma = \sqrt{Np(1-p)}$. For (6), $p' = \frac{1}{2} + 0.0714$, $\mu' = Np'$ and $\sigma' = \sqrt{Np'(1-p')}$. For the normal distribution, the cumulative distribution function gives value $1 - 2^{-39.5}$ at 7σ, and value 0.023 at -2σ. If the following relation holds

$$u' - u \ge 7\sigma + 2\sigma'\,, \tag{7}$$

then on average, each strong key is wrongly identified as weak key (false positive) with probability $2^{-39.5}$, and each weak key is not identified as weak key (false negative) with probability 0.023. It means that the weak keys can be successfully identified since one weak key exists among 2^{32} keys. Solving (7), the amount of samples required is $N = 3954$. For each sample, we only need to perform two XORs and one addition. With $3594+127 = 4081$ outputs from each key, we can successfully identify the weak keys of ABC v2.

The number of outputs can be reduced if we consider the 2^ith power of $g(x)$ for $i = 5, 6, 7, 8$. With 1615 outputs, we can obtain 3956 samples. Thus the keystream required in the identification of the weak keys can be reduced to 1615 outputs.

The identification of a weak key implies directly a distinguishing attack on ABC v2. If there are 2^{32} keystreams generated from 2^{32} random keys, and each key produces 1615 outputs, then the keystream can be distinguished from random with high probability. In order to find one weak key, the total amount of keystream required are $2^{32} \times 1615 \times 4 = 2^{44.7}$ bytes, and the amount of computations required are $2^{32} \times 3956 \times 2 \approx 2^{45}$ XORs and 2^{44} additions.

Experiment 1. We use the original ABC v2 source code provided by the ABC v2 designers in the experiment. After testing 2^{34} random keys, we obtain five weak keys, and one of them is (fe 39 b5 c7 e6 69 5b 44 00 00 00 00 00 00 00 00). From this weak key we generate 2^{30} outputs, and the bias defined in (6) is 0.5714573. The experimental results confirm that the probability that a randomly chosen key is weak is about 2^{-32}, and the bias of a weak key keystream is large.

4 Recovering the Internal State

Once a weak key is identified, we proceed to recover the internal state resulting from the weak key. In Sect. 4.1, we apply the fast correlation attack to recover the LFSR. The components B and C are recovered in Sect. 4.2. The complexity of the attack is given in Sect. 4.3. Section 4.4 applies the attack to ABC v1.

4.1 Recovering the Initial Value of the LFSR

The initial value of the LFSR is recovered by exploiting the strong correlation between the LFSR and the keystream. From Lemma 1, we get

$$\Pr(\overline{z}^{i}_{0,15} \oplus c_{i,16} = 0) = \frac{3}{4}\,. \tag{8}$$

From (8) and (4), we obtain the following correlation:

$$\Pr(\overline{z}^{i}_{0,16} \oplus \overline{z}^{i}_{0,15} \oplus y_{i,16} = 0) = \frac{3}{4}\,. \tag{9}$$

The strong correlation in (9) indicates that the cipher is very weak.

The fast correlation attack Algorithm A of Meier and Staffelbach [9] can be applied to recover the LFSR. There are some advanced fast correlation attacks [10,6,7], but the original attack given by Meier and Staffelbach is sufficient here since we are dealing with a strong correlation and a two-tap LFSR.

We now apply the fast correlation attack Algorithm A [9] to recover the LFSR. Let $p = \frac{3}{4}$, $u_i = \overline{z}^i_{0,16} \oplus \overline{z}^i_{0,15}$, and $w_i = y_{i,16}$. By squaring the polynomial (1) iteratively, we obtain a number of linear relations for every u_i:

$$u_i \oplus u_{i+63\cdot 2^j} \oplus u_{i+127\cdot 2^j} = 0 \quad (j \geq 0). \tag{10}$$

From (9) and (10), we obtain

$$s = \Pr(w_i \oplus w_{i+63\cdot 2^j} \oplus w_{i+127\cdot 2^j} = 0 \mid u_i = w_i) = p^2 + (1-p)^2 , \tag{11}$$

where each value of j indicates one relation for w_i (also for $w_{i+63\cdot 2^j}$ and $w_{i+127\cdot 2^j}$). On average there are m relations for w_i as

$$m = m(N, k, t) = (t+1) \cdot \log_2(\frac{N}{2k}) , \tag{12}$$

where N is the number of outputs, $k = 127$ (the length of the LFSR), $t = 2$ (taps) for ABC v2. The probability that w_i satisfies at least h of the m relations equals

$$Q(h) = \sum_{i=h}^{m} \binom{m}{i} \cdot (p \cdot s^i \cdot (1-s)^{m-i} + (1-p) \cdot (1-s)^i \cdot s^{m-i}) . \tag{13}$$

If $u_i = w_i$, then the probability that w_i satisfies h of these m relations is equal to

$$R(h) = \sum_{i=h}^{m} \binom{m}{i} \cdot p \cdot s^h \cdot (1-s)^{m-h} . \tag{14}$$

According to [9], $N \cdot Q(h)$ is the number of u_i's being predicted in the attack, and $N \cdot R(h)$ is the number of u_i's being correctly predicted.

For $N = 4500$, there are on average about 12 relations for each w_i. For $h = 11$, 98.50 bits can be predicted with 98.32 bits being predicted correctly. For $h = 10$, 384.99 bits can be predicted with 383.21 bits being predicted correctly. To predict 127 bits, we can predict 98.50 bits for $h = 11$, then predict 127-98.50 = 28.50 bits using the w_i's satisfying only 10 relations. Then in average there are $98.32 + 28.50 \times \frac{383.21-98.31}{384.99-98.50} = 126.66$ bits being predicted correctly. It shows that 127 u_i's can be determined with about 0.34 wrong bits. Then the LFSR can be recovered by solving 127 linear equations.

We carry out an experiment to verify the above analysis. In order to reduce the programming complexity, we consider only the w_i's with 12 relations, thus we use 8000 outputs in the experiment. Using more outputs to recover the LFSR has no effect on the overall attack since recovering the component B requires about $2^{17.5}$ outputs, as shown in Sect. 4.2.

Experiment 2. From the weak key (fe 39 b5 c7 e6 69 5b 44 00 00 00 00 00 00 00 00), we generate 8000 outputs, but consider only those $8000 - 2 \cdot 2^{\frac{12}{3}} \cdot 127 = 3936$ w_i's with 12 relations. We repeat the experiments 256 times with different IVs. For $h = 11$, 104.66 bits can be predicted with 104.35 bits being predicted correctly. For the w_i's satisfying only 10 relations, 278.37 bits can be predicted with 276.08 bits being predicted correctly. To predict 127 bits, $127 - 104.66 = 22.34$ w_i's satisfying only 10 relations should be used. Among the 127 predicted bits, $104.35 + 22.34 \times \frac{276.08}{278.37} = 126.51$ bits are correct.

4.2 Recovering the Components B and C

After recovering the LFSR, we proceed to recover the component B. In the previous attack on ABC v1 [3], about 2^{77} operations are required to recover the components B and C. That complexity is too high. We give here a very simple method to recover the components B and C with about $2^{33.3}$ operations.

In ABC v2, there are four secret terms in the component B: x, d_0, d_1, and d_2, where d_0, d_1 and d_2 are static, x is updated as

$$x_i = (((x_{i-1} \oplus d_0) + d_1) \oplus d_2) + \overline{z}_3^i \bmod 2^{32}. \qquad (15)$$

Note that the higher significant bits never affect the less significant bits. It allows us to recover x, d_0, d_1, and d_2 bit-by-bit.

Since the initial value of the LFSR is known, the value of each $\overline{z}_0^i$ can be computed, thus we know the value of each $S(x_i)$. In average, the probability that $x_i = x_j$ is about 2^{-32}. For a weak key, the least significant bit of $S(x_i)$ is always 0, and the probability that $S(x_i) = S(x_j)$ is about $2^{-32} + (1 - 2^{-32}) \cdot 2^{-31} \approx 2^{-32} + 2^{-31}$. Given $2^{17.5}$ outputs, there are about $\binom{2^{17.5}}{2} \times (2^{-32} + 2^{-31}) \approx 12$ cases that $S(x_i) = S(x_j)$ $(i \neq j)$. And there are about $\binom{2^{17.5}}{2} \times 2^{-32} \approx 4$ cases that $x_i = x_j$ among those 12 cases. Choose four cases from those 12 cases randomly, the probability that $x_{i_u} = x_{j_u}$ for $0 \leq u < 4$ is $(\frac{4}{12})^4 = \frac{1}{81}$ (here (i_u, j_u) indicates one of those 12 pairs (i, j) satisfying $S(x_i) = S(x_j)$ $(i \neq j)$).

The value of each $\overline{z}_3^i$ in (15) is already known. When we solve the four equations $x_{i_u} = x_{j_u}$ $(0 \leq u < 4)$ to recover x, d_0, d_1, and d_2, we obtain the four unknown terms bit-by-bit from the least significant bit to the most significant bit. The four most significant bits cannot be determined exactly, but the four least significant bits can be determined exactly since only the least significant bit of x is unknown. (We mention here during this bit-by-bit approach, the four bits at each bit position may not be determined exactly, and further filtering is required in the consequent computations.) On average, we expect that solving each set of four equations gives about 8 possible values of x, d_0, d_1, and d_2. Also note that each set of four equations holds true with probability $\frac{1}{81}$, we have about $81 \times 8 = 648$ possible solutions for x, d_0, d_1, and d_2.

After recovering the component B, we know the input and output of each $S(x_i)$, so the component C can be recovered by solving 32 linear equations. This process is repeated 648 times since there are about 648 possible solutions of x, d_0, d_1, and d_2. The exact B and C can be determined by generating some outputs and comparing them to the keystream.

4.3 The Complexity of the Attack

According to the experiment, recovering the LFSR requires about 8000 outputs. For each w_i, testing 12 relations requires about $\frac{12\cdot 2}{3} = 8$ XORs and 12 additions. After predicting 127 u_i's, each u_i should be expressed in terms of the initial state of the LFSR. It is almost equivalent to running the LFSR $8000\cdot 32$ steps, with the LFSR being initialized with only one non-zero bit $\overline{z}^0_{1,31}$. Advancing the LFSR 32 steps requires 2 XORs and 2 shifts. Solving a set of 127 binary linear equations requires about $\frac{2\cdot 127^3}{3}\cdot\frac{1}{32}\approx 42675$ operations on the 32-bit microprocessor. So about $2^{17.8}$ operations are required to recover the LFSR.

Recovering the component B requires about $2^{17.5}$ outputs and solving 81 sets of equations. Each set of equations can be solved bit-by-bit, and it requires about $32\cdot 2^4\cdot 2^{17.5} = 2^{26.5}$ operations. Recovering the component C requires solving 648 sets of equations. Each set of equations consists of 32 linear equations with binary coefficients, and solving it is almost equivalent to inverting a 32×32 binary matrix which requires about $\frac{2\cdot 32^3}{3}\cdot\frac{1}{32}\approx 683$ operations. So $81\cdot 2^{26.5}+648\cdot 683 = 2^{32.8}$ operations are required to recover the components B and C.

Recovering the internal state of a weak key requires $2^{17.5}$ outputs and $2^{17.8} + 2^{32.8} = 2^{32.8}$ operations in total.

4.4 The Attack on ABC v1

The previous attack on ABC v1 deals with a general type of weak keys [3], but the complexity is too high (2^{95} operations). The above attack can be slighty modified and applied to break ABC v1 (with the weak keys defined in Sect. 3) with much lower complexity. We outline the attack on ABC v1 below.

The LFSR in ABC v1 is 63 bits. The shorter LFSR results in more relations for the same amount of keystream. Identifying a weak key requires 1465 outputs from each key instead of the 1615 outputs required in the attack on ABC v2. In theory, recovering the LFSR with the fast correlation attack requires 2500 outputs instead of the 4500 outputs required in the attack on ABC v2. The component B in ABC v1 has only three secret variables. Recovering the component B requires $2^{17.3}$ outputs, with the complexity reduced to $2^{30.1}$ operations, smaller than the $2^{32.8}$ operations required to recover the component B of ABC v2. In total the attack to recover the internal state of ABC v1 with a weak key requires $2^{17.3}$ outputs and $2^{30.1}$ operations.

5 Conclusion

Due to the large amount of weak keys and the serious impact of each weak key, ABC v1 and ABC v2 are practically insecure.

In order to resist the attack presented in this paper, a straightforward solution is to ensure that at least one of the least significant bits of the 33 elements in the component B should be nonzero. However, ABC v2 with such improvement is still insecure. A new type of weak keys with all the two less (but not least) significant bits being 0 still exists. After eliminating all the similar weak keys, the linear relation in (2) can still be applied to exploit the non-randomness in the outputs of the component C to launch a distinguishing attack. ABC v3 is the latest version of ABC, and it eliminates the weak keys described in this paper. However, a recent attack exploiting the non-randomness in the outputs of the component C is still able to identify a new weak key with about 2^{60} outputs [15]. It seems difficult to improve the ABC cipher due to the risky design that the 32-bit-to-32-bit S-box is generated from only 33 key-dependent elements.

We recommend updating the secret S-box of ABC v2 frequently during the keystream generation process. In ABC v2, the key-dependent S-box is fixed. For block cipher design, the S-box has to remain unchanged, but such restriction is not applicable to stream cipher. Suppose that the size of the key-dependent S-box of a stream cipher is large (it is risky to use the small randomly generated key-dependent S-box). We can update the S-box frequently, such as updating at least one element of the S-Box at each step (in a cyclic way to ensure that all the elements of the S-box get updated) with enough random information in an unpredictable way. When a weak S-box appears, only a small number of outputs are generated from it before the weak S-box disappears, and it becomes extremely difficult for an attacker to collect enough outputs to analyze a weak S-box. Thus an attacker has to deal with the average property of the S-box, instead of dealing with the weakest S-box. For example, the eSTREAM submissions HC-256 [13], HC-128 [14], Py [4] and Pypy [5] use the frequently updated large S-boxes to reduce the effect resulting from the weak S-boxes. The security of ABC stream cipher can be improved in this way, but its performance will be affected.

Acknowledgements. The authors would like to thank the anonymous reviewers of SAC 2006 for their helpful comments. Thanks also go to Adi Shamir and others for their comments on the security of updating a key-dependent S-box at the SAC 2006 conference.

References

1. Anashin, V., Bogdanov, A., Kizhvatov, I.: ABC: A New Fast Flexible Stream Cipher. Available at http://www.ecrypt.eu.org/stream/ciphers/abc/abc.pdf
2. Anashin, V., Bogdanov, A., Kizhvatov, I.: Security and Implementation Properties of ABC v.2. SASC 2006 - Stream Ciphers Revisited, pp. 278–292, (2006), Available at http://www.ecrypt.eu.org/stream/papersdir/2006/026.pdf
3. Berbain, C., Gilbert, H.: Cryptanalysis of ABC. Available at http://www.ecrypt.eu.org/stream/papersdir/048.pdf
4. Biham, E., Seberry, J.: Py: A Fast and Secure Stream Cipher Using Rolling Arrays. Available at http://www.ecrypt.eu.org/stream/p2ciphers/py/py_p2.ps
5. Biham, E., Seberry, J.: Pypy: Another Version of Py. Available at http://www.ecrypt.eu.org/stream/p2ciphers/py/pypy_p2.ps
6. Chepyzhov, V.V., Johansson, T., Smeets, B.: A Simple Algorithm for Fast Correlation Attacks on Stream Ciphers. In: Schneier, B. (ed.) FSE 2000. LNCS, vol. 1978, pp. 181–195. Springer, Heidelberg (2001)

7. Johansson, T., Jönsson, F.: Fast Correlation Attacks through Reconstruction of Linear Polynomials. In: Bellare, M. (ed.) CRYPTO 2000. LNCS, vol. 1880, pp. 300–315. Springer, Heidelberg (2000)
8. Khazaei, S.: Divide and Conquer Attack on ABC Stream Cipher. Available at http://www.ecrypt.eu.org/stream/papersdir/052.pdf
9. Meier, W., Staffelbach, O.: Fast Correlation Attacks on Stream Ciphers. Journal of Cryptology 1(3), 159–176 (1989)
10. Mihaljević, M., Fossorier, M.P.C., Imai, H.: A Low-Complexity and High-Performance Algorithm for Fast Correlation Attack. In: Schneier, B. (ed.) FSE 2000. LNCS, vol. 1978, pp. 196–212. Springer, Heidelberg (2001)
11. Murphy, S., Robshaw, M.J.B.: Key-dependent S-boxes and differential cryptanalysis. Designs, Codes, and Cryptography 27(3), 229–255 (2002)
12. Vaudenay, S.: On the Weak Keys of Blowfish. In: Gollmann, D. (ed.) Fast Software Encryption. LNCS, vol. 1039, pp. 27–32. Springer, Heidelberg (1996)
13. Wu, H.: A New Stream Cipher HC-256. In: Roy, B., Meier, W. (eds.) FSE 2004. LNCS, vol. 3017, pp. 226–244. Springer, Heidelberg (2004), Full version available at http://eprint.iacr.org/2004/092.pdf
14. Wu, H.: The Stream Cipher HC-128. Available at http://www.ecrypt.eu.org/stream/p2ciphers/hc256/hc128_p2.pdf
15. Zhang, H., Li, L., Wang, X.: Fast Correlation Attack on Stream Cipher ABC v3 (2006), Available at http://www.ecrypt.eu.org/stream/papersdir/2006/049.pdf
16. Zhang, H., Wang, S., Wang, X.: The Probability Advantages of Two Linear Expressions in Symmetric Ciphers (2006), Available at http://www.ecrypt.eu.org/stream/papersdir/2006/046.pdf

The Design of a Stream Cipher LEX

Alex Biryukov

University of Luxembourg, FDEF,
Campus Limpertsberg, 162 A, Avenue de la Faiencerie
L-1511 Luxembourg, Luxembourg
alex.biryukov@uni.lu

Abstract. In this paper we define a notion of leak extraction from a block cipher. We demonstrate this new concept on an example of AES. A result is LEX: a simple AES-based stream cipher which is at least 2.5 times faster than AES both in software and in hardware.

1 Introduction

In this paper we suggest a simple notion of a *leak extraction* from a block cipher. The idea is to extract parts of the internal state at certain rounds and give them as the output key stream (possibly after passing an additional filter function). This idea applies to any block cipher but a careful study by cryptanalyst is required in each particular case in order to decide which parts of the internal state may be given as output and at what frequency. This mainly depends on the strength of the cipher's round function and on the strength of the cipher's key-schedule. For example, ciphers with good diffusion might allow to output larger parts of the internal state at each round than ciphers with weak diffusion.

In this paper we describe our idea on an example of 128/192/256 bit key AES. Similar approach may be applied to the other block-ciphers, for example to Serpent. Interesting lessons learnt from LEX so far are that: LEX setup and resynchronization which are just a single AES key-setup and a single AES encryption are much faster than for most of the other stream ciphers (see performance evaluation of eSTREAM candidates [8]). This is due to the fact that many stream ciphers aimed at fast encryption speed have a huge state which takes very long time to initialize. Also, the state of the stream ciphers has to be at least double of the keysize in order to avoid tradeoff attacks, but on the other hand it does not have to be more than that. Moreover unlike in a typical stream cipher, where all state changes with time, in LEX as much as half of the state does not need to be changed or may evolve only very slowly.

2 Description of LEX

In this section we describe a 128-bit key stream cipher LEX (which stands for Leak EXtraction, and is pronounced "leks"). In what follows we assume that the reader is familiar with the Advanced Encryption Standard Algorithm (AES) [7].

© Springer-Verlag Berlin Heidelberg 2007

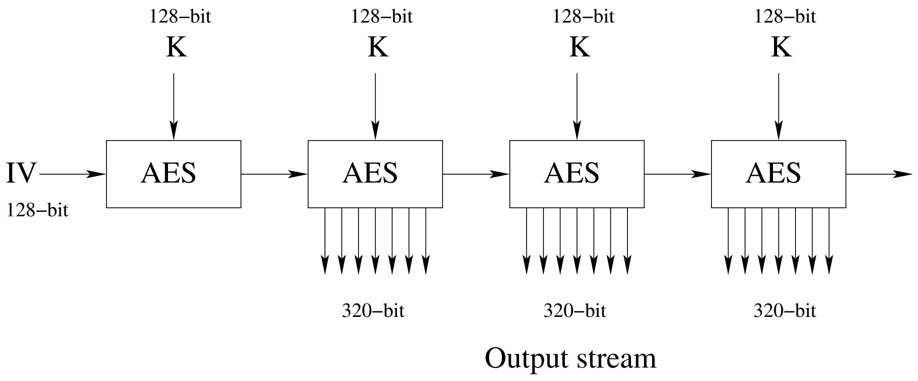


Fig. 1. Initialization and stream generation

The design is simple and is using AES in a natural way: at each AES round we output certain four bytes from the intermediate state. The AES with all three different key lengths (128, 192, 256) can be used. The difference with AES is that the attacker never sees the full 128-bit ciphertext but only portions of the intermediate state. Similar principle can be applied to any other block-cipher.

In Fig. 1 we show how the cipher is initialized and chained[1]. First a standard AES key-schedule for some secret 128-bit key K is performed. Then a given 128-bit IV is encrypted by a single AES invocation: $S = AES_K(IV)$. The 128-bit result S together with the secret key K constitute a 256-bit secret state of the stream cipher.[2] S is changed by a round function of AES every round and K is kept unchanged (or in a more secure variant is changing every 500 AES encryptions).

The most crucial part of this design is the exact location of the four bytes of the internal state that are given as output as well as the frequency of outputs (every round, every second round, etc.). So far we suggest to use the bytes $b_{0,0}, b_{2,0}, b_{0,2}, b_{2,2}$ at every odd round and the bytes $b_{0,1}, b_{2,1}, b_{0,3}, b_{2,3}$ at every even round. We note that the order of bytes is not relevant for the security but is relevant for the fast software implementation. The order of bytes as given above allows to extract a 32-bit value from two 32-bit row variables t_0, t_2 in just four operations (that can be pipelined):

$$out32 = ((t_0 \& 0xFF00FF) << 8) \oplus (t_2 \& 0xFF00FF),$$

while each round of AES uses about 40 operations. Here t_i is a row of four bytes: $t_i = (b_{i,0}, b_{i,1}, b_{i,2}, b_{i,3})$. So far we do not propose to use any filter function and output the bytes as they are. The choice of the output byte locations (see also Fig. 2) is motivated by the following: both sets constitute an invariant subset of

[1] There is a small caveat: we use full AES to encrypt the IV, but we use AES with slightly modified last round for the stream generation, as will be explained further in this section.

[2] In fact the K part is expanded by the key-schedule into ten 128-bit subkeys.

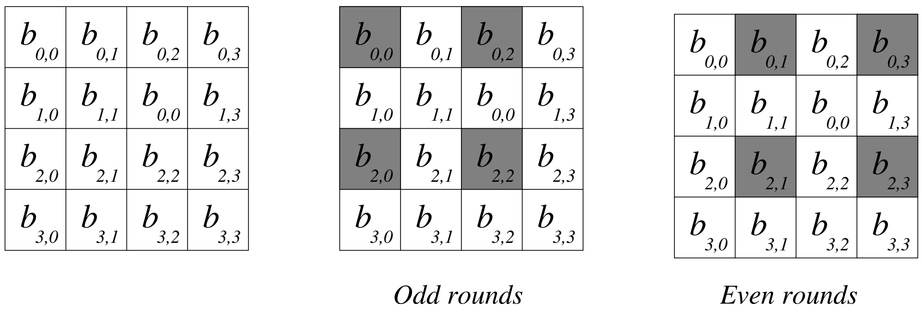


Fig. 2. The positions of the leak in the even and in the odd rounds

the `ShiftRows` operation (the first row is not shifted and the third is rotated by two bytes). By alternating the two subsets in even and odd rounds we ensure that the attacker does not see input and output bytes that are related by a single `SubBytes` and a single `MixColumn`. This choice ensures that the attacker will have to analyze two consecutive rounds. The two rounds of AES have full diffusion thus limiting divide-and-conquer capabilities of the attacker. Note also that in AES the 10th round differs from the rest, there is no `MixColumn` and there is a XOR of the last (11th) subkey. In LEX there is no need to make the 10th round different from any other round. Any LEX encryption round consists of:

```
Round(State, i)
{ SubBytes(State);
  ShiftRows(State);
  MixColumns(State);
  AddRoundKey(State, ExpandedKey[i mod N_r]);
}
```

Here N_r is the number of rounds and is equal to 10 for 128-bit key AES. The full T iterations of LEX would then look like:

```
LEX(State, SecretKey)
{
AESKeyExpansion(SecretKey, ExpandedKey);
State = AESEncrypt(IV, ExpandedKey);
AddroundKey(State, ExpandedKey[0]);
for (i=1; i < T; i++){
  Round(State, i);
  Output[i] = LeakExtract(State, i mod 2);
 }
}
```

It is advisable to change the *SecretKey* at least every 2^{32} IV setups, and to change the IV every $T = 500$ iterations.

Note also that IV setup is performed by full AES encryption and the subtle difference in the last round of AES and absence of such difference in encryption rounds of LEX is crucial to break similarity which otherwise could be exploited by slide attacks [5, 11] (see Section 3.8 for a discussion).

The speed of this cipher is more than 2.5 times faster than 128-bit key AES, 3 times faster than 192-bit key AES, and 3.5 times faster than 256-bit key AES. So far there are no weaknesses known to the designers as well as there are no hidden weaknesses inserted by the designers.

3 Analysis of LEX

In this section we analyze resistance of LEX to various attacks.

3.1 Period of the Output Sequence

The way we use AES is essentially an Output Feedback Mode (OFB), in which instead of using the ciphertexts as a key-stream we use the leaks from the intermediate rounds as a key-stream. The output stream will eventually cycle when we traverse the full cycle of the AES-generated permutation. If one assumes that AES is indistinguishable from a random permutation for any fixed key, one would expect the cycle size to be of the order $O(2^{128})$ since the probability of falling into one of the short cycles is negligible[3].

If the output stream is produced by following a cycle of a random permutation it is easily distinguished from random after observing absence of 128-bit collisions in a stream of 2^{64} outputs. In our case since we output only part of the state at each round, the mapping from internal state to the output is not 1-1 and thus such collisions would occur.

3.2 Tradeoff Attacks

For a stream cipher to be secure against time-memory and time-memory-data tradeoff attacks [1, 9, 4] the following conditions are necessary: $|K| = |IV| = |State|/2$. This ensures that the best tradeoff attack has complexity roughly the same as the exhaustive key-search. The IV's may be public, but it is very important that full-entropy IV's are used to avoid tradeoff-resynchronization attacks [3, 10]. In the case of LEX $|K| = |IV| = |Block| = 128$ bits, where $Block$ denotes an intermediate state of the plaintext block during the encryption. Internal state is the pair (IV, K) at the start and $(Block, Key)$ during the stream generation, and thus $|K| + |IV| = |K| + |S| = 256$ bits which is enough to avoid the tradeoff attacks. Note that if one uses LEX construction with larger key variants of AES this might be a "problem". For example for 192-bit key AES the state would consist of 128-bit internal variable and the 192-bit key. This would

[3] A random permutation over n-bit integers typically consists of only about $O(n)$ cycles, the largest of them spanning about 62% of the space.

allow to apply a time-memory-data tradeoff attack with roughly 2^{160} stream, memory and time. For 256-bit key AES it would be 2^{192} stream, memory and time. Such attack is absolutely impractical but may be viewed as a certificational weakness.

3.3 Algebraic Attacks

Algebraic attack on stream ciphers [6] is a recent and a very powerful type of attack. Applicability of these to LEX is to be carefully investigated. If one could write a non-linear equation in terms of the outputs and the key – that could lead to an attack. Re-keying every 500 AES encryptions may help to avoid such attacks by limiting the number of samples the attacker might obtain while targeting a specific subkey. We expect that after the re-keying the system of non-linear equations collected by the attacker would become obsolete. Shifting from AES key-schedule to a more robust one might be another precaution against these attacks. Note also that unlike in LFSR-based stream ciphers we expect that there do not exist simple relations that connect internal variables at distances of 10 or more steps. Such relations if they would exist would be useful in cryptanalysis of AES itself.

3.4 Differential, Linear or Multiset Resynchronization Attacks

If mixing of IV and the key is weak the cipher might be prone to chosen or known IV attacks similar to the chosen plaintext attacks on the block-ciphers. However in our case this mixing is performed via a single AES encryption. Since AES is designed to withstand such differential, linear or multiset attacks we believe that such attacks pose no problem for our scheme either.

3.5 Potential Weakness – AES Key-Schedule

There is a simple way to overcome weaknesses in AES key-schedule (which is almost linear) and which might be crucial for our construction. One might use ten consecutive encryptions of the IV as subkeys, prior to starting the encryption. This method will however loose in key agility, since key-schedule time will be 11 AES encryptions instead of one. If better key-agility is required a faster dedicated key-schedule may be designed.

If bulk encryption is required then it might be advisable to replace the static key with a slowly time-varying key. One possibility would be to perform an additional 10 AES encryptions every 500 AES encryptions and to use the 10 results as subkeys. This method is quite efficient in software but might not be suitable for small hardware due to the requirement to store 1280 bits (160 bytes) of the subkeys. The overhead of such key-change is only 2% slowdown, while it might stop potential attacks which require more than 500 samples gathered for a specific subkey. An alternative more gate-efficient solution would be to perform a single AES encryption every 100 steps without revealing the intermediate values and use the result as a new 128-bit key. Then use the keyschedule of AES to generate the subkeys. Note, that previously by iterating AES with the same key we explored a single cycle of AES, which was likely to be of length $O(2^{128})$ due to the cipher being a permutation of 2^{128} values. However by doing intermediate key-changes we are now in a random mapping scenario. Since state size of our random mapping is 256 bits (key + internal state), one would expect to get into a "short cycle" in about $O(2^{128})$ steps, which is the same as in the previous case and poses no security problem.

3.6 No Weak Keys

Since there are no weak keys known for the underlying AES cipher we believe that weak keys pose no problem for this design either. This is especially important since we suggest frequent rekeying to make the design more robust against other cryptanalytic attacks.

3.7 Dedicated Attacks

An obvious line of attack would be to concentrate on every 10th round, since it reuses the same subkey, and thus if the attacker guesses parts of this subkey he still can reuse this information $10t, t = 1, 2, \ldots$ rounds later. Note however that unlike in LFSR or LFSM based stream ciphers the other parts of the intermediate state have hopelessly changed in a complex non-linear manner and any guesses spent for those are wasted (unless there is some weakness in a full 10-round AES).

3.8 The Slide Attack

In [11] a slide attack [5] on resynchronization mechanism of LEX (as it was described for the eSTREAM project) is shown. The attack requires the ability to perform 2^{61} resynchronizations and uses 2^{75} bytes of output stream data produced under a single key and different IVs, which need to be stored and sorted in 2^{75} bytes of memory. This attack is comparable in complexity to time-memory-key tradeoff attacks which are applicable to any block cipher in popular modes of operation like ECB, CBC (time-memory-data complexity of $O(2^{64})$ for any 128-bit cipher) [2, 3][4] This attack thus does not make LEX weaker than 128-bit key AES.

However the observation leading to the attack is of interest since it can be easily generalized and would apply to any leak-extraction cipher in which resynchronization and encryption are performed by the same function. The idea of the

[4] One may argue that attack on a single key is more interesting than the tradeoff attack that breaks one key out of 2^{64}. Firstly we think that it is subjective and depends on the appliation. Secondly, if we limit the amount of stream produced per key to 2^{32} as is typical for many other stream-ciphers, this argument will not be valid any more. The slide attack will have 2^{96} complexity and will need to try the same amount of keys as the tradeoff attack – 2^{64}, before it succeeds.

attack is simple: iterations of LEX explore a cycle of the size about 2^{128} starting from IV. Random IV selections would sample random points on this cycle. If the IV setup is performed by the same function as the subsequent stream generation then one may pick an IV which is equal to the block-state just after the IV setup of another sample. This causes the attacker to know the full block input of the cipher and the result of the leak one round later, which clearly leaks lots of information about the secret subkey of that round. In order to find such colliding block-states the attacker needs at least 2^{65} block samples stored and sorted in memory. The attack assumes the ability to perform about 2^{64} resynchronizations for the same key.

A natural way to increase resistance against the attack would be to require a change of keys every 2^{32} IV's. There would still remain a chance of 2^{-64} to find colliding block-states in a collection of 2^{32} IV samples. However the complexity of the attack would increase to 2^{96} and the attacker would need to try the attack for 2^{64} different keys – the same number as in the tradeoff attack. Such high complexity should be a sufficient protection for most of the practical purposes. In addition, in order to completely get rid of the sliding property one should use two different functions for the resynchronization and the encryption. Moreover even a small difference between the two would suffice. For example, if one uses the full AES with the XOR of the last subkey for the IV setup and AES without the XOR of this subkey for the encryption – this is enough to break the similarities used by sliding.

4 Implementation

As one may observe from software performance test done by ECRYPT [8], LEX holds to its promise and runs 2.5 times faster than 128-bit key AES. We expect that the same holds for hardware implementations. It is also somewhat pleasantly surprising that LEX is one of the fastest ciphers out of the 32 candidates on many of the platforms: 6th on Intel Pentium M, 1700MHz; 4th on Intel Pentium 4, 2.40GHz; 6th on AMD Athlon 64 3000+, 1.80GHz; 7th on PowerPC G4 533MHz; 6th on Alpha EV5.6, 400MHz; 5th on HP 9000/785, 875MHz; 5th on UltraSPARC-III, 750MHz). It is also one of the best in terms of agility of the key-setup, the IV-setup, and the combined Internet packet metric IMIX. LEX is thus very well suited for the short packet exchanges typical for the Internet environment.

Since LEX could reuse existing AES implementations it might provide a simple and cheap speedup option in addition to the already existing base AES encryption. For example, if one uses a fast software AES implementation which runs at 14-15 clocks per byte we may expect LEX to be running at about 5-6 clocks per byte. The same leak extraction principle naturally applies to 192 and 256-bit AES resulting in LEX-192 and LEX-256. LEX-192 should be 3 times faster than AES-192, and LEX-256 is 3.5 times faster than AES-256. Note that unlike in AES the speed penalty for using larger key versions is much smaller in LEX (a slight slowdown for a longer keyschedule and resynchronization but not for the stream generation).

5 Strong Points of the Design

Here we list some benefits of using this design:

- AES hardware/software implementations can be reused with few simple modifications. The implementors may use all their favorite AES implementation tricks.
- The cipher is at least 2.5 times faster than AES. In order to get an idea of the speed of LEX divide cycles-per-byte performance figures of AES by a factor 2.5. The speed of key and IV setup is equal to the speed of AES keyschedule followed by a single AES encryption. In hardware the area and gate count figures are essentially those of the AES.
- Unlike in the AES the key-setup for encryption and decryption in LEX are the same.
- The cipher may be used as a speedup alternative to the existing AES implementation and with only minor changes to the existing software or hardware.
- Security analysis benefits from existing literature on AES.
- The speed/cost ratio of the design is even better than for the AES and thus it makes this design attractive for both fast software and fast hardware implementations. The design will also perform reasonably well in restricted resource environments.
- Since this design comes with explicit specification of IV size and resynchronization mechanism it is secure against time-memory-data tradeoff attacks. This is not the case for the AES in ECB mode or for the AES with IV's shorter than 128-bits.
- Side-channel attack countermeasures developed for the AES will be useful for this design as well.

6 Summary

In this paper we have suggested a new concept of conversion of block ciphers into stream ciphers via *leak extraction*. As an example of this approach we have described efficient extensions of AES into the world of stream ciphers, which we called LEX. We expect that (if no serious weaknesses would be found) LEX may provide a very useful speedup option to the existing base implementations of AES. We hope that there are no attacks on this design faster than $O(2^{128})$ steps. The design is rather bold and of course requires further study.

Acknowledgment

This paper is a result of several inspiring discussions with Adi Shamir. We would like to thank Christophe De Cannière, Joseph Lano, Ingrid Verbauwhede and other cosix for the exchange of views on the stream cipher design. We also would like to thank anonymous reviewers for comments that helped to improve this paper.

References

[1] Babbage, S.: Improved "exhaustive search" attacks on stream ciphers. In: ECOS 95 (European Convention on Security and Detection), no. 408 in IEE Conference Publication (May 1995)

[2] Biham, E.: How to decrypt or even substitute DES-encrypted messages in 2^{28} steps. Information Processing Letters 84, 117–124 (2002)

[3] Biryukov, A., Mukhopadhyay, S., Sarkar, P.: Improved Time-Memory Trade-offs with Multiple Data. In: Preneel, B., Tavares, S. (eds.) SAC 2005. LNCS, vol. 3897, Springer, Heidelberg (2006)

[4] Biryukov, A., Shamir, A.: Cryptanalytic time/memory/data tradeoffs for stream ciphers. In: Okamoto, T. (ed.) ASIACRYPT 2000. LNCS, vol. 1976, pp. 1–13. Springer, Heidelberg (2000)

[5] Biryukov, A., Wagner, D.: Slide attacks. In: Knudsen, L.R. (ed.) FSE 1999. LNCS, vol. 1636, pp. 245–259. Springer, Heidelberg (1999)

[6] Courtois, N.T., Meier, W.: Algebraic attacks on stream ciphers with linear feedback. In: Biham, E. (ed.) EUROCRPYT 2003. LNCS, vol. 2656, pp. 345–359. Springer, Heidelberg (2003)

[7] Daemen, J., Rijmen, V.: The design of Rijndael: AES — The Advanced Encryption Standard. Springer, Heidelberg (2002)

[8] eSTREAM. eSTREAM Optimized Code HOWTO, (2005), http://www.ecrypt.eu.org/stream/perf/

[9] Golic, J.D.: Cryptanalysis of alleged A5 stream cipher. In: Fumy, W. (ed.) EUROCRYPT 1997. LNCS, vol. 1233, pp. 239–255. Springer, Heidelberg (1997)

[10] Hong, J., Sarkar, P.: Rediscovery of time memory tradeoffs (2005), http://eprint.iacr.org/2005/090

[11] Wu, H., Preneel, B.: Attacking the IV Setup of Stream Cipher LEX. In: Robshaw, M. (ed.) FSE 2006. LNCS, vol. 4047, Springer, Heidelberg (2006)

Dial C for Cipher

Le chiffrement était presque parfait[⋆]

Thomas Baignères[⋆⋆] and Matthieu Finiasz

EPFL
CH-1015 Lausanne – Switzerland
http://lasecwww.epfl.ch

Abstract. We introduce C, a practical provably secure block cipher with a slow key schedule. C is based on the same structure as AES but uses independent random substitution boxes instead of a fixed one. Its key schedule is based on the Blum-Blum-Shub pseudo-random generator, which allows us to prove that all obtained security results are still valid when taking into account the dependencies between the round keys. C is provably secure against several general classes of attacks. Strong evidence is given that it resists an even wider variety of attacks. We also propose a variant of C with simpler substitution boxes which is suitable for most applications, and for which security proofs still hold.

Keywords: Block Cipher, provable security, AES, Blum-Blum-Shub generator, decorrelation.

1 Introduction

When designing a public key cryptosystem, proving tight security results often requires to rely on hard problems such as factoring or discrete logarithm computation which, by nature, require to manipulate complex objects. When designing block ciphers, speed requirements do not allow to do so. As a consequence, security arguments often rely on heuristic assumptions which, in some cases, might prove wrong. At SAC 2005, Baignères and Vaudenay [5] showed that replacing the substitution boxes of AES by independent perfectly random permutations is enough to prove that 4 rounds are enough to resist linear and differential cryptanalysis and that 10 rounds are enough to resist any iterated attack of order 1. Here, we use the exact same construction, improve some results, and plug in a key schedule based on a provably secure pseudo-random generator. We propose to use the Blum-Blum-Shub pseudo-random generator [15,16] as its security is well established (even for practical parameters), although any provably secure generator (like for example QUAD [7] or any fast construction based on Goldreich and Levin's hard-core predicate [26]) could be used, possibly leading to faster

⋆ Refering to the famous movie by Alfred Hitchcock *Dial M for Murder* [28], this is how the title of this article should be translated to French.

⋆⋆ Supported by the Swiss National Science Foundation, 200021-107982/1.

© Springer-Verlag Berlin Heidelberg 2007

implementations. We obtain C, a block cipher with a slow key schedule, but as fast as AES when it comes to encryption/decryption and provably secure against most common attacks: linear and differential cryptanalysis, iterated attacks of order 1, impossible differentials and presumably algebraic attacks, slide attacks, boomerang attack, and, to a certain extent, saturation attacks. Note that all the security results we obtain take into account the key schedule. To the best of our knowledge, all current iterated block cipher constructions consider in their "security proofs" that the round keys are statistically independent, which is not true in practice as they all derive from the same key.

We start this article with a detailed description of C and of its key schedule. Ensues a review of all security results on C, starting with those which are proven, and going on with some results which, though not proven, seem quite reasonable. We then present a way of considerably speeding up the key schedule while preserving all security results and finish with implementation considerations.

2 The Block Cipher C

In this paper, a *perfectly random permutation* denotes a random permutation uniformly distributed among all possible permutations. Consequently, when referring to a *random permutation*, nothing is assumed about its distribution.

2.1 High Overview

The block cipher $\mathsf{C} : \{0,1\}^{128} \to \{0,1\}^{128}$ is an iterated block cipher. It is made of a succession of *rounds*, all identical in their structure. Each round is parameterized by a *round-key* which is derived from the main 128 bits secret key using a so-called *key schedule algorithm*. The structure of each round is made of a (non-linear) substitution layer followed by a (linear) permutation layer. The non-linear part of the round mixes the key bits with the text bits in order to bring *confusion* (in the sense of [43]). The linear part dissipates the eventual redundancy, bringing *diffusion*. Such an iterated block cipher is often referred to as a *substitution-permutation network* (SPN). Several modern block ciphers (such as AES [21] or SAFER [36]) follow this structure. In what follows, we successively detail the SPN of C and its key schedule algorithm.

2.2 The Substitution-Permutation Network

In a nutshell, C follows the same SPN as AES [21], except that there is no round key addition, that the fixed substitution box is replaced by independent perfectly random permutations, and that the last round of C only includes the non-linear transformation. This construction exactly corresponds to the one studied in [5].

C is made of $r = 10$ *independent* rounds $\mathsf{R}^{(1)}, \ldots, \mathsf{R}^{(r)} : \{0,1\}^{128} \to \{0,1\}^{128}$, so that $\mathsf{C} = \mathsf{R}^{(r)} \circ \cdots \circ \mathsf{R}^{(1)}$. A r round version of C will either be denoted by

$\mathsf{C}[r]$ or simply by C when the number of rounds is clear from the context. Each round considers the 128 bit text input as a four by four array of bytes seen as elements of the finite field $\mathrm{GF}(q)$ where $q = 2^8$. Consequently, if $a \in \{0,1\}^{128}$ denotes some input of the round transformation, we will denote a_ℓ (resp. $a_{i,j}$) the ℓ-th (resp. the $(i+4j)$-th) byte of a for $0 \le \ell \le 15$ (resp. $0 \le i,j \le 3$) and call such an input a *state*. Except for the last one, each round $\mathsf{R}^{(i)}$ successively applies a non-linear transformation $\mathsf{S}^{(i)}$ followed by a linear transformation L so that $\mathsf{R}^{(i)} = \mathsf{L} \circ \mathsf{S}^{(i)}$ for $i = 1, \ldots, r-1$. The last round $\mathsf{R}^{(r)}$ excludes the linear transformation, i.e., $\mathsf{R}^{(r)} = \mathsf{S}^{(r)}$.

The non-linear transformation $\mathsf{S}^{(i)}$ is a set of 16 *independent* and *perfectly random* permutations[1] of $\mathrm{GF}(q)$. Denoting $\mathsf{S}^{(i)} = \{\mathsf{S}_0^{(i)}, \ldots, \mathsf{S}_{15}^{(i)}\}$ the 16 permutations of round i and $a, b \in \{0,1\}^{128}$ the input and the output of $\mathsf{S}^{(i)}$ respectively, we have $b = \mathsf{S}^{(i)}(a) \Leftrightarrow b_\ell = \mathsf{S}_\ell^{(i)}(a_\ell)$ for $0 \le \ell \le 15$. Depending on the level of security/performance one wants to achieve, the round permutations can be *de-randomized* (see Section 5).

The linear transformation L does not depend on the round number. It first applies a rotation to the left on each row of the input state (considered as a four by four array), over four different offsets. A linear transformation is then applied to each column of the resulting state. More precisely, if a, b denote the input and the output of L respectively, we have (considering indices modulo 4)

$$\begin{pmatrix} b_{0,j} \\ b_{1,j} \\ b_{2,j} \\ b_{3,j} \end{pmatrix} = \begin{pmatrix} \mathsf{02} & \mathsf{03} & \mathsf{01} & \mathsf{01} \\ \mathsf{01} & \mathsf{02} & \mathsf{03} & \mathsf{01} \\ \mathsf{01} & \mathsf{01} & \mathsf{02} & \mathsf{03} \\ \mathsf{03} & \mathsf{01} & \mathsf{01} & \mathsf{02} \end{pmatrix} \times \begin{pmatrix} a_{0,j} \\ a_{1,j+1} \\ a_{2,j+2} \\ a_{3,j+3} \end{pmatrix}.$$

2.3 The Key-Schedule Algorithm

Generating a perfectly random permutation of $\{0,1\}^8$. As there are $2^8!$ possible permutations of $\{0,1\}^8$, it is possible to define a one to one mapping between $[0\,;2^8!-1]$ and the set of permutations of $\{0,1\}^8$. The mapping we choose is described in Table 1. We simply need to derive pseudo-random integers in $[0\,;2^8!-1]$ from the 128 bit secret key. As each of the ten rounds involves 16 permutations, we need 160 such integers, representing a total of $160 \cdot \lceil \log_2(2^8!) \rceil = 269440$ pseudo-random bits.

[1] Note that a random 8 bit permutation is usually more biased than the substitution box of AES [42, 52]. However this bias is key-dependent and thus does not represent a threat. Biases on the AES box are independent of the key and thus can help to distinguish (reduced rounds of) AES from the perfect cipher when the key is unknown. Exploiting the strong bias of the substitution boxes of C requires to know the location of this bias, which is impossible without the knowledge of the permutation that was used (i.e., of the key). For instance the maximum ELP of the transformation made of a random key addition followed by the AES substitution box is 2^{-6} whereas the perfectly random substitution boxes we use have a maximum ELP of $1/(q-1) \approx 2^{-8}$. Intuitively, a cipher cannot become weaker when replacing an (arbitrary) random permutation by a perfectly random permutation.

Table 1. Defining a one to one mapping from integers between 0 and $2^8!$ onto the set of permutations of $\{0,1\}^8$

Input: An integer $0 \le \kappa < 2^8!$
Output: A table π of size 256 such that $\pi[0], \dots, \pi[255] \in \{0, \dots, 255\}$ is a permutation of $\{0,1\}^8$ uniquely defined by κ
External Procedure: `EucDiv(a,b)` returns the quotient and remainder of the Euclidean division of a by b.

```
0: $q \leftarrow \kappa$,   $\pi[0] \leftarrow 0$,   $\pi[1] \leftarrow 1$ , ... ,   $\pi[255] \leftarrow 255$
1: for $m = 256$ down to 1
2:     $(q, r) \leftarrow$ EucDiv$(q, m)$
3:     Swap the values of $\pi$ at positions $r$ and $m$
4: end for
```

Deriving an extended key from the secret key

Definition 1. *An extended key of* $\mathsf{C}[r]$ *is a set of* $16 \cdot r$ *integers in* $[0\,;2^8!-1]$.

In order to derive an extended key from the 128 secret key, we need to generate $16 \cdot r$ pseudo-random integers of $[0\,;2^8!-1]$. We propose to use the Blum-Blum-Shub pseudo-random number generator [16].

Definition 2. *A prime p is a strong-prime if $(p-1)/2$ is prime. A prime p is a strong-strong-prime if both p and $(p-1)/2$ are strong-primes.*

Let p and q be two (fixed) 1024-bit strong-strong-prime numbers[2], and let $n = p \cdot q$. Considering the secret key k as a 128 bit integer, let $\{x_i \in \mathbf{Z}_n^* \;:\; i = -1, 0, 1, 2, \dots\}$ be the sequence defined by

$$\begin{cases} x_{-1} = k \cdot 2^{894} + 2^{1023} & \text{and} \\ x_i \;\; = x_{i-1}^2 \bmod n & \text{for } i \ge 0. \end{cases}$$

Let BBS $= a_1 b_1 a_2 b_2 \dots$ be the pseudo-random bit string where $a_i, b_i \in \{0,1\}$ respectively denote the least and most significant[3] bits of x_i. We will use BBS to generate the 160 integers we need.

Dividing the BBS sequence into $\lceil \log_2(2^8!) \rceil$-bit substrings, we can obtain pseudo-random integers in $[0\,;2^{\lceil \log_2(2^8!) \rceil}-1]$, thus sometimes larger than $2^8!$. A naive approach to deal with those too large integers is to discard the substrings leading to such integers, thus having to generate $\lceil \log_2(2^8!) \rceil$ more bits each time this happens. This strategy requires the generation of $160 \cdot 2^{\lceil \log_2(2^8!) \rceil}/2^8! \approx 270\,134$ pseudo-random bits in average. More efficient approaches exits (e.g., discarding only a few bits instead of a whole block), but the improvement in terms of efficiency is not worth the loss in terms of clarity.

[2] Note that strong-strong-primes are always congruent to 3 modulo 4, i.e., are Blum integers. We use strong-strong primes to ensure that the generator will have a long period. See sections 3.5 and 6 for more details.

[3] The most significant bit corresponds to being larger or smaller than $(n-1)/2$.

Table 2. Exact value of $\max_{a\neq 0,b}\mathrm{ELP}^{\mathsf{C}}(a,b)$ (and $\max_{a\neq 0,b}\mathrm{EDP}^{\mathsf{C}}(a,b)$) for various number of rounds

2 rounds	3 rounds	4 rounds	5 rounds	6 rounds	7 rounds	8 rounds	9 rounds
$2^{-33.98}$	$2^{-55.96}$	$2^{-127.91}$	$2^{-127.91}$	$2^{-127.99}$	$2^{-127.99}$	$2^{-128.00}$	$2^{-128.00}$

3 Security Results: What Is Known for Sure

3.1 C Is Resistant to Linear and Differential Cryptanalysis

Linear Cryptanalysis (LC) [38,37,45], aims at uncovering correlations between linear combinations of plaintext and ciphertext bits. It is known that the data complexity of LC is inversely proportional to the linear probability (LP) [17,39]. For given input/output masks $a, b \in \{0,1\}^{128}$ on C, $\mathrm{LP}^{\mathsf{C}}(a,b) = \big(2\Pr[a \bullet X = b \bullet \mathsf{C}(X)] - 1\big)^2$, where the probability is taken over the uniformly distributed input $X \in \{0,1\}^{128}$ and where $\bullet$ denotes a scalar product. $\mathrm{LP}^{\mathsf{C}}(a,b)$ is a function of the random variable C (the randomness coming from the key). The block cipher is considered to be provably secure against LC if, for all input/output masks $a, b \in \{0,1\}^{128}$, the *expected value* $\mathrm{ELP}^{\mathsf{C}}(a,b)$ of the linear probability $\mathrm{LP}^{\mathsf{C}}(a,b)$ (the mean being taken over all possible instances of the block cipher, that is, over all possible keys) is close to the one of the perfect cipher C^*, i.e., close to $1/(q^{16}-1)$ in our case.

Nyberg showed in [41] that the ELP of an iterated block cipher can be expressed as a sum of *linear characteristics*, which means in our case that for any input/output masks $c_0, c_r \in \{0,1\}^{128}$,

$$\mathrm{ELP}^{\mathsf{C}[r]}(c_0,c_r) = \sum_{c_1,..,c_{r-1}} \prod_{i=1}^{r} \mathrm{ELP}^{\mathsf{R}_i}(c_{i-1},c_i) \geq \max_{c_1,..,c_{r-1}} \prod_{i=1}^{r} \mathrm{ELP}^{\mathsf{R}_i}(c_{i-1},c_i) \quad (1)$$

where the sum is taken over all possible input/output masks on the individual rounds of C. Choosing a specific characteristic (i.e., a sequence of masks $c_1, \ldots, c_{r-1}$), it is possible to lower bound the value of $\mathrm{ELP}^{\mathsf{C}[r]}(a,b)$. This is usually enough to attack the block cipher: a lower bound on the ELP gives an upper bound on the number of samples needed to perform the attack. However, such a bound is not enough to prove the security of the system, as the cumulative effect of linear hulls (the set of all intermediate masks for given input/output masks) may lead to an attack much more efficient than expected[4]. In security proofs of block ciphers, it is often considered without any formal justification that one characteristic is overwhelming, so that the sum in (1) is of same order than the max. In our case, for any input/output masks $a, b \in \{0,1\}^{128}$, the *exact* value of $\mathrm{ELP}^{\mathsf{C}}(a,b)$ can be made arbitrarily close to the ELP of the perfect cipher by taking a sufficient number of rounds. It is also possible to compute

[4] Most block cipher designers choose to compensate for possible hull effects by adding an arbitrary number of rounds. This is the case for AES [21], Camellia [2], CAST256 [1], Crypton [34], CS-Cipher [44], FOX [30] and many others.

the exact value of the ELP (see Table 2), and thus determine the exact minimal number of rounds required to resist LC. See [5] for a proof of these results.

Differential Cryptanalysis (DC) [11,12] looks for input/output difference pairs occurring with non-negligible probability for the block cipher. The efficiency of DC is inversely proportional to the *differential probability* defined by $\mathrm{DP}^{\mathsf{C}}(a, b) = \Pr[\mathsf{C}(X \oplus a) = \mathsf{C}(X) \oplus b]$ for any input/output differences $a, b \in \{0,1\}^{128}$ and uniformly distributed $X \in \{0,1\}^{128}$. Similarly to LC, the block cipher is considered to be secure against DC if for all (a, b) pairs, the expected value $\mathrm{EDP}^{\mathsf{C}}(a, b)$ of $\mathrm{DP}^{\mathsf{C}}(a, b)$ is close to that of the perfect cipher C^*, i.e., to $1/(q^{16} - 1)$ in our case.

The development we propose for LC applies similarly for DC. Indeed, in the case of Markov ciphers [32], an equation identical to (1) can be written for the EDP coefficients. The concept of linear hulls translates into the one of differentials. Again, security proofs tend to approximate the differentials using a single differential characteristic. In our case, the EDP can be made arbitrarily close to the optimal value. It is possible to compute the exact value of the EDP (see Table 2), and thus to determine the exact minimal number of rounds to resist DC. See [5] for a proof of these results.

Theorem 3. *Considering* C *on* r *rounds and any non-zero* $a, b \in \{0,1\}^{128}$ *(either considered as input/output masks or as input/output differences), we have*

$$\mathrm{ELP}^{\mathsf{C}[r]}(a, b) \xrightarrow[r\to\infty]{} \mathrm{ELP}^{\mathsf{C}^*}(a, b) \quad \textit{and} \quad \mathrm{EDP}^{\mathsf{C}[r]}(a, b) \xrightarrow[r\to\infty]{} \mathrm{EDP}^{\mathsf{C}^*}(a, b),$$

where $\mathrm{ELP}^{\mathsf{C}^*}(a, b) = \mathrm{EDP}^{\mathsf{C}^*}(a, b) = (q^{16} - 1)^{-1}$. *Moreover, four rounds of* C *are enough to prove its security against linear (resp. differential) cryptanalysis as* $\max_{a\neq 0,b} \mathrm{ELP}^{\mathsf{C}[4]}(a, b) = \max_{a\neq 0,b} \mathrm{EDP}^{\mathsf{C}[4]}(a, b) = 2^{-127.91}$.

3.2 C Is Resistant to Impossible Differentials

Impossible Differentials [8] attacks are a variation of DC. They consist in finding pairs of input/output differences such that for any instance c of C we have $\mathrm{DP}^{\mathsf{c}}(a, b) = 0$. In other words, an input difference of a can never (i.e., for any input and any key) lead to an output difference of b. In the case of C we can prove that five rounds are enough to have no impossible differential[5], i.e., given any input/output masks a and b, there exists an instance c of $\mathsf{C}[5]$ (i.e., a key defining 80 permutations) such that $\mathrm{DP}^{\mathsf{c}}(a, b) \neq 0$.

Definition 4. *Let* $a \in \{0,1\}^{128}$ *be an arbitrary state. The* support *of* a *is a four by four binary array with 1's at the non-zero positions of* a *and 0 elsewhere. It is denoted* $\textsc{supp}(a)$. *The* weight *of the support is denoted* $w(\textsc{supp}(a))$ *or simply* $w(a)$, *and is the Hamming weight of the support. A state is said to be of* full support *when its weight is equal to 16.*

[5] There exists an impossible differential on 4 rounds of AES leading to an attack on 6 rounds [18].

Lemma 5. *Let $a, b \in \{0,1\}^{128}$ be any two differences of full support. One substitution layer S is enough to ensure that there exists an instance s of S such that $\mathrm{DP}^{\mathsf{s}}(a,b) \neq 0$.*

Proof. Considering the two plaintexts 0 and a, we can define the 16 substitution boxes $\mathsf{s}_0, \ldots, \mathsf{s}_{15}$ of one round such that $\mathsf{s}_i(0) = 0$ and $\mathsf{s}_i(a_i) = b_i$. As both a_i and b_i are non-zero (a and b are of full support), both conditions can be verified without being inconsistent with the fact that s_i is a permutation. $\square$

Lemma 6. *Let $a \in \{0,1\}^{128}$ be a non-zero difference of arbitrary support. Considering two full rounds of C (i.e., $\mathsf{C} = \mathsf{L}^{(2)} \circ \mathsf{S}^{(2)} \circ \mathsf{L}^{(1)} \circ \mathsf{S}^{(1)}$), there exists a difference $b \in \{0,1\}^{128}$ of full support and an instance c of C such that $\mathrm{DP}^{\mathsf{c}}(a,b) \neq 0$.*

Proof (sketch). For simplicity reasons, we restrict ourselves to the case where the support of a is of weight 1. Without loss of generality, assume $a_0 \neq 0$ while $a_i = 0$ for $i = 1, \ldots, 15$. We consider the two plaintexts to be 0 and a. Letting $\mathsf{S}_i^{(1)}(0) = 0$ for all i, we have $\mathsf{L}^{(1)} \circ \mathsf{S}^{(1)}(0) = 0$. By carefully choosing $\mathsf{S}_0^{(1)}(a_0)$, we can make sure that $\mathsf{L}^{(1)} \circ \mathsf{S}^{(1)}(a)$ has a support of weight 4 (on the first columns of the four by four array). Proceeding in the same manner in the second round, we can make sure that $\mathsf{C}(0) = 0$ and $b = \mathsf{C}(a)$ is of full support. $\square$

Consider any two differences $a, b \in \{0,1\}^{128}$ and a five round version of $\mathsf{C} = \mathsf{S}^{(5)} \circ \mathsf{L}^{(4)} \circ \mathsf{S}^{(4)} \circ \mathsf{L}^{(3)} \circ \mathsf{S}^{(3)} \circ \mathsf{L}^{(2)} \circ \mathsf{S}^{(2)} \circ \mathsf{L}^{(1)} \circ \mathsf{S}^{(1)}$. From Lemma 6, there exists an instance $\mathsf{c}_{\mathrm{start}}$ of the first two rounds $\mathsf{L}^{(2)} \circ \mathsf{S}^{(2)} \circ \mathsf{L}^{(1)} \circ \mathsf{S}^{(1)}$ and a difference d of full support such that $\mathrm{DP}^{\mathsf{c}_{\mathrm{start}}}(a,d) \neq 0$. Starting from the end, there exists an instance $\mathsf{c}_{\mathrm{end}}$ of $\mathsf{S}^{(5)} \circ \mathsf{L}^{(4)} \circ \mathsf{S}^{(4)} \circ \mathsf{L}^{(3)}$ and a difference e of full support such that $\mathrm{DP}^{\mathsf{c}_{\mathrm{end}}^{-1}}(b,e) \neq 0$, so that $\mathrm{DP}^{\mathsf{c}_{\mathrm{end}}}(e,b) \neq 0$. From Lemma 5, there exists an instance $\mathsf{c}_{\mathrm{mid}}$ of $\mathsf{S}^{(3)}$ such that $\mathrm{DP}^{\mathsf{c}_{\mathrm{mid}}}(d,e) \neq 0$. Consequently, $\mathrm{DP}^{\mathsf{c}_{\mathrm{end}} \circ \mathsf{c}_{\mathrm{mid}} \circ \mathsf{c}_{\mathrm{start}}}(a,b) \neq 0$.

Property 7 (Provable security of C against Impossible Differentials). *Five rounds of C are enough to ensure that no impossible differential exists.*

3.3 C Is Resistant to 2-Limited Adaptive Distinguishers

In the Luby-Rackoff model [35], an adversary has an unbounded computational power and is only limited by its number of queries to an oracle $\mathcal{O}$ implementing a random permutation. Let $\mathcal{A}$ be an adversary in this model. The goal of $\mathcal{A}$ is to guess whether $\mathcal{O}$ is implementing an instance drawn uniformly among the permutations defined by the block cipher C or among all possible permutations, knowing that these two events are equiprobable and that one of them is eventually true. Denoting C^* a perfectly random permutation on $\{0,1\}^{128}$ (i.e., C^* is the perfect cipher), the ability of the adversary to succeed is measured by means of its *advantage*.

Definition 8. *The* advantage *of an adversary $\mathcal{A}$ of distinguishing two random permutations P_0 and P_1 is defined by*

$$\mathrm{Adv}_{\mathcal{A}}(P_0, P_1) = \Pr\left[\mathcal{A}(P_0) = 0\right] - \Pr\left[\mathcal{A}(P_1) = 0\right].$$

In this model, the most powerful adversary performs a d-limited adaptive attack, where d denotes the number of oracle queries. Theorem 14 in [5] gives a loose bound against 2-limited adaptive distinguishers. Using the decorrelation theory, we manage to obtain the *exact* value of the advantage of the best distinguisher.

A Dash of Decorrelation Theory. We briefly recall the results from the decorrelation theory on which our proofs are based. For the sake of simplicity, we restrict to block ciphers defined on $\{0,1\}^{128}$. Given a block cipher B, the d-wise distribution matrix $[B]^d$ is a $2^{128d} \times 2^{128d}$ matrix defined by $[B]^d_{(x_1,\ldots,x_d),(y_1,\ldots,y_d)} = \Pr_B[B(x_1) = y_1, \ldots, B(x_d) = y_d]$. Theorem 10 in [47] tells us that the advantage of the best d-limited *non-adaptive* distinguisher is given by

$$\begin{aligned}\mathrm{Adv}_{\mathcal{A}_{\mathrm{na}}}(B, \mathsf{C}^*) &= \frac{1}{2}|||[B]^d - [\mathsf{C}^*]^d|||_\infty \\ &= \frac{1}{2}\max_{x_1}\cdots\max_{x_d}\sum_{y_1}\cdots\sum_{y_d}\left|[B]^d_{(x_1,\ldots,x_d),(y_1,\ldots,y_d)} - [\mathsf{C}^*]^d_{(x_1,\ldots,x_d),(y_1,\ldots,y_d)}\right|.\end{aligned}$$

Similarly, Theorem 11 in [47] gives the advantage of the best d-limited *adaptive* distinguisher

$$\begin{aligned}\mathrm{Adv}_{\mathcal{A}}(B, \mathsf{C}^*) &= \frac{1}{2}\|[B]^d - [\mathsf{C}^*]^d\|_a \\ &= \frac{1}{2}\max_{x_1}\sum_{y_1}\cdots\max_{x_d}\sum_{y_d}\left|[B]^d_{(x_1,\ldots,x_d),(y_1,\ldots,y_d)} - [\mathsf{C}^*]^d_{(x_1,\ldots,x_d),(y_1,\ldots,y_d)}\right|.\end{aligned}$$

Finally, if A and B are two independent random permutations, $[A \circ B]^d = [A]^d \times [B]^d$. For an iterated block cipher with r independent rounds, it is thus enough to compute the distribution matrix of one round and to raise it to the power r.

Computing $[\mathbf{C}]^2$. C is built as a succession of independent substitution and linear layers $\mathsf{S}^{(r)} \circ \mathsf{L} \circ \mathsf{S}^{(r-1)} \circ \cdots \circ \mathsf{L} \circ \mathsf{S}^{(1)}$. Therefore, as all the substitution layers have the same distribution matrix $[\mathsf{S}]^2$, the distribution matrix of C is given by $[\mathsf{C}]^2 = [\mathsf{S}]^2 \times [\mathsf{L}]^2 \times [\mathsf{S}]^2 \times \cdots \times [\mathsf{L}]^2 \times [\mathsf{S}]^2$.

Let $q = 2^8$ be the size of the field. For a perfectly random substitution box S we have $\Pr[S(u) = v \cap S(u') = v'] = q^{-1}$ if $u = u'$ and $v = v'$, $\Pr[S(u) = v \cap S(u') = v'] = q^{-1}(q-1)^{-1}$ if $u \neq u'$ and $v \neq v'$, and 0 otherwise. As the 16 substitution boxes of S are independent, we obtain

$$[\mathsf{S}]^2_{(x,x'),(y,y')} = \mathbf{1}_{\textsc{supp}(x\oplus x')=\textsc{supp}(y\oplus y')} q^{-16}(q-1)^{-w(x\oplus x')},$$

where we recall that $w(x \oplus x')$ denotes the Hamming weight of the support of $x \oplus x'$. We note that $[\mathsf{S}]^2$ only depends on the respective supports of the input and output differences. We will use this property to dramatically reduce the size of the matrices we have to manipulate. Denoting SP the $2^{256} \times 2^{16}$ matrix such that $SP_{(u,u'),\gamma} = \mathbf{1}_{\textsc{supp}(u\oplus u')=\gamma}$ and PS the $2^{16} \times 2^{256}$ matrix such that $PS_{\gamma,(u,u')} = \mathbf{1}_{\textsc{supp}(u\oplus u')=\gamma}\, q^{-16}(q-1)^{-w(\gamma)}$, we obtain $PS \times SP = Id$ and

$SP \times PS = [\mathsf{S}]^2$. As the last round of C misses the linear operation, we deduce that $[\mathsf{C}]^2 = SP \times \overline{\mathsf{L}}^{r-1} \times PS$, where $\overline{\mathsf{L}} = PS \times [\mathsf{L}]^2 \times SP$ is a $2^{16} \times 2^{16}$ matrix indexed by supports. Noting that $[\mathsf{L}]^2_{(x,x'),(y,y')} = \mathbf{1}_{\mathsf{L}(x)=y}\mathbf{1}_{\mathsf{L}(x')=y'}$ and using the fact that L is linear, it is possible to expand the expression of $\overline{\mathsf{L}}$ and obtain $\overline{\mathsf{L}}_{\gamma,\gamma'} = (q-1)^{-w(\gamma)} \sum_u \mathbf{1}_{\text{SUPP}(u)=\gamma}\mathbf{1}_{\text{SUPP}(\mathsf{L}(u))=\gamma'}$. The matrix $\overline{\mathsf{L}}$ happens to be *precisely* the one used in the expression of the expected linear probability of C given in Theorem 6 in [5]. With our notations, the theorem states that for all support γ, γ' and any states u, u' of respective support γ and γ', we can write $(\overline{\mathsf{L}}^{r-1})_{\gamma,\gamma'} = (q-1)^{w(\gamma')}\text{ELP}^{\mathsf{C}[r]}(u,u')$, where $\text{ELP}^{\mathsf{C}[r]}(u,u')$ is the expected linear probability on r rounds of C given an input (resp. output) mask u (resp. u'). Because ELP^{C} obviously only depends on the supports γ and γ' of u and u', we will denote it from now on $\text{ELP}^{\mathsf{C}}(\gamma,\gamma')$. From this, we easily obtain the following property.

Property 9. *Let $q = 2^8$ and let $\text{ELP}^{\mathsf{C}}(\gamma,\gamma')$ be the expected linear probability of $r > 1$ rounds of C given an input (resp. output) mask of support γ (resp. γ'). The 2-wise distribution matrix of r rounds of C is such that $[\mathsf{C}]^2_{(x,x'),(y,y')} = q^{-16}\,\text{ELP}^{\mathsf{C}}(\text{SUPP}(x \oplus x'), \text{SUPP}(y \oplus y'))$.*

Computing $\mathbf{Adv}_{\mathcal{A}}$ and $\mathbf{Adv}_{\mathcal{A}_{na}}$. The expression we just obtained for $[\mathsf{C}]^2$ leads to the following expression for $\|[\mathsf{C}]^2 - [\mathsf{C}^*]^2\|_a$:

$$\max_x \sum_y \max_{x'} \sum_{y'} \left| q^{-16}\text{ELP}^{\mathsf{C}}(\text{SUPP}(x \oplus x'), \text{SUPP}(y \oplus y')) - [\mathsf{C}^*]^2_{(x,x'),(y,y')} \right| .$$

In the case where $x = x'$, the inner sum of the previous equation is 0 as $q^{-16}\text{ELP}^{\mathsf{C}}(0,0) = [\mathsf{C}^*]^2_{(x,x),(y,y)} = q^{-16}$ and as $\text{ELP}^{\mathsf{C}}(0,\gamma') = [\mathsf{C}^*]^2_{(x,x),(y,y')} = 0$ when $\gamma' = \text{SUPP}(y \oplus y')$ and $y' \neq y$. We obtain

$$\begin{aligned} \|[\mathsf{C}]^2 - [\mathsf{C}^*]^2\|_a &= \tfrac{1}{q^{16}} \max_x \sum_y \max_{\gamma \neq 0} \sum_{\gamma' \neq 0} \left| \text{ELP}^{\mathsf{C}}(\gamma,\gamma') - \tfrac{1}{q^{16}-1} \right| \sum_{y' \neq y} \mathbf{1}_{\text{SUPP}(y \oplus y')=\gamma'} \\ &= \max_{\gamma \neq 0} \sum_{\gamma' \neq 0} \left| \text{ELP}^{\mathsf{C}}(\gamma,\gamma') - \tfrac{1}{q^{16}-1} \right| (q-1)^{w(\gamma')}. \end{aligned}$$

Using similar techniques, one can derive the exact same expression for $|||[\mathsf{C}]^2 - [\mathsf{C}^*]^2|||_\infty$. This implies that, when limited to two queries, an adaptive distinguisher against C is not more powerful than a non-adaptive one. This is not surprising as the first query does not leak any information. A single substitution layer S is enough to have such a result.

Theorem 10. *The respective advantages of the best 2-limited* non-adaptive *distinguisher $\mathcal{A}_{\text{na}}$ and of the best 2-limited* adaptive *distinguisher $\mathcal{A}$ against $r > 1$ rounds of C are such that $\text{Adv}_{\mathcal{A}}(\mathsf{C},\mathsf{C}^*) = \text{Adv}_{\mathcal{A}_{\text{na}}}(\mathsf{C},\mathsf{C}^*)$ and (taking the sum over all non-zero supports)*

$$\text{Adv}_{\mathcal{A}}(\mathsf{C},\mathsf{C}^*) = \tfrac{1}{2} \max_\gamma \sum_{\gamma' \neq 0} \left| \text{ELP}^{\mathsf{C}}(\gamma,\gamma') - \text{ELP}^{\mathsf{C}^*}(\gamma,\gamma') \right| (q-1)^{w(\gamma')}.$$

Table 3. Exact values of the advantage of the best 2-limited adaptive distinguisher for several number of rounds r compared to the bounds given in [5]

r	2	3	4	5	6	7	8	9	10	11	12
Bound	$2^{94.0}$	$2^{72.0}$	$2^{-4.0}$	$2^{-4.0}$	$2^{-24.2}$	$2^{-46.7}$	$2^{-72.4}$	$2^{-95.9}$	$2^{-142.8}$	-	-
Exact	1	$2^{-4.0}$	$2^{-23.4}$	$2^{-45.8}$	$2^{-71.0}$	$2^{-126.3}$	$2^{-141.3}$	$2^{-163.1}$	$2^{-185.5}$	$2^{-210.8}$	$2^{-238.9}$

Practical computations can take into account the fact that $\mathrm{ELP}^{\mathsf{C}}(\gamma, \gamma')$ actually only depends on the 4 weights of the diagonals of γ and on those of the columns of γ' (from Theorem 12 in [5], see Appendix A). Results of our practical computations are reported in Table 3 (together with the corresponding upper bounds obtained in [5]). Finally, we can obtain the following corollary from Theorem 3 and Theorem 10.

Corollary 11. *The advantage of the best* 2*-limited adaptative distinguisher* $\mathcal{A}$ *against* $\mathsf{C}[r]$ *tends towards 0 as* r *increases, i.e.,* $\mathrm{Adv}_{\mathcal{A}}(\mathsf{C}[r], \mathsf{C}^*) \xrightarrow[r\to\infty]{} 0$.

3.4 C Is Resistant to Iterated Attacks of Order 1

Iterated attacks of order 1 [47, 46] are very similar to LC except that the bit of information retrieved from each plaintext/cipher pair does not necessarily have to be derived in a linear way. Such attacks have proven to be sometimes much more powerful than linear cryptanalysis[6]. According to Theorem 18 in [47], proving resistance against $2d$-limited adaptive distinguishers is enough to prove resistance to iterated attacks of order d. We can deduce that C is immune to any iterated attack of order 1.

Property 12 (Provable Security of C against iterated attacks of order 1). *Seven rounds of* C *are sufficient to obtain provable security against iterated attacks of order 1.*

3.5 All Substitution Boxes of C Are Indistinguishable from Independent Perfectly Random Permutations

A pseudo-random bit generator is said to be cryptographically secure if no polynomial-time statistical test can distinguish an output sequence of this generator from a perfectly random bit string with a significant advantage [51]. Such a generator can always be distinguished if the length of the bit string is longer than the generator's period. We need to prove that the Blum-Blum-Shub generator we use has a period long enough to generate a complete extended key.

We know from the original paper [15] that the period of the x_i's sequence of the BBS generator divides $\lambda(\lambda(n))$ (where λ denotes the Carmichael function) if

[6] See for example [4, pg. 9], where an example of a biased source is given. Although impossible to distinguish from a true random source with a linear distinguisher, this source is easily broken by a non-linear distinguisher.

both p and q are strong-primes and both p and q are Blum integers. Obviously, the period of the bit string output by BBS divides the period of the x_i's. By making sure that $\lambda(\lambda(n))$ does not contain small factors, we can prove that this length will be large enough. This can be done by choosing strong-strong-primes p and q. In such a case we can write $p = 2p_1+1 = 4p_2+3$ and $q = 2q_1+1 = 4q_2+3$, and obtain $\lambda(\lambda(n)) = \lambda(\mathrm{lcm}(2\,p_1, 2\,q_1)) = \lambda(2\,p_1\,q_1) = \mathrm{lcm}(2\,p_2, 2\,q_2) = 2\,p_2\,q_2$. Therefore, if the period of the bit string is not 2, it is necessarily long enough to generate a complete extended key as $\min(p_2, q_2) \gg 300\,000$.

It is known that the original Blum-Blum-Shub pseudo-random bit generator is cryptographically secure [16,15]. Vazirani and Vazirani showed that outputting both the least and most significant bits of the quadratic residues produced by the generator is also cryptographically secure [48,49].

Definition 13. *Let s_0 and s_1 be two bit strings, such that s_0 is obtained using the BBS pseudo-random generator and s_1 is perfectly random. The advantage of an adversary $\mathcal{A}$ trying to distinguish s_0 from s_1 is given by*

$$\mathrm{Adv}_{\mathcal{A}}^{\mathsf{BBS}} = \Pr\left[\mathcal{A}(s_0) = 0\right] - \Pr\left[\mathcal{A}(s_1) = 0\right].$$

Assuming that the problem of deciding the quadratic residuosity modulo n is hard (an assumption we will refer to as the *quadratic residuosity assumption* [27]), we know that $\mathrm{Adv}_{\mathcal{A}}^{\mathsf{BBS}}$ can be made arbitrarily small by increasing the value of n. The key schedule of C relies on the BBS generator and makes sure that the mapping from the set of 2^{128} keys to the set of possible seeds of the pseudo-random generator is injective. Therefore, the pseudo-random sequence produced by the key schedule of C is indistinguishable from a perfectly random binary sequence of the same length. The method we use to convert this binary sequence into substitution boxes makes sure that for an unbiased sequence one obtains an unbiased set of substitution boxes. By choosing a suitable n, the substitution boxes of C can thus be made indistinguishable from independent perfectly random permutations.

3.6 The Keyed C Is Not Less Secure Than C

Definition 14. *Let k_0 and k_1 be two extended keys of C, such that k_0 is obtained through the key schedule seeded by a perfectly random 128 bit key and k_1 is perfectly random. The advantage of an adversary $\mathcal{A}$ trying to distinguish k_0 from k_1 is given by*

$$\mathrm{Adv}_{\mathcal{A}}^{\mathsf{key}} = \Pr\left[\mathcal{A}(k_0) = 0\right] - \Pr\left[\mathcal{A}(k_1) = 0\right].$$

Property 15. *Let k_0 and k_1 be two extended keys as in Definition 14 and s_0 and s_1 be two bit strings as in Definition 13. An adversary $\mathcal{A}$ able to distinguish k_0 from k_1 with probability p can distinguish s_0 from s_1 with probability $p' \geq p$, i.e., $\mathrm{Adv}_{\mathcal{A}}^{\mathsf{key}} \leq \mathrm{Adv}_{\mathcal{A}}^{\mathsf{BBS}}$.*

Proof. Given s_b ($b \in \{0, 1\}$), the adversary can derive an acceptable extended key k_b. From this, the adversary has an advantage $\mathrm{Adv}_{\mathcal{A}}^{\mathsf{key}}$ of guessing the correct value of b and thus obtains a distinguisher on BBS with advantage $\mathrm{Adv}_{\mathcal{A}}^{\mathsf{key}}$. $\square$

The strongest notion of security for a block cipher is its indistinguishability from a perfectly random permutation C^*. Proving the security of C against a distinguishing attack performed by $\mathcal{A}$ consists in upper bounding $\mathrm{Adv}_{\mathcal{A}}(\mathsf{C}, \mathsf{C}^*)$.

Let k_0 and k_1 be two random extended keys of C picked as in Definition 14, defining two random instances of C denoted $\mathsf{C}_{\mathsf{key}}$ and $\mathsf{C}_{\mathsf{rand}}$ respectively. Obviously, distinguishing $\mathsf{C}_{\mathsf{key}}$ from $\mathsf{C}_{\mathsf{rand}}$ is harder than distinguishing k_0 from k_1, so that $\mathrm{Adv}_{\mathcal{A}}(\mathsf{C}_{\mathsf{key}}, \mathsf{C}_{\mathsf{rand}}) \leq \mathrm{Adv}_{\mathcal{A}}^{\mathsf{key}}$.

Assume there exists a distinguishing attack on $\mathsf{C}_{\mathsf{key}}$ that does not work on $\mathsf{C}_{\mathsf{rand}}$ such that, for an adversary $\mathcal{A}$ using it, $\mathrm{Adv}_{\mathcal{A}}(\mathsf{C}_{\mathsf{key}}, \mathsf{C}^*) \geq 2 \cdot \mathrm{Adv}_{\mathcal{A}}(\mathsf{C}_{\mathsf{rand}}, \mathsf{C}^*)$. From the triangular inequality we have $\mathrm{Adv}_{\mathcal{A}}(\mathsf{C}_{\mathsf{key}}, \mathsf{C}^*) - \mathrm{Adv}_{\mathcal{A}}(\mathsf{C}_{\mathsf{rand}}, \mathsf{C}^*) \leq \mathrm{Adv}_{\mathcal{A}}(\mathsf{C}_{\mathsf{key}}, \mathsf{C}_{\mathsf{rand}})$ so that $\mathrm{Adv}_{\mathcal{A}}(\mathsf{C}_{\mathsf{key}}, \mathsf{C}^*) \leq 2 \cdot \mathrm{Adv}_{\mathcal{A}}(\mathsf{C}_{\mathsf{key}}, \mathsf{C}_{\mathsf{rand}}) \leq 2 \cdot \mathrm{Adv}_{\mathcal{A}}^{\mathsf{key}}$.

In conclusion, using Property 15, any distinguishing attack twice as efficient on $\mathsf{C}_{\mathsf{key}}$ than on $\mathsf{C}_{\mathsf{rand}}$ gives an advantage which is bounded by $2 \cdot \mathrm{Adv}_{\mathcal{A}}^{\mathsf{BBS}}$. Under the quadratic residuosity assumption, such an attack cannot be efficient.

Although the quadratic residuosity problem is not equivalent to the problem of factoring $p \cdot q$, the best known attacks require it. The exact cost of this factorization is not obvious. For a given symmetric key size, there are several estimates for an equivalent asymmetric key size [31]. According to the NIST recommendations, a 2048 bit modulus is equivalent to a 112 bit symmetric key [24].

Property 16 (Provable security of $\mathsf{C}_{\mathsf{key}}$). *Under the quadratic residuosity assumption,* C *used with the key schedule described in Section 2.3 is as secure as* C *used with independent perfectly random substitution boxes.*

3.7 The Keyed C Has No Equivalent Keys

Two block cipher keys are said to be *equivalent* when they define the same permutation. It is easy to build equivalent *extended* keys for C (when *not* using the key schedule). Consider an extended key k_1 defining a set of 160 substitution boxes such that the first 32 are the identity. We consider a second extended key k_2 defining another set of substitution boxes such that the last 128 are identical to that defined by k_1 and such that the first 16 boxes simply xor a constant $a \in \{0,1\}^{128}$ to the plaintext, the remaining boxes (in the second layer) correcting the influence of a by xoring $\mathsf{L}(a)$ to its input. Although they are different, k_1 and k_2 define the same permutation. Such a property could be a threat to the security of C. If too many such extended keys were equivalent, it could be possible to find equivalent 128 bit keys for $\mathsf{C}_{\mathsf{key}}$. We can prove that the probability that two 128 bit equivalent keys exist is negligible.

The probability that two equivalent 128 bit keys exist depends on the number of equivalence classes among the extended keys. Considering a one round version of C, it can be seen that no equivalent extended keys exist. Consequently, there are at least $(2^8!)^{16} \approx 2^{26944}$ equivalence classes. Adding rounds (thus increasing the extended key size) cannot decrease this number of classes. Assuming that the key schedule based on BBS uniformly distributes the extended keys obtained from the 128 bit keys among these classes, the probability that two keys fall into the same class can be upper bounded by

$$1 - e^{-(2^{128})^2/(2*2^{26944})} \approx 2^{-26689}.$$

Property 17 ($\mathsf{C}_{\mathsf{key}}$ has no Equivalent Keys). *The probability that two 128 bit keys lead to the same instance of* C *is upper bounded by* 2^{-26689}.

4 Security Results: What We Believe to Be True

4.1 C Is (Not That) Resistant to Saturation Attacks

Saturation attacks [20] are chosen-plaintext attacks on byte-oriented ciphers. An attack on four rounds of AES can be performed [22] by choosing a set of 2^8 plaintexts equal on all but one byte. After 3 rounds of AES, the xor of all the corresponding ciphertexts is 0. This makes it easy to guess the key of the fourth round, as all round key bytes can be guessed independently.

In our case, the property on the third round output still holds. Nevertheless, it only allows to exclude 255 out of 256 keys for each substitution box. This was enough for AES, but in our case an adversary would still be left with 255! valid substitution boxes, so that a more subtle approach is needed.

In [13], Biryukov and Shamir present an attack on SASAS, a generic construction with three rounds of random key-dependent substitution boxes linked by random key-dependent affine layers. Following their approach, the saturation attacks on the AES can be adapted to C but with a non-negligible cost. In this approach, an exhaustive search on 8 bits (as necessary with the AES) is replaced by a linear algebra step which requires 2^{24} operations. The additional workload is thus of the order of 2^{16}. This overhead implies that any attack with a complexity higher than 2^{112} becomes infeasible. In particular the saturation attacks on 7 rounds of the AES [23] should not apply to C.

We believe that saturation-like attacks are the biggest threat for reduced rounds versions of C. Chances that such attacks apply to 10 rounds are however very low.

4.2 C Is Resistant to a Wide Variety of Attacks

Algebraic attacks consist in rewriting the whole block cipher as a system of algebraic equations. The solutions of this system correspond to valid plaintext, ciphertext, and key triples. Algebraic attack attempts on AES take advantage of the simple algebraic structure of the substitution box [19]. In our case, substitution boxes can by no means be described by simple algebraic forms, and thus, algebraic attacks will necessarily be much more complex against C than against AES. We do believe that they will be more expensive than exhaustive key search.

Slide attacks [14] exploit a correlation between the different round keys of a cipher. These attacks apply for example against ciphers with weak key schedules or against block ciphers with key-dependent substitution boxes and periodic key schedules. C uses independent perfectly random substitution boxes, so that all rounds are independent from each other. Slide attacks cannot apply here.

The boomerang attack [50] is a special type of differential cryptanalysis. It needs to find a differential characteristic on half the rounds of the cipher. Four rounds of C being sufficient to be provably secure against DC, 10 rounds are necessarily sufficient to resist the boomerang attack. Similarly, neither differential-linear cryptanalysis [33,10] nor the rectangle attack [9] apply to C.

5 Reducing the Extended Key Size

The main drawback in the design of C is the huge amount of pseudo-random bits required for the key schedule. Having to generate hundreds of thousands of bits with the Blum-Blum-Shub generator is unacceptable for many applications. We propose here an adaptation of C, enjoying the same security proofs, but requiring much less pseudo-random bits.

Using Order 2 Decorrelated Substitutions Boxes. As stated in [5], the bounds on the LP and DP obtained when replacing the substitution boxes of the AES by independent perfectly random permutations remain exactly the same if one uses independent order 2 decorrelated substitution boxes instead. This is also the case concerning resistance against 2-limited adaptive distinguishers and, as a consequence, resistance against iterated attacks of order 1.

Suppose we have a family $\mathcal{D}_2$ of order 2 decorrelated substitution boxes. Using the Blum-Blum-Shub generator and the same method as for the standard C key schedule, we can generate a set of 160 substitution boxes from $\mathcal{D}_2$ indistinguishable from 160 randomly chosen $\mathcal{D}_2$ boxes. Again, it is possible to prove that any attack on a keyed C using substitution boxes in $\mathcal{D}_2$ requires to be able to distinguish the output of the Blum-Blum-Shub generator from a perfectly random binary stream.

Hence, apart from the resistance to impossible differentials, all proven security arguments of C remain untouched when using boxes of $\mathcal{D}_2$. However, each time the key schedule required $\log_2 256!$ bits from the Blum-Blum-Shub generator, it only requires $\log_2 |\mathcal{D}_2|$ now.

$\boldsymbol{A \oplus \frac{B}{X}}$**: a Good Family of Order 2 Decorrelated Substitution Boxes.** From what we have just seen, whatever the family $\mathcal{D}_2$ we use, security results will still hold. For optimal efficiency, we need to select the smallest possible such family. It was shown in [3] that any family of the form $\mathcal{D}_2 = \big\{X \mapsto A \oplus B \cdot S(X); A, B \in \{0,1\}^8, B \neq 0\big\}$ where S is any *fixed* permutation of $\mathrm{GF}(2^8)$ (and where $\cdot$ represents a product in $\mathrm{GF}(2^8)$) is decorrelated at order 2.

We propose to use the family $\mathcal{D}_2 = \big\{X \mapsto A \oplus \frac{B}{X}; A, B \in \{0,1\}^8, B \neq 0\big\}$. This family contains 2^{16} elements and the substitution boxes can be chosen uniformly in $\mathcal{D}_2$ from 16 bits of the Blum-Blum-Shub generator. The first 8 bits define A, the last 8 define B. So, the whole key schedule for ten rounds of C only requires 2 560 pseudo-random bits and should be about 100 times faster than an unmodified C with perfectly random permutations. One may believe that this construction is very similar to that of the AES (assuming that the round keys are independent and perfectly random). Nevertheless, deriving the AES construction from ours requires to set $B = 1$. The family obtained in this case is no longer decorrelated at order 2, so that, unfortunately, none of the security results we obtained for C directly applies to the AES.

Security Considerations. Even if this might not be the case for any order 2 decorrelated family of substitution boxes, it is interesting to note that C built on the family $\mathcal{D}_2$ we chose is also resistant to impossible differentials. As for perfectly random permutations, lemmas 5 and 6 can both be proven for boxes of the form $A \oplus \frac{B}{X}$.

None of the security results we obtained requires using perfectly random permutations and substitution boxes of the form $A \oplus \frac{B}{X}$ are enough. We believe that achieving the same security level with perfectly random permutations is possible with fewer rounds. More precisely, it may be possible to obtain a trade-off between the number of rounds and the level of decorrelation of the random substitution boxes. Fewer rounds lead to fast encryption/decryption procedures but require a higher level of decorrelation. In this case, more pseudo-random bits are necessary to generate each substitution box, and this may lead to a (very) slow key schedule. The best choice depends on the application.

6 Implementation and Performances

Implementation. As seen in Section 2.3, before being able to use the Blum-Blum-Shub generator, one needs to generate two strong-strong-primes p and q, which is not an easy operation: it has a complexity of $O((\log p)^6)$. For primes of length 1024, this takes one million times more operations than generating a prime of the same size. Some optimizations exist to improve the constant factor in the prime number generation [29] and can become very useful for strong-strong-prime numbers.

When implementing C, the same optimizations as for AES are possible. In particular, one round of C can be turned into 16 table look-ups and 12 xors. Basically, the output can be split in four 32 bits blocks, each of which only depends on four bytes of the input. However, all the tables of C are different from each other. This is the only reason why encrypting/decrypting with C could be slower than with AES. Considering standard 32-bits computers, this has little influence in practice as the 160 tables still fit in the cache of the CPU. The required memory is $160 \cdot 256 \cdot 4 = 160$kBytes. This however becomes an issue when implementing C on a smartcard (but who wants to implement Blum-Blum-Shub on a smartcard anyway?) or on a CPU with 128 kBytes of cache.

We programmed C in C using GMP [25] for the key schedule operations. On a 3.0 GHz Pentium D, we obtain encryption/decryption speeds of 500 Mbits/s. Generating the 160 substitution boxes from the 128 bit secret key takes 2.5s when using perfectly random permutations and 25ms when using the $A \oplus \frac{B}{X}$ construction. Note that to decrypt, it is also necessary to invert the substitution boxes. This takes a negligible time compared to the generation of the extended key, which is the most expensive step of the key schedule.

Applications. Given the timings we obtained, it appears that using C for encryption purpose is practical, in particular with the shortened key schedule. Of course, a key schedule of 25ms is much slower than most existing key schedules but is still acceptable in a large majority of applications. This can become negligible when the amount of data to encrypt becomes large.

The 2.5s obtained for the "most secure" version using perfectly random substitution boxes is suitable for only a few very specific applications. However, we believe that in the case where a very high security level is required, this price is not that high. This might not be an issue in certain cases when the key schedule is run in parallel with some other slow operation, like for hard disk drive encryption (for which the key schedule is performed only once during a boot sequence which already takes several seconds).

In some other circumstances however, C is not usable at all. For example, when using it as a compression function in a Merkle-Damgård construction, as one key schedule has to be performed for each block (hashing a 1 MByte message would take more than one day).

Further Improvements. It is known that outputting $\alpha(n) = O(\log\log n)$ bits at each iteration of the Blum-Blum-Shub generator is cryptographically secure [49]. However, for a modulus n of given bit length, no explicit range for $\alpha(n)$ was ever given in the literature [40]. Finding such a constant could considerably improve the speed of the key schedule of C.

Another possible improvement to the key schedule would be to rely on some other cryptographically secure pseudo-random generator. The pseudo-random generator on which the stream cipher QUAD [7,6] is based may be a good candidate: it offers provable security results and achieves speeds up to 5.7Mbits/s. Using such a construction would certainly improve the key schedule time by an important factor, so that the "most secure" version of C might compare to the current version using derandomized substitution boxes.

7 Conclusion

We have introduced C, a block cipher provably secure against a wide range of attacks. It is as fast as AES for encryption on a standard workstation. Provable security requires a cryptographically secure key schedule. Consequently, the key schedule of C is too slow for some applications.

As far as we know, C is the first practical block cipher to provide tight security proofs that do take into account the key schedule. It is proven that C resists: linear cryptanalysis (taking into account the possible cumulative effects of a linear hull), differential cryptanalysis (similarly considering cumulative effects of differentials), 2-limited adaptive distinguishers, iterated attacks of order 1, and impossible differentials. We also give strong evidence that it also resists: algebraic attacks, slide attacks, the boomerang attack, the rectangle attack, differential-linear cryptanalysis, and, to some extent, saturation attacks. From our point of view, the most significant improvement that could be made on C would be to give a bound on the advantage of the best d-limited adversary for $d > 2$.

"Mind you, even I didn't think of that one... extraordinary."
Chief Insp. Hubbard

References

1. Adams, C., Heys, H.M., Tavares, S.E., Wiener, M.: CAST256: a submission for the advanced encryption standard. In: First AES Candidate Conference (AES1) (1998)
2. Aoki, K., Ichikawa, T., Kanda, M., Matsui, M., Moriai, S., Nakajima, J., Tokita, T.: Camellia: a 128-bit block cipher suitable for multiple platforms - design and analysis. In: Stinson, D.R., Tavares, S. (eds.) SAC 2000. LNCS, vol. 2012, pp. 39–56. Springer, Heidelberg (2001)
3. Aoki, K., Vaudenay, S.: On the use of GF-inversion as a cryptographic primitive. In: Matsui, M., Zuccherato, R. (eds.) SAC 2003. LNCS, vol. 3006, pp. 234–247. Springer, Heidelberg (2004)
4. Baignères, T., Junod, P., Vaudenay, S.: How far can we go beyond linear cryptanalysis? In: Lee, P.J. (ed.) ASIACRYPT 2004. LNCS, vol. 3329, pp. 432–450. Springer, Heidelberg (2004)
5. Baignères, T., Vaudenay, S.: Proving the security of AES substitution-permutation network. In: Preneel, B., Tavares, S.E. (eds.) SAC 2005. LNCS, vol. 3897, pp. 65–81. Springer, Heidelberg (2006)
6. Berbain, C., Billet, O., Gilbert, H.: Efficient implementations of multivariate quadratic systems. In: Biham, E., Youssef, A.M. (eds.) Selected Areas in Cryptography SAC'06. LNCS, Springer, Heidelberg (to appear)
7. Berbain, C., Gilbert, H., Patarin, J.: QUAD: a practical stream cipher with provable security. In: Vaudenay, S. (ed.) EUROCRYPT 2006. LNCS, vol. 4004, pp. 109–128. Springer, Heidelberg (2006)
8. 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)
9. Biham, E., Dunkelman, O., Keller, N.: The rectangle attack - rectangling the Serpent. In: Pfitzmann, B. (ed.) EUROCRYPT 2001. LNCS, vol. 2045, pp. 340–357. Springer, Heidelberg (2001)
10. Biham, E., Dunkelman, O., Keller, N.: Enhancing differential-linear cryptanalysis. In: Zheng, Y. (ed.) ASIACRYPT 2002. LNCS, vol. 2501, pp. 254–266. Springer, Heidelberg (2002)
11. Biham, E., Shamir, A.: Differential cryptanalysis of DES-like cryptosystems. Journal of Cryptology 4, 3–72 (1991)
12. Biham, E., Shamir, A.: Differential cryptanalysis of DES-like cryptosystems (extended abstract). In: Menezes, A.J., Vanstone, S.A. (eds.) CRYPTO 1990. LNCS, vol. 537, pp. 2–21. Springer, Heidelberg (1991)

13. Biryukov, A., Shamir, A.: Structural cryptanalysis of SASAS. In: Pfitzmann, B. (ed.) EUROCRYPT 2001. LNCS, vol. 2045, pp. 394–405. Springer, Heidelberg (2001)
14. Biryukov, A., Wagner, D.: Slide attacks. In: Knudsen, L.R. (ed.) FSE 1999. LNCS, vol. 1636, pp. 245–259. Springer, Heidelberg (1999)
15. Blum, L., Blum, M., Shub, M.: Comparison of two pseudo-random number generators. In: Chaum, D., Rivest, R.L., Sherman, A. (eds.) Advances in Cryptology - Crypto'82, Plemum, pp. 61–78 (1983)
16. Blum, L., Blum, M., Shub, M.: A simple unpredictable pseudo-random number generator. SIAM Journal on Computing 15(2), 364–383 (1986)
17. Chabaud, F., Vaudenay, S.: Links between differential and linear cryptanalysis. In: De Santis, A. (ed.) EUROCRYPT 1994. LNCS, vol. 950, pp. 356–365. Springer, Heidelberg (1995)
18. Cheon, J.H., Kim, M.J., Kim, K., Lee, J.-Y., Kang, S.W.: Improved impossible differential cryptanalysis of Rijndael and Crypton. In: Kim, K.-c. (ed.) ICISC 2001. LNCS, vol. 2288, pp. 39–49. Springer, Heidelberg (2002)
19. Courtois, N., Pieprzyk, J.: Cryptanalysis of block ciphers with overdefined systems of equations. In: Zheng, Y. (ed.) ASIACRYPT 2002. LNCS, vol. 2501, pp. 267–287. Springer, Heidelberg (2002)
20. Daemen, J., Knudsen, L., Rijmen, V.: The block cipher SQUARE. In: Biham, E. (ed.) FSE 1997. LNCS, vol. 1267, pp. 149–165. Springer, Heidelberg (1997)
21. Daemen, J., Rijmen, V.: AES proposal: Rijndael. NIST AES Proposal (1998)
22. Daemen, J., Rijmen, V.: The Design of Rijndael. In: ISC 2002, Springer, Heidelberg (2002)
23. Ferguson, N., Kelsey, J., Lucks, S., Schneier, B., Stay, M., Wagner, D., Whiting, D.: Improved cryptanalysis of Rijndael. In: Schneier, B. (ed.) FSE 2000. LNCS, vol. 1978, pp. 213–230. Springer, Heidelberg (2001)
24. Gehrmann, C., Näslund, M.: Ecrypt yearly report on algorithms and keysizes, 2005. Technical report, Ecrypt (2006)
25. GMP. GNU Multiple Precision arithmetic library, `http://www.swox.com/gmp`
26. Goldreich, O., Levin, L.A.: A hard-core predicate for all one-way functions. In: Proceedings of the twenty-first annual ACM symposium on Theory of computing - STOC'89, pp. 25–32. ACM Press, New York (1989)
27. Goldwasser, S., Micali, S.: Probabilistic Encryption. Journal of Computer and System Sciences 28(2), 270–299 (1984)
28. Hitchcock, A.: Dial M for Murder (1954), `http://www.imdb.com/title/tt0046912`
29. Joye, M., Paillier, P., Vaudenay, S.: Efficient generation of prime numbers. In: Paar, C., Koç, Ç.K. (eds.) CHES 2000. LNCS, vol. 1965, pp. 340–354. Springer, Heidelberg (2000)
30. Junod, P., Vaudenay, S.: FOX: a new family of block ciphers. In: Handschuh, H., Hasan, A. (eds.) SAC 2004. LNCS, vol. 3357, pp. 114–129. Springer, Heidelberg (2004)
31. Keylength.com., `http://www.keylength.com`
32. Lai, X., Massey, J., Murphy, S.: Markov ciphers and differential cryptanalysis. In: Davies, D.W. (ed.) EUROCRYPT 1991. LNCS, vol. 547, pp. 17–38. Springer, Heidelberg (1991)
33. Langford, S.K., Hellman, M.E.: Differential-linear cryptanalysis. In: Desmedt, Y.G. (ed.) CRYPTO 1994. LNCS, vol. 839, pp. 17–25. Springer, Heidelberg (1994)
34. Lim, C.H.: A revised version of CRYPTON: CRYPTON V1.0. In: Knudsen, L. (ed.) FSE 1999. LNCS, vol. 1636, pp. 31–45. Springer, Heidelberg (1999)

35. Luby, M., Rackoff, C.: How to construct pseudorandom permutations from pseudorandom functions. SIAM Journal on Computing 17(2), 373–386 (1988)
36. Massey, J.: SAFER-K64: a byte-oriented block-ciphering algorithm. In: Anderson, R.J. (ed.) FSE'93. LNCS, vol. 809, pp. 1–17. Springer, Heidelberg (1994)
37. Matsui, M.: The first experimental cryptanalysis of the Data Encryption Standard. In: Desmedt, Y.G. (ed.) CRYPTO 1994. LNCS, vol. 839, pp. 1–11. Springer, Heidelberg (1994)
38. Matsui, M.: Linear cryptanalysis method for DES cipher. In: Helleseth, T. (ed.) EUROCRYPT 1993. LNCS, vol. 765, pp. 386–397. Springer, Heidelberg (1994)
39. Matsui, M.: New structure of block ciphers with provable security against differential and linear cryptanalysis. In: Gollmann, D. (ed.) FSE'96. LNCS, vol. 1039, pp. 205–218. Springer, Heidelberg (1996)
40. Menezes, A., Van Oorschot, P., Vanstone, S.: Handbook of applied cryptography. The CRC Press series on discrete mathematics and its applications. CRC Press (1997)
41. Nyberg, K.: Linear approximation of block ciphers. In: De Santis, A. (ed.) EUROCRYPT 1994. LNCS, vol. 950, pp. 439–444. Springer, Heidelberg (1995)
42. O'Connor, L.: Properties of linear approximation tables. In: Preneel, B. (ed.) Fast Software Encryption. LNCS, vol. 1008, pp. 131–136. Springer, Heidelberg (1995)
43. Shannon, C.E.: Communication theory of secrecy systems. Bell Systems Technical Journal 28(4), 656–715 (1993), Re-edited in Claude Elwood Shannon - Collected Papers. IEEE Press, New York (1993)
44. Stern, J., Vaudenay, S.: CS-Cipher. In: Vaudenay, S. (ed.) FSE 1998. LNCS, vol. 1372, pp. 189–204. Springer, Heidelberg (1998)
45. Tardy-Corfdir, A., Gilbert, H.: A known plaintext attack of FEAL-4 and FEAL-6. In: Feigenbaum, J. (ed.) CRYPTO 1991. LNCS, vol. 576, pp. 172–182. Springer, Heidelberg (1992)
46. Vaudenay, S.: Resistance against general iterated attacks. In: Stern, J. (ed.) EUROCRYPT 1999. LNCS, vol. 1592, pp. 255–271. Springer, Heidelberg (1999)
47. Vaudenay, S.: Decorrelation: a theory for block cipher security. Journal of Cryptology 16(4), 249–286 (2003)
48. Vazirani, U., Vazirani, V.: Efficient and secure pseudo-random number generation. In: Blakely, G., Chaum, D. (eds.) CRYPTO 1984. LNCS, vol. 196, pp. 193–202. Springer, Heidelberg (1985)
49. Vazirani, U., Vazirani, V.: Efficient and secure pseudo-random number generation (extended abstract). In: Proceedings of FOCS'84, pp. 458–463. IEEE, Los Alamitos (1985)
50. Wagner, D.: The boomerang attack. In: Knudsen, L. (ed.) FSE 1999. LNCS, vol. 1636, pp. 156–170. Springer, Heidelberg (1999)
51. Yao, A.C.: Theory and applications of trapdoor functions (extended abstract). In: Proceedings of FOCS'82, pp. 80–91 (1982)
52. Youssef, A.M., Tavares, S.E.: Resistance of balanced S-boxes to linear and differential cryptanalysis. Information Processing Letters 56, 249–252 (1995)

A Further Reducing the Matrix Size

From Theorem 12 in [5], we know that $\mathrm{ELP}^{\mathsf{C}}(\gamma, \gamma')$ actually only depends on the weights of the diagonals of γ and of the columns of γ'. Respectively denoting

$\nu = (\nu_0, \nu_1, \nu_2, \nu_3)$ and $\mu = (\mu_0, \mu_1, \mu_2, \mu_3)$ those two sets of 4 weights, we obtain from Theorem 10 that

$$\begin{aligned} 2\,\mathrm{Adv}_{\mathcal{A}} &= \max_{\gamma} \sum_{\gamma' \neq 0} \left| \mathrm{ELP}^{\mathsf{C}}(\gamma, \gamma') - \mathrm{ELP}^{\mathsf{C}^*}(\gamma, \gamma') \right| (q-1)^{w(\gamma')} \\ &= \max_{\nu} \sum_{\mu \neq 0} \left| \mathrm{ELP}^{\mathsf{C}}(\nu, \mu) - \mathrm{ELP}^{\mathsf{C}^*}(\nu, \mu) \right| (q-1)^{w(\mu)} B[\mu], \end{aligned}$$

where $B[\mu] = \binom{4}{\mu_0}\binom{4}{\mu_1}\binom{4}{\mu_2}\binom{4}{\mu_3}$ denotes the number of distinct supports having a column weight pattern equal to μ. Consequently, the final computation can be reduced to computations on 625×625 matrices.

Improved Security Analysis of XEX and LRW Modes

Kazuhiko Minematsu

NEC Corporation, 1753 Shimonumabe, Nakahara-Ku, Kawasaki 211-8666, Japan
k-minematsu@ah.jp.nec.com

Abstract. We study block cipher modes that turn a block cipher into a tweakable block cipher, which accepts an auxiliary variable called tweak in addition to the key and message. Liskov et al. first showed such a mode using two keys, where one is the block cipher's key and the other is used for some non-cryptographic function. Later, Rogaway proposed the XEX mode to reduce these two keys to one key. In this paper, we propose a generalization of the Liskov et al.'s scheme with a concrete security proof. Using this, we provide an improved security proof of the XEX and some improvements to the LRW-AES, which is a straightforward AES-based instantiation of Liskov et al.'s scheme proposed by the IEEE Security in Storage Workgroup.

1 Introduction

Tweakable block ciphers are block ciphers that accept a variable called tweak in addition to the key and message. They were formally defined by Liskov, Rivest, and Wagner [10]. In their definition, a tweak is used to provide variability: any two different tweaks give two instances of an ordinary (i.e., not tweakable) block cipher. Formally, tweakable block ciphers are defined as a function $\widetilde{E} : \mathcal{M} \times \mathcal{K} \times \mathcal{T} \to \mathcal{M}$, where $(\mathcal{M}, \mathcal{K}, \mathcal{T})$ denotes (message space, key space, tweak space). For any two tweak values, $T \neq T'$, the outputs of $\widetilde{E}_{K,T}$ should appear to be independent of outputs of $\widetilde{E}_{K,T'}$ even if T and T' are public but K is secret. Liskov et al. showed that a standard block cipher could be easily converted into a tweakable one by using a mode of operation similar to DESX [9]. They also pointed out that tweakable block ciphers are key components to build advanced modes such as authenticated encryption modes. Their proposal, which we call the LRW mode, is as follows. For plaintext M with tweak T, the ciphertext is $C = E_K(M \oplus \Delta(T)) \oplus \Delta(T)$, where Δ is a keyed function of T called the offset function. They proved that the LRW mode was provably secure if the key of Δ, denoted by K_Δ, was independent of K, and Δ was ϵ-almost XOR universal (ϵ-AXU) for sufficiently small ϵ (see Def. 2). The security considered here is the indistinguishability from the ideal tweakable block cipher using any combination of chosen-plaintext attack (CPA) and chosen-ciphertext attack (CCA) for chosen tweaks. A tweakable block cipher with this property is called a strong tweakable block cipher.

© Springer-Verlag Berlin Heidelberg 2007

The LRW mode needs two independent keys. However, what would happen if K_Δ is *not* independent of the block cipher key, K? For example, is it safe to use E_K(constant) as (a part of) K_Δ? In this paper, we study this problem. Our main contribution is a general construction of strong tweakable block ciphers with a concrete security proof. Our scheme has basically the same structure as that of the LRW mode, but we allow Δ to invoke E_K and/or another function which is possibly independently keyed of K. Using our scheme, we provide some improvements to the previous modes. The first target is the XEX mode [19], an one-key tweakable block cipher similar to the LRW mode. Here, 'one-key' means that the key of the mode is the block cipher key, and only one block cipher key scheduling is needed. XEX mode has a parameter, and in the initial definition of XEX, it was claimed that it was strongly secure if its parameter provided "unique representations" [18]. However, providing unique representations is not a sufficient condition. XEX having a bad parameter is vulnerable to a very simple attack even if it provides unique representations (see [19] and Sect. 4.1 of this paper). The published version of XEX fixed this problem [19]. Our generalized construction clearly explains why this fix works well (which is only briefly mentioned in [19]) and provides a security proof of the fixed XEX, which improves the one shown in [19].

Our second target is the LRW-AES, which is a straightforward instantiation of LRW mode using AES [23]. It has been discussed by the IEEE security in storage working group (SISWG) as a standard mode for storage encryption. The offset function of LRW-AES uses multiplication in GF(2^{128}), where a tweak is multiplied by the 128-bit key independent of the block cipher's key. Using our scheme, we demonstrate how to reduce two keys of the LRW-AES to one key without increasing the computational cost or reducing the allowed tweak set. The underlying idea is similar (but not identical) to one applied to XEX. We also present an alternative mode of AES using the 4-round AES as the offset function. That is, the mode is essentially AES-based and no dedicated AXU function is needed. XEX mode of AES has this property too; however, it allows only incremental update of a tweak. In contrast, our proposal enables us to update a tweak arbitrarily at the cost of 4-round AES invocation. We provide an experimental implementation of our AES-based mode and demonstrate that ours is much more efficient than the reference LRW-AES implementation.

2 Preliminaries

2.1 Notation

Σ^n denotes $\{0,1\}^n$. If a random variable X is uniformly distributed over a set $\mathcal{X}$, we write $X \in_{\mathrm{u}} \mathcal{X}$. An n-bit block uniform random function (URF), denoted by R, is a random variable uniformly distributed over $\{f : \Sigma^n \rightarrow \Sigma^n\}$. Similarly, a random variable distributed over all n-bit permutations is an n-bit block uniform random permutation (URP) and is denoted by P. A tweakable n-bit URP with the tweak space $\mathcal{T}$ is defined by the set of $|\mathcal{T}|$ independent URPs (i.e., an independent n-bit URP is used for each tweak in $\mathcal{T}$) and is denoted

by $\widetilde{\mathsf{P}}$. If $F_K : \mathcal{X} \to \mathcal{Y}$ is a keyed function, then F_K is a random variable (not necessarily uniformly) distributed over $\{f : \mathcal{X} \to \mathcal{Y}\}$. If its key, K, is uniform over $\mathcal{K}$, we have $\Pr(F_K(x) = y) = \{k \in \mathcal{K} : f(k, x) = y\}/|\mathcal{K}|$ for some function $f : \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$. If K is fixed to $k \in \mathcal{K}$, F_k denotes a function $f(k, *)$. If K is clear from the context, we will omit the subscript K and write $F : \mathcal{X} \to \mathcal{Y}$.
Elements of GF(2^n). We express the elements of field GF(2^n) by the n-bit coefficient vectors of the polynomials in the field. We alternatively represent n-bit coefficient vectors by integers $0, 1, \ldots, 2^n - 1$. For example, 5 corresponds to the coefficient vector $(00\ldots0101)$ (which corresponds to the polynomial $\mathtt{x}^2 + 1$) and 1 corresponds to $(00\ldots01)$, i.e., the identity element.

2.2 Security Notion

Definition 1. *Let F and G be two keyed n-bit block functions. Let us assume that the oracle has implemented H, which is identical to one of F or G. An adversary, A, guesses if H is F or G using CPA. The maximum CPA-advantage in distinguishing F from G is defined as*

$$\mathtt{Adv}^{\mathtt{cpa}}_{F,G}(q, \tau) \overset{\text{def}}{=} \max_{A:(q,\tau)\text{-CPA}} \left| \Pr(A^F = 1) - \Pr(A^G = 1) \right|, \tag{1}$$

where $A^F = 1$ denotes that A's guess is 1, which indicates one of F or G, and (q, τ)-CPA denotes a CPA that uses q queries with time complexity τ (see [1] for a detailed description of τ). If the attacks have unlimited computational power, we write $\mathtt{Adv}^{\mathtt{cpa}}_{F,G}(q)$.

Let E_K be an n-bit block cipher and let $\widetilde{E}_K$ be an n-bit tweakable block cipher with tweak space $\mathcal{T}$. For any $x \in \Sigma^n$, $t \in \mathcal{T}$, and $w \in \{0, 1\}$, we define:

$$E^{\pm}_K(x, w) \overset{\text{def}}{=} \begin{cases} E_K(x) & \text{if } w = 0 \\ E^{-1}_K(x) & \text{if } w = 1, \end{cases} \qquad \widetilde{E}^{\pm}_K(x, t, w) \overset{\text{def}}{=} \begin{cases} \widetilde{E}_K(x, t) & \text{if } w = 0 \\ \widetilde{E}^{-1}_K(x, t) & \text{if } w = 1, \end{cases}$$

where E^{-1}_K denotes the inversion of E_K. The securities of E_K and $\widetilde{E}_K$ are measured by

$$\mathtt{Adv}^{\mathtt{sprp}}_{E_K}(q, \tau) \overset{\text{def}}{=} \mathtt{Adv}^{\mathtt{cca}}_{E_K,\mathsf{P}}(q, \tau) \overset{\text{def}}{=} \mathtt{Adv}^{\mathtt{cpa}}_{E^{\pm}_K,\mathsf{P}^{\pm}}(q, \tau), \quad \text{and} \tag{2}$$

$$\mathtt{Adv}^{\widetilde{\mathtt{sprp}}}_{\widetilde{E}_K}(q, \tau) \overset{\text{def}}{=} \mathtt{Adv}^{\widetilde{\mathtt{cca}}}_{\widetilde{E}_K,\widetilde{\mathsf{P}}}(q, \tau) \overset{\text{def}}{=} \mathtt{Adv}^{\mathtt{cpa}}_{\widetilde{E}^{\pm}_K,\widetilde{\mathsf{P}}^{\pm}}(q, \tau), \tag{3}$$

where P ($\widetilde{\mathsf{P}}$) is an n-bit URP (tweakable URP with tweak space $\mathcal{T}$). A keyed permutation that can not be efficiently distinguished from URP (i.e., CPA-advantage is negligibly small for any practical (q, τ)) is called a pseudorandom permutation (PRP) [3]. A PRP with a negligibly small Chosen ciphertext attack (CCA)-advantage (i.e., $\mathtt{Adv}^{\mathtt{sprp}}_{E_K}(q, \tau)$) is a strong PRP (SPRP). We focus on modes that turn an n-bit SPRP into an n-bit strong tweakable block cipher, which has negligibly small $\widetilde{\text{CCA}}$-advantage, i.e., $\mathtt{Adv}^{\widetilde{\mathtt{sprp}}}_{\widetilde{E}_K}(q, \tau)$, for any practical (q, τ).

3 Previous Tweakable Block Cipher Modes

3.1 General DESX-Like Mode

All modes dealt within this paper including our proposals have the form defined as

$$\widetilde{E}_{K,K_\Delta}(T,M) = E_K(M \oplus \Delta(T)) \oplus \Delta(T), \tag{4}$$

where $T \in \mathcal{T}$ is a tweak and $M \in \Sigma^n$ is a plaintext. Here, E_K is n-bit block and $\Delta : \mathcal{T} \to \Sigma^n$ is a keyed function of tweak and called an *offset function*. Its key is denoted by K_Δ. Δ can invoke E_K and/or another function which is fixed or independently keyed of K. Thus K_Δ is not always independent of K.

3.2 LRW Mode

Liskov et al.'s scheme, which we call the LRW mode, uses the offset function Δ, where its key K_Δ is independent of the block cipher key, K. To prove its security, we need the notion of an ϵ-almost XOR universal hash function, which is as follows.

Definition 2. *Let $F_K : \mathcal{X} \to \Sigma^n$ be a keyed function with $K \in_{\mathrm{u}} \mathcal{K}$. If $\Pr(F_K(x) \oplus F_K(x') = c) \le \epsilon$ for any $x \ne x'$ and $c \in \Sigma^n$, then F_K is an ϵ-almost XOR universal (ϵ-AXU) hash function.*

The following theorem proves the security of LRW mode. The proof is in Appendix B.

Theorem 1. *Let $\widetilde{E}_{K,K_\Delta}$ be the LRW mode using the offset function Δ, where its key is $K_\Delta \in_{\mathrm{u}} \mathcal{K}_\Delta$ and tweak $T \in \mathcal{T}$ (see Eq. (4)). If Δ is ϵ-AXU (for input T) and K_Δ is independent of the block cipher key K, then $\mathtt{Adv}^{\widetilde{\mathtt{sprp}}}_{\widetilde{E}_{K,K_\Delta}}(q,\tau) \le \mathtt{Adv}^{\mathtt{sprp}}_{E_K}(q,\tau') + \epsilon q^2$, where $\tau' = \tau + O(q)$.*

This is better than the result of Liskov et al. (theorem 2 of [10]), as they showed $3\epsilon q^2$ instead of ϵq^2. A straightforward instantiation of the LRW is to define $\Delta(T) = K_\Delta \cdot T$, where $K_\Delta \in_{\mathrm{u}} \Sigma^n$ and $T \in \Sigma^n$, and $\cdot$ denotes multiplication in $\mathrm{GF}(2^n)$. This apparently has bias $\epsilon = 1/2^n$. The mode of AES with this offset function has been considered by the IEEE SISWG under the name LRW-AES.

3.3 XEX Mode

XEX mode was proposed by Rogaway [19]. It was designed to be a strong tweakable block cipher. According to the definition of XEX, a base is an element of $\Sigma^n \setminus \{0\}$, and a set $\mathbb{I}_1^d \overset{\mathrm{def}}{=} \mathbb{I}_1 \times \mathbb{I}_2 \times \cdots \times \mathbb{I}_d$ is called an index set, where $\mathbb{I}_i \subseteq \{0,1,\ldots,2^n-1\}$ for all i. A pair of a list of bases $\alpha_1,\ldots,\alpha_d$ and an index set $\mathbb{I}_1^d$ is a parameter setting of XEX.

A XEX mode with a parameter setting $((\alpha_1,\ldots,\alpha_d),\mathbb{I}_1^d)$ has the tweak space $\mathbb{I}_1^d \times \Sigma^n$. Let $(i_1,\ldots,i_d,N) \in \mathbb{I}_1^d \times \Sigma^n$. The offset function of XEX is defined as:

$$\Delta(i_1,\ldots,i_d,N) = \alpha_1^{i_1} \cdot \alpha_2^{i_2} \cdot \cdots \cdot \alpha_d^{i_d} \cdot V, \text{ where } V = E_K(N). \tag{5}$$

Here, multiplications are done in GF(2^n). Since Δ uses E_K as the source of randomness, XEX mode is one-key and needs only one block cipher key scheduling. XEX mode is highly efficient: if we want to increment a tweak (i.e., increment one of i_j w/o changing other indexes), then it is done with one bitshift and one XOR operation. This technique is called the powering-up construction and has been adopted by other modes [5,6]. Consequently, XEX mode requires no special functions other than the block cipher. Although we can not change a tweak arbitrarily, we can still increment a tweak (with respect to one of i_j) with negligibly small cost.

4 Construction of Strong Tweakable Block Cipher

4.1 A Bug in the Initial XEX and an Attack Against OCB1

A parameter setting of the XEX is said to provide unique representations if it contains no collisions, i.e., $\prod_{j=1}^{d} \alpha_j^{i_j} \neq \prod_{j=1}^{d} \alpha_j^{i'_j}$ for any $(i_1, \ldots, i_d), (i'_1, \ldots, i'_d)$ such that $(i_1, \ldots, i_d) \neq (i'_1, \ldots, i'_d)$. The following example is a parameter setting providing unique representations shown by Rogaway [18].

Example 1. $\alpha_1 = 2, \alpha_2 = 3$ and $\mathbb{I}_1 = \{0, 1, \ldots, 2^{n/2}\}, \mathbb{I}_2 = \{0, 1\}$.

In the initial definition of XEX [18], it was claimed that XEX was a strong tweakable block cipher if its parameter setting provided unique representations. However, this claim turned out to be false, as pointed out by [19]. In general, XEX is broken if its parameter setting allows an index vector, $(i_1, \ldots, i_d)$, such that $\alpha_1^{i_1} \cdot \cdots \cdot \alpha_d^{i_d} = 1$. We call it a "reduced-to-1" index vector. For example, the parameter setting described in Ex. 1 allows the following attack [19].

1. Ask the oracle to decrypt $C_1 = 0$ with tweak $T_1 = (0, 0, N)$ for some N, and obtain a plaintext $M_1 = E_K^{-1}(E_K(N)) \oplus E_K(N) = N \oplus E_K(N)$. Compute $E_K(N) = M_1 \oplus N$.
2. Then, the encryption of $M_2 = 2 \cdot (M_1 \oplus N) \oplus N$ with tweak $T_2 = (1, 0, N)$, which is denoted by C_2, is predictable from $E_K(N)$: $C_2 = E_K(N) \oplus 2 \cdot E_K(N)$.

On The Security of OCB1. The above attack can be used as an attack against OCB1 [18,19], which is an improvement to the famous OCB mode proposed by Rogaway [17]. He proved that (a generalized form of) OCB1 could use any tweakable block cipher as its component, and that it was a secure AE mode if the underlying tweakable block cipher was strong, i.e., $\widetilde{\text{CCA}}$-secure. It would be natural to wonder if one can attack against OCB1 using the XEX with a bad parameter setting (i.e., one containing a "reduced-to-1" index vector). We show this holds true[1], if the *inverse* of XEX, denoted by XEX^{-1}, is used to instantiate OCB1. For instance, let us use the parameter setting of Ex. 1. Then, XEX^{-1} gives the ciphertext $C = E_K^{-1}(M \oplus \Delta(i_1, i_2, N)) \oplus \Delta(i_1, i_2, N)$ where

[1] The OCB1 defined in [18] and [19] are slightly different, however, our attack can be applied to both versions.

$\Delta(i_1, i_2, N) = 2^{i_1}3^{i_2}E_K(N)$. Although this implementation was not mentioned in [19], it was as efficient as the XEX-based one. Moreover, using XEX^{-1} would be preferable to using XEX in some situations. For example, it would be desirable to use XEX^{-1} if E_K is faster[2] than E_K^{-1} and fast operation of the receiver (rather than the sender) is required. Our attack is presented in Appendix C.

4.2 The Security of Fixed XEX

The attack presented in the previous section crucially depends on the existence of reduced-to-1 index vector. Thus it would be natural to think of the idea of removing reduced-to-1 index vector from the allowed tweak set. Here, we prove that this simple fix is theoretically fine.

Theorem 2. *Let* XEX$[E_K]$ *be the XEX mode of* E_K *with a parameter setting providing unique representations and containing no "reduced-to-*1*" index vector. Then, we have* $\mathtt{Adv}^{\widetilde{\mathtt{sprp}}}_{\mathrm{XEX}[E_K]}(q, \tau) \le \mathtt{Adv}^{\mathtt{sprp}}_{E_K}(2q, \tau') + \frac{4.5q^2}{2^n}$, where $\tau' = \tau + O(q)$.

For example, we can fix the parameter setting of Ex. 1 by removing $(i_1, i_2) = (0, 0)$. If $n = 128$, the fixed XEX is secure if $q \ll 2^{63}$. The same fix has already been proposed in [19]. However, our proof improves the one shown in [19], which proved $\frac{9.5q^2}{2^n}$ instead of $\frac{4.5q^2}{2^n}$. The proof of Theorem 2 will be provided in Sect. 4.3. One of our purposes is to provide a clear and comprehensive explanation why this fix works well.

4.3 The Proof of Theorem 2

Tools for the Proof. Since we will uses a methodology developed by Maurer [11], we briefly describe his notations. Consider event a_i defined for i input/output pairs (and possibly internal variables) of a keyed function, F. Here, we omit the description of key throughout. Let $\overline{a_i}$ be the negation of a_i. We assume a_i is monotone; i.e., a_i never occurs if $\overline{a_{i-1}}$ occurs. For instance, a_i is monotone if it indicates that all i outputs are distinct. An infinite sequence of monotone events $\mathcal{A} = a_0a_1\dots$ is called a *monotone event sequence* (MES). Here, a_0 denotes some tautological event. Note that $\mathcal{A} \wedge \mathcal{B} = (a_0 \wedge b_0)(a_1 \wedge b_1)\dots$ is a MES if $\mathcal{A} = a_0a_1\dots$ and $\mathcal{B} = b_0b_1\dots$ are both MESs. For any sequence of random variables, $X_1, X_2, \dots,$ let X^i denote $(X_1, \dots, X_i)$. After this, dist(X^i) will denote an event where $X_1, X_2, \dots, X_i$ are distinct. If dist(X^i, Y^j) holds true, then we have no collision among $\{X_1, \dots, X_i, Y_1, \dots, Y_j\}$.

Let MESs $\mathcal{A}$ and $\mathcal{B}$ be defined for two keyed functions, $F : \mathcal{X} \to \mathcal{Y}$ and $G : \mathcal{X} \to \mathcal{Y}$, respectively. Let $X_i \in \mathcal{X}$ and $Y_i \in \mathcal{Y}$ be the i-th input and output. Let P^F be the probability space defined by F. For example, $P^F_{Y_i|X^iY^{i-1}}(y^i, x^i)$ means $\Pr[Y_i = y_i|X^i = x^i, Y^{i-1} = y^{i-1}]$ where $Y_j = F(X_j)$ for $j \ge 1$. If $P^F_{Y_i|X^iY^{i-1}}(y^i, x^i) = P^G_{Y_i|X^iY^{i-1}}(y^i, x^i)$ for all possible (y^i, x^i), then we write

[2] For instance, some AES software implementations, including the reference code [22], have this property.

$P^F_{Y_i|X^iY^{i-1}} = P^G_{Y_i|X^iY^{i-1}}$. Inequalities such as $P^F_{Y_i|X^iY^{i-1}} \leq P^G_{Y_i|X^iY^{i-1}}$ are similarly defined.

Definition 3. *We write $F^{\mathcal{A}} \equiv G^{\mathcal{B}}$ if $P^F_{Y_ia_i|X^iY^{i-1}a_{i-1}} = P^G_{Y_ib_i|X^iY^{i-1}b_{i-1}}$ holds for all $i \geq 1$, which means $P^F_{Y_ia_i|X^iY^{i-1}a_{i-1}}(y^i, x^i) = P^G_{Y_ib_i|X^iY^{i-1}b_{i-1}}(y^i, x^i)$ holds for all possible (y^i, x^i) such that both $P^F_{a_{i-1}|X^{i-1}Y^{i-1}}(y^{i-1}, x^{i-1})$ and $P^G_{b_{i-1}|X^{i-1}Y^{i-1}}(y^{i-1}, x^{i-1})$ are positive.*

Definition 4. *We write $F|\mathcal{A} \equiv G|\mathcal{B}$ if $P^F_{Y_i|X^iY^{i-1}a_i}(y^i, x^i) = P^G_{Y_i|X^iY^{i-1}b_i}(y^i, x^i)$ holds for all possible (y^i, x^i) and all $i \geq 1$.*

Note that if $F^{\mathcal{A}} \equiv G^{\mathcal{B}}$, then $F|\mathcal{A} \equiv G|\mathcal{B}$ (but not vice versa).

Definition 5. *For MES $\mathcal{A}$ defined for F, $\nu(F, \overline{a_q})$ denotes the maximal probability of $\overline{a_q}$ for any (q, ∞)-CPA that interacts with F.*

Note that, for any tweakable block cipher $\widetilde{E}_K$, $\nu(\widetilde{E}^{\pm}_K, \overline{a_q})$ is the maximal probability of $\overline{a_q}$ for any $\widetilde{\text{CCA}}$-attacker, i.e., CPA/CCA for chosen tweaks. For simplicity, it will be abbreviated to $\nu(\widetilde{E}_K, \overline{a_q})$. These equivalences are crucial to the information-theoretic security proof. For example, the following theorem holds true.

Theorem 3. *(Theorem 1 (i) of [11]) Let $\mathcal{A}$ and $\mathcal{B}$ be MESs defined for F and G. If $F^{\mathcal{A}} \equiv G^{\mathcal{B}}$ or $F|\mathcal{A} \equiv G$, then $\mathtt{Adv}^{\mathtt{cpa}}_{F,G}(q) \leq \nu(F, \overline{a_q})$.*

We will use some of Maurer's results including Theorem 3 to make simple and intuitive proofs[3]. For completeness, these results are cited in Appendix A.

General Scheme and Its Security Proof. We proceed as follows. First, we describe a general scheme (which has the form of Eq. (4)) for a tweakable block cipher. Then, we prove that it is a strong tweakable block cipher if its offset function satisfies certain conditions. As the fixed XEX satisfies these conditions, we immediately obtain Theorem 2 as a corollary.

For any two keyed n-bit block functions E_K and $G_{K'}$, let TW$[E_K, G_{K'}]$ be an n-bit block tweakable block cipher with tweak space $\mathcal{T} = (\mathcal{L}, \Sigma^n)$ for some finite set $\mathcal{L}$. Here E_K must be invertible. Its offset function is defined as

$$\Delta(T) = (F_{K''}(L, G_{K'}(N))), \text{ where } T = (L, N) \in \mathcal{L} \times \Sigma^n. \tag{6}$$

Here, $F_{K''}$ is a keyed function : $\mathcal{L} \times \Sigma^n \to \Sigma^n$ with key $K'' \in_{\text{u}} \mathcal{K}''$ (see Fig. 1). The key of the offset function is (K', K''). We assume that K and K' are not necessarily independent (e.g., $G_{K'} = E_K$ is possible). We also assume that K'' is independent of (K, K') or a *constant* k'' (i.e., $F_{K''}$ can be a fixed function $F_{k''}$). The ranges of keys can be different. What we want to do is to clarify the

[3] Maurer's methodology [11] can only be applied to information-theoretic settings. In most cases information-theoretic proofs can be easily converted into computational ones, but this is not always the case [12,16]. However, we do not encounter such difficulties in this paper. His methodology can also be applied to *random systems*, i.e., stateful probabilistic functions. However, none of our proposals require underlying functions to be stateful.

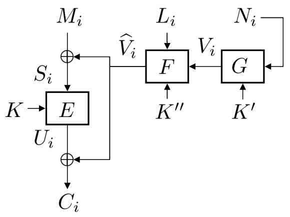


Fig. 1. General scheme for a tweakable block cipher

sufficient condition for $F_{K''}$ to make $\mathrm{TW}[E_K, E_K]$ provably secure. As a first step, we have

$$\mathtt{Adv}^{\widetilde{\mathtt{sprp}}}_{\mathrm{TW}[\mathsf{P},\mathsf{P}]}(q) = \mathtt{Adv}^{\widetilde{\mathtt{cca}}}_{\mathrm{TW}[\mathsf{P},\mathsf{P}],\widetilde{\mathsf{P}}}(q) \le \mathtt{Adv}^{\widetilde{\mathtt{cca}}}_{\mathrm{TW}[\mathsf{P},\mathsf{P}],\mathrm{TW}[\mathsf{P},\mathsf{R}]}(q) + \mathtt{Adv}^{\widetilde{\mathtt{sprp}}}_{\mathrm{TW}[\mathsf{P},\mathsf{R}]}(q), \quad (7)$$

which follows from the triangle inequality. Here, P and R are the n-bit URP and URF, and $\widetilde{\mathsf{P}}$ is the n-bit tweakable URP with tweak space $\mathcal{T}$. Note that P and R in $\mathrm{TW}[\mathsf{P},\mathsf{R}]$ are independent, however, two Ps in $\mathrm{TW}[\mathsf{P},\mathsf{P}]$ denote the same function. We start by analyzing $\mathtt{Adv}^{\widetilde{\mathtt{cca}}}_{\mathrm{TW}[\mathsf{P},\mathsf{P}],\mathrm{TW}[\mathsf{P},\mathsf{R}]}(q)$ in Eq. (7), which is the main technical part. We need some definitions before the analysis. For any $\mathrm{TW}[E_K, G_{K'}]$, let M_i (C_i) denote the i-th plaintext (ciphertext). In addition, let $T_i = (L_i, N_i)$ be the i-th tweak. We define internal variables of $\mathrm{TW}[E_K, G_{K'}]$ such as $V_i \stackrel{\text{def}}{=} G_{K'}(N_i)$ and $\widehat{V}_i \stackrel{\text{def}}{=} F_{K''}(L_i, V_i)$. Moreover, we have $S_i \stackrel{\text{def}}{=} M_i \oplus \widehat{V}_i$ and $U_i \stackrel{\text{def}}{=} C_i \oplus \widehat{V}_i$.

The following lemma tells us what probability we have to analyze.

Lemma 1. *Let a_q be* $\mathrm{dist}(S^q, \mathrm{uni}(N^q))$, *where* $\mathrm{uni}(N^q)$ *consists of all distinct elements among N^q. I.e., a_q means that all elements in $\{S_1, \dots, S_q, N_1, \dots, N_q\}$ are distinct except for the collisions between N_is. Similarly, let b_q denote* $\mathrm{dist}(U^q, \mathrm{uni}(V^q))$. *Here, if* $\mathrm{uni}(N^q) = (N_{i_1}, \dots, N_{i_\theta})$ *for some $\{i_1, \dots, i_\theta\} \subseteq \{1, \dots, q\}$, then* $\mathrm{uni}(V^q) = (V_{i_1}, \dots, V_{i_\theta})$. *Then, we have*

$$\mathtt{Adv}^{\widetilde{\mathtt{cca}}}_{\mathrm{TW}[\mathsf{P},\mathsf{P}],\mathrm{TW}[\mathsf{P},\mathsf{R}]}(q) \le \nu(\mathrm{TW}[\mathsf{P},\mathsf{R}], \overline{a_q \wedge b_q}). \quad (8)$$

Proof. Let us consider the following probabilistic functions: $\Sigma^n \times \{0,1,2\} \to \Sigma^n$.

$$\mathrm{PP}(x,w) = \begin{cases} \mathsf{P}(x) & \text{if } w = 0 \text{ or } 2, \\ \mathsf{P}^{-1}(x) & \text{if } w = 1, \end{cases} \qquad \mathrm{PR}(x,w) = \begin{cases} \mathsf{P}(x) & \text{if } w = 0, \\ \mathsf{P}^{-1}(x) & \text{if } w = 1, \\ \mathsf{R}(x) & \text{if } w = 2. \end{cases}$$

Here, P and R are independent n-bit URP and URF. Observe that there exists a procedure, $\mathbb{F}$, such that $\mathrm{TW}[\mathsf{P},\mathsf{P}]$ ($\mathrm{TW}[\mathsf{P},\mathsf{R}]$) is equivalent to $\mathbb{F}[\mathrm{PP}]$ ($\mathbb{F}[\mathrm{PR}]$). Consider the game of distinguishing PP from PR using CPA (note that this game is quite easy to win). For PP and PR, let $(X_i, W_i) \in \Sigma^n \times \{0,1,2\}$ be the i-th query, and $Y_i \in \Sigma^n$ be the i-th output. For convenience, we allow adversaries to make colliding queries having $W_i = 2$ such as $(X_1, 2)$ and $(X_2, 2)$ where $X_1 = X_2$. Let $\mathcal{I} = \{i \in \{1, \dots, q\} : W_i \in \{0,2\}\}$. Let a'_q be the event that all

X_is with $i \in \mathcal{I}$ are distinct, except for the trivial collisions (i.e., $X_i = X_j$ such that $W_i = W_j = 2$ and $i \neq j$). Similarly, b'_q denotes the event that all Y_is with $i \in \mathcal{I}$ are distinct, except for the trivial collisions. Note that a'_q is equivalent to b'_q in PP, but not in PR. Then, for two MESs $\mathcal{A}' = a'_0 a'_1 \ldots$ and $\mathcal{B}' = b'_0 b'_1 \ldots$,

$$\mathrm{PP}|\mathcal{A}' \wedge \mathcal{B}' \equiv \mathrm{PP}|\mathcal{A}' \equiv \mathrm{PR}|\mathcal{A}' \wedge \mathcal{B}' \tag{9}$$

holds. Let Z^q be the q-th transcript (X^q, W^q, Y^q). Then, we obtain

$$P^{\mathrm{PR}}_{a'_q b'_q | Z^{q-1} X_q w_q a'_{q-1} b'_{q-1}} \leq P^{\mathrm{PP}}_{a'_q | Z^{q-1} X_q w_q a'_{q-1}} \tag{10}$$

since the r.h.s. of Eq. (10) is always 0 or 1 and if it is 0, then the l.h.s. is also 0 (recall that Eq. (10) means that the inequality holds for all possible arguments). From Eqs. (9) and (10) and Lemma 3, there is an MES defined for PP, $\mathcal{C}'$, such that

$$\mathrm{PP}^{\mathcal{A}' \wedge \mathcal{B}' \wedge \mathcal{C}'} \equiv \mathrm{PP}^{\mathcal{A}' \wedge \mathcal{C}'} \equiv \mathrm{PR}^{\mathcal{A}' \wedge \mathcal{B}'} \tag{11}$$

holds true. It is easy to see that $\mathcal{A}' \wedge \mathcal{B}'$ is equivalent to $\mathcal{A} \wedge \mathcal{B}$ where $\mathcal{A} = a_0 a_1 \ldots$ and $\mathcal{B} = b_0 b_1 \ldots$ are defined for $\mathrm{TW}[\mathsf{P}, \mathsf{P}]$ and $\mathrm{TW}[\mathsf{P}, \mathsf{R}]$. From this fact, and Eq. (11), and Lemma 4, we obtain

$$\mathrm{TW}[\mathsf{P}, \mathsf{P}]^{\mathcal{A} \wedge \mathcal{B} \wedge \mathcal{C}} \equiv \mathrm{TW}[\mathsf{P}, \mathsf{R}]^{\mathcal{A} \wedge \mathcal{B}}, \text{ for some MES } \mathcal{C} \text{ defined for } \mathrm{TW}[\mathsf{P}, \mathsf{P}]. \tag{12}$$

Combining Eq. (12) and Theorem 3 proves the lemma. □

Next, we have to evaluate the r.h.s. of Eq. (8). Our result is the following.

Lemma 2. *Let γ, ϵ, and ρ be given such that $F_{K''}$ satisfies the following three conditions when $V, V' \in_{\mathrm{u}} \Sigma^n$ and V is independent of V'.*

1. $\max_{l \in \Sigma^n, c \in \Sigma^n} \Pr(F_{K''}(l, V) = c) \leq \gamma$ *(here, probability is defined by $K'' \in_{\mathrm{u}} \mathcal{K}''$ and $V \in_{\mathrm{u}} \Sigma^n$).*
2. $\max_{l, l' \in \mathcal{L}, l \neq l', c \in \Sigma^n} \Pr(F_{K''}(l, V) \oplus F_{K''}(l', V) = c) \leq \epsilon$ *and* $\max_{l, l' \in \mathcal{L}, c \in \Sigma^n} \Pr(F_{K''}(l, V) \oplus F_{K''}(l', V') = c) \leq \epsilon$.
3. $\max_{l \in \mathcal{L}, c \in \Sigma^n} \Pr(F_{K''}(l, V) \oplus V = c) \leq \rho$.

Then we have

$$\nu(\mathrm{TW}[\mathsf{P}, \mathsf{R}], \overline{a_q \wedge b_q}) \leq \left(\gamma + \epsilon + \rho + \frac{1}{2^{n+1}}\right) q^2. \tag{13}$$

Proof. Let $\widetilde{\mathsf{P}}$ be the n-bit tweakable URP with tweak space $\mathcal{T} = \mathcal{L} \times \Sigma^n$. All variables and events defined for $\mathrm{TW}[\mathsf{P}, \mathsf{R}]$ are similarly defined for $\widetilde{\mathsf{P}}$ by using dummy functions. For example, $S_i = F_{K''}(L, \mathsf{R}(N)) \oplus M_i$ and $a_q = \mathrm{dist}(S^q, \mathrm{uni}(N^q))$. Note that $\mathrm{dist}(S^q)$ and $\mathrm{dist}(U^q)$ are equivalent in $\mathrm{TW}[\mathsf{P}, \mathsf{R}]$, but not in $\widetilde{\mathsf{P}}$. Let Z^q be the q-th transcript (M^q, C^q, T^q), and X_q be the q-th query (i.e., X_q is (M_q, T_q) or (C_q, T_q)), and Y_q be the q-th answer from the oracle, which is M_q or C_q. Let K_Δ be the key of Δ, which determines the instance of $(\mathsf{R}, F_{K''})$. From the assumption, K_Δ is uniformly distributed over $\mathcal{K}_\Delta \overset{\mathrm{def}}{=} \{f : \Sigma^n \to \Sigma^n\} \times \mathcal{K}''$ and independent of P.

Then, it is easy to verify that $P^{\mathrm{TW}[\mathsf{P},\mathsf{R}]}_{Y_q|Z^{q-1}X_q a_q b_q K_\Delta} = P^{\widetilde{\mathsf{P}}}_{Y_q|Z^{q-1}X_q a_q b_q K_\Delta}$ and $P^{\mathrm{TW}[\mathsf{P},\mathsf{R}]}_{K_\Delta|Z^{q-1}X_q a_q b_q} = P^{\widetilde{\mathsf{P}}}_{K_\Delta|Z^{q-1}X_q a_q b_q}$ hold. Therefore, we have

$$P^{\mathrm{TW}[\mathsf{P},\mathsf{R}]}_{Y_q|Z^{q-1}X_q a_q b_q} = \sum_{K_\Delta} P^{\mathrm{TW}[\mathsf{P},\mathsf{R}]}_{Y_q|Z^{q-1}X_q a_q b_q K_\Delta} \cdot P^{\mathrm{TW}[\mathsf{P},\mathsf{R}]}_{K_\Delta|Z^{q-1}X_q a_q b_q} \tag{14}$$

$$= \sum_{K_\Delta} P^{\widetilde{\mathsf{P}}}_{Y_q|Z^{q-1}X_q a_q b_q K_\Delta} \cdot P^{\widetilde{\mathsf{P}}}_{K_\Delta|Z^{q-1}X_q a_q b_q} = P^{\widetilde{\mathsf{P}}}_{Y_q|Z^{q-1}X_q a_q b_q}, \tag{15}$$

where summations are taken for all $\delta \in \mathcal{K}_\Delta$. This indicates the following conditional equivalence.

$$\mathrm{TW}[\mathsf{P},\mathsf{R}]|\mathcal{A} \wedge \mathcal{B} \equiv \widetilde{\mathsf{P}}|\mathcal{A} \wedge \mathcal{B}. \tag{16}$$

Then, we determine if

$$P^{\widetilde{\mathsf{P}}}_{a_q b_q|Z^{q-1}X_q a_{q-1} b_{q-1} K_\Delta} \leq P^{\mathrm{TW}[\mathsf{P},\mathsf{R}]}_{a_q b_q|Z^{q-1}X_q a_{q-1} b_{q-1} K_\Delta} \tag{17}$$

holds. We first analyze the r.h.s. of Eq. (17). Let us assume the variables in the condition are fixed such as $(Z^{q-1}, X_q, K_\Delta) = (z^{q-1}, x^q, \delta)$ and the q-th query is a chosen-plaintext query. Then, all variables except U_q are uniquely determined. Therefore, whether $a_q^+ \overset{\mathrm{def}}{=} a_q \wedge \mathrm{dist}(U^{q-1}, \mathrm{uni}(V^q))$ holds or not is a function of (Z^{q-1}, X_q, K_Δ). If a_q^+ holds, then U_q is uniform over $\Omega \overset{\mathrm{def}}{=} \Sigma^n \setminus \{U_1 \ldots, U_{q-1}\}$, and $a_q \wedge b_q$ occurs if $U_q \in \Omega \setminus \{V_1, \ldots, V_{q-1}\}$. Note that $\{U_1 \ldots, U_{q-1}\} \cap \{V_1, \ldots, V_q\} = \emptyset$ if a_q^+ holds. From these observations, we have

$$P^{\mathrm{TW}[\mathsf{P},\mathsf{R}]}_{a_q b_q|Z^{q-1}X_q a_{q-1} b_{q-1} K_\Delta} = \begin{cases} 0 & \text{if } a_q^+ \text{ does not hold,} \\ 1 - \frac{\theta}{2^n-(q-1)} & \text{otherwise ,} \end{cases} \tag{18}$$

where θ denotes the number of unique elements among $\{V_1, \ldots, V_q\}$. How about the l.h.s. of Eq. (17)? The occurrence of a_q^+ is a function of (Z^{q-1}, X_q, K_Δ) as well as the r.h.s. However, the distribution of U_q is different. Let Ψ be a set of indexes defined by $\Psi = \{i \in \{1, \ldots, q-1\} : T_q = T_i\}$ and let $|\Psi|$ be ψ. If a_q^+ holds, U_q is uniform over $\Omega' \overset{\mathrm{def}}{=} \Sigma^n \setminus \{U_i : i \in \Psi\}$ and $a_q \wedge b_q$ occurs if $U_q \in \Omega' \setminus \{\{V_1, \ldots, V_q\} \cup \{U_i : i \in \Psi^c\}\}$. Therefore, we have

$$P^{\widetilde{\mathsf{P}}}_{a_q b_q|Z^{q-1}X_q a_{q-1} b_{q-1} K_\Delta} = \begin{cases} 0 & \text{if } a_q^+ \text{ does not hold,} \\ 1 - \frac{\theta+q-\psi-1}{2^n-\psi} & \text{otherwise .} \end{cases} \tag{19}$$

Note that $0 \leq \psi \leq q-1$ and $1 \leq \theta \leq q$. Thus, when $q \leq 2^n - \theta + 1$ we obtain

$$\frac{\theta+q-\psi-1}{2^n-\psi} - \frac{\theta}{2^n-(q-1)} = \frac{(q-\psi-1)\cdot(2^n-(q-1)-\theta)}{(2^n-\psi)\cdot(2^n-(q-1))} \geq 0. \tag{20}$$

Since $\theta \leq q$, Eq. (20) holds unless $q > 2^{n-1} + 0.5$. The same analysis holds when the q-th query is a chosen-ciphertext query. Therefore, Eq. (17) holds if $q \leq 2^{n-1}$.

It is almost trivial to see that $P^{\mathrm{TW}[\mathsf{P},\mathsf{R}]}_{K_\Delta|Z^{q-1}X_q a_{q-1} b_{q-1}} = P^{\widetilde{\mathsf{P}}}_{K_\Delta|Z^{q-1}X_q a_{q-1} b_{q-1}}$. By combining this and Eqs. (17), we have

$$P^{\widetilde{\mathsf{P}}}_{a_q b_q|Z^{q-1}X_q a_{q-1} b_{q-1}} \le P^{\mathrm{TW}[\mathsf{P},\mathsf{R}]}_{a_q b_q|Z^{q-1}X_q a_{q-1} b_{q-1}}, \quad \text{if } q \le 2^{n-1}. \tag{21}$$

From this inequality, and Eq. (16), and Lemma 3, $\mathrm{TW}[\mathsf{P},\mathsf{R}]^{\mathcal{A}\wedge\mathcal{B}\wedge\mathcal{C}} \equiv \widetilde{\mathsf{P}}^{\mathcal{A}\wedge\mathcal{B}}$ holds for some MES $\mathcal{C} = c_0 c_1 \ldots$, if $q \le 2^{n-1}$. Therefore, using Lemma 5, we obtain

$$\nu(\mathrm{TW}[\mathsf{P},\mathsf{R}], \overline{a_q \wedge b_q}) \le \nu(\mathrm{TW}[\mathsf{P},\mathsf{R}], \overline{a_q \wedge b_q \wedge c_q}) = \nu(\widetilde{\mathsf{P}}, \overline{a_q \wedge b_q}), \text{ if } q \le 2^{n-1}.$$

Note that any adversary's strategy against $\widetilde{\mathsf{P}}$ must be independent of R and $F_{K''}$, as they do not affect the input or output of $\widetilde{\mathsf{P}}$. Therefore, evaluating $\nu(\widetilde{\mathsf{P}}, \overline{a_q \wedge b_q})$ is quite easy: we only have to consider non-adaptive strategies. Let $P^{\widetilde{\mathsf{P}}}$ denote the probability space defined by $\widetilde{\mathsf{P}}$ and some fixed q inputs. Then, the second condition of Lemma 2 implies that $P^{\widetilde{\mathsf{P}}}(S_i = S_j) \le \epsilon$ and $P^{\widetilde{\mathsf{P}}}(U_i = U_j) \le \epsilon$ if $i \ne j$. Moreover, we have $P^{\widetilde{\mathsf{P}}}(S_i = N_j) = P^{\widetilde{\mathsf{P}}}(\widehat{V}_i = N_j \oplus M_i) \le \gamma$ for any i, j, and $P^{\widetilde{\mathsf{P}}}(U_i = V_j) = P^{\widetilde{\mathsf{P}}}(\widehat{V}_i = C_i \oplus V_j) \le \rho$ for any i, j (more precisely, it is at most ρ if $N_i = N_j$ and $1/2^n$ otherwise). Therefore, if $q \le 2^{n-1}$, we have

$$\begin{aligned}\nu(\widetilde{\mathsf{P}}, \overline{a_q \wedge b_q}) &\le P^{\widetilde{\mathsf{P}}}(\overline{\mathrm{dist}(S^q)}) + P^{\widetilde{\mathsf{P}}}(\overline{\mathrm{dist}(U^q)}) + P^{\widetilde{\mathsf{P}}}(S_i = N_j \text{ for some } i, j \le q) \\ &\quad + P^{\widetilde{\mathsf{P}}}(U_i = V_j \text{ for some } i, j \le q) + P^{\widetilde{\mathsf{P}}}(V_i = V_j \text{ for some } i, j \le q,\, i \ne j) \\ &\le 2\binom{q}{2}\epsilon + q^2\gamma + q^2\rho + \binom{q}{2}2^{-n} \le q^2(\epsilon + \gamma + \rho + 2^{-n-1}). \end{aligned} \tag{22}$$

This upper bound reaches 1 if $q \sim 2^{n/2}$, thus the condition $q \le 2^{n-1}$ is redundant. This concludes the proof. □

Note that the second condition of Lemma 2 implies that the offset function is ϵ-AXU for input $T = (L, N) \in \mathcal{L} \times \Sigma^n$. By combining Eq. (7) and Lemmas 1, 2 and Theorem 1, the security of $\mathrm{TW}[\mathsf{P},\mathsf{P}]$ is proved in the following theorem.

Theorem 4. *If the assumption of Lemma 2 holds true for $F_{K''}$, we have*

$$\mathtt{Adv}^{\widetilde{\mathtt{sprp}}}_{\mathrm{TW}[\mathsf{P},\mathsf{P}]}(q) \le \left(2\epsilon + \gamma + \rho + \frac{1}{2^{n+1}}\right) q^2. \tag{23}$$

The proof of Theorem 2. From Theorem 4, we can easily see that the following offset function enables a simple one-key tweakable block cipher.

Corollary 1. *Let* $\mathrm{TW}[\mathsf{P},\mathsf{P}]$ *use the offset function defined as* $\Delta(T) = L \cdot E_K(N)$, *where* $T = (L, N) \in (\Sigma^n \setminus \{0,1\}) \times \Sigma^n$. *Then, we have* $\mathtt{Adv}^{\widetilde{\mathtt{sprp}}}_{\mathrm{TW}[\mathsf{P},\mathsf{P}]}(q) \le \frac{4.5q^2}{2^n}$. *Moreover, for any block cipher* E_K,

$$\mathtt{Adv}^{\widetilde{\mathtt{sprp}}}_{\mathrm{TW}[E_K,E_K]}(q,\tau) \le \mathtt{Adv}^{\mathtt{sprp}}_{E_K}(2q,\tau') + \frac{4.5q^2}{2^n}, \text{ where } \tau' = \tau + O(q).$$

Proof. Note that $L \cdot V$, $L \cdot V \oplus L' \cdot V$, and $L \cdot V \oplus V$ are permutations of V for any $L, L' \in \Sigma^n \setminus \{0,1\}$ such that $L \neq L'$. This indicates $\epsilon = \gamma = \rho = 1/2^n$ and thus Theorem 4 proves the first claim. The second claim follows from the first and the standard conversion from the information-theoretic setting to the computational setting. □

Recall that an output of XEX's offset function is $\prod_{j=1}^{d} \alpha_j^{i_j} \cdot E_K(N)$, where a tweak is $(i_1, \ldots, i_d, N)$. In the fixed XEX, $\prod_{j=1}^{d} \alpha_j^{i_j} \neq \prod_{j=1}^{d} \alpha_j^{i'_j}$ whenever $(i_1, \ldots, i_d) \neq (i'_1, \ldots, i'_d)$, and $\prod_{j=1}^{d} \alpha_j^{i_j}$ never be 0 or 1 (in GF(2^n)). Therefore, Theorem 2 is immediately obtained from Corollary 1.

Applications of Theorem 4. Theorem 4 provides not only the improved proof of XEX, but also useful tools for the design of strong tweakable block cipher. For example, consider the LRW mode based on a dedicated AXU hash function such as MMH or NMH (see e.g., [2]). Then, Theorem 4 tells us what properties are needed (in addition to the AXU property) if we want to substitute (a part of) the key of LRW's offset function with an encryption of the block cipher. This is achieved by our generalized construction. In particular, for the offset function of the form $\Delta(L, N) = g(L \oplus E_K(N))$ where g is a fixed n-bit permutation, the conditions of Lemma 2 become simpler: since $g(l \oplus V)$ and $g(l \oplus V) \oplus g(l' \oplus V')$ are permutations of V, γ and the second ϵ in the second condition are naturally $1/2^n$. The remaining conditions can be interpreted such that g is differentially ϵ-uniform [15] and is a $(2^n\rho - 1)$-almost orthomorphism [21] (equivalently, a permutation with maximum self-differential probability ρ [13], where self-differential means the differential between the input and output). An example of such a permutation is the inversion on GF(2^n), inv($*$), where $\mathrm{inv}(x) = x^{-1}$ if $x \neq 0$, and $\mathrm{inv}(0) = 0$. If g is the inversion on GF(2^n), $\epsilon = 4/2^n$ holds from [15], and a simple analysis proves that $\rho = 3/2^n$. Consequently, the mode with the offset function $\Delta(L, N) = \mathrm{inv}(L \oplus E_K(N))$ is provably secure and has the bound $(2\epsilon + \gamma + \rho + 0.5)\frac{q^2}{2^n} = \frac{12.5q^2}{2^n}$. This demonstrates that strong tweakable ciphers with arbitrary tweak update are possible from permutations with good differential and self-differential property. We will use this idea in the next section.

5 Improving LRW-AES

Theorem 4 also gives some improvements to the LRW-AES described in Sect. 3.2. Here, we propose two improvements.

LRW-AES-Sqr: One-Key LRW-AES having 2^n tweak values. As mentioned, LRW-AES is the mode for AES that provides a strong tweakable block cipher using $\Delta(T) = K_\Delta \cdot T$, where $T \in \Sigma^n$ and $K_\Delta \in_{\mathrm{u}} \Sigma^n$ is independent of the key of the AES. Although the original LRW-AES needs two keys, Corollary 1 provides some ways to reduce these two keys to the one AES key. The simplest fix is the same as one used for the XEX: let $K_\Delta = E_K(0)$ and $T \in \Sigma^n \setminus \{0,1\}$. However, the resulting mode is not strictly compatible with LRW-AES because of the reduced tweak set. However, we still have several options to achieve one-key LRW-AES having 2^n tweak values. An efficient one among these options is to use squaring, which is as follows. We first generate $V = E_K(0)$ in the preprocessing. For tweak $T \in \Sigma^n$, the offset function is defined as:

$$\Delta(T) = \begin{cases} V^2 & \text{if } T = 0, \\ a \cdot V^2 & \text{if } T = 1, \\ T \cdot V & \text{otherwise.} \end{cases} \tag{24}$$

Here, a is a fixed element of $\Sigma^n \setminus \{0,1\}$. This requires only one AES encryption in the preprocessing, and the cost for updating a tweak (i.e., the cost for computing $\Delta(T)$) is almost the same as that of the original LRW-AES, namely one GF multiplication. To be precise, the computation of $a \cdot V^2$ requires two multiplications; however the cost for multiplication by constant a can be negligibly small with the powering-up construction. The security of this scheme, which we call LRW-AES-Sqr, is proved as follows.

Theorem 5. $\mathtt{Adv}^{\widetilde{\mathtt{sprp}}}_{\text{LRW-AES-Sqr}}(q,\tau) \le \mathtt{Adv}^{\mathtt{sprp}}_{\text{AES}_K}(q+1,\tau') + \frac{7.5q^2}{2^{128}}$, *where* $\tau' = \tau + O(q)$.

Proof. We apply Lemma 2 to the offset function in Eq. (24). Since squaring in a field with characteristic two is a permutation, both V^2 and $a \cdot V^2$ are permutations of V. Also, $T \cdot V$ with $T \notin \{0,1\}$ is a permutation. Thus we have $\gamma = 1/2^n$. Every sum of two offset values (i.e., $T \cdot V \oplus T' \cdot V$, $V^2 \oplus a \cdot V^2$, $V^2 \oplus T \cdot V$, and $a \cdot V^2 \oplus T \cdot V$ for any $T, T' \in \Sigma^n \setminus \{0,1\}$ with $T \neq T'$) is a quadratic or linear function of V, but can not be reduced to a constant since $a \notin \{0,1\}$. As a function with degree d has at most d solutions, every sum has bias of at most $2/2^n$, which means $\epsilon = 2/2^n$. Moreover, both $V^2 \oplus V$ and $a \cdot V^2 \oplus V$ have bias $2/2^n$, and $T \cdot V \oplus V$ with $T \notin \{0,1\}$ has bias $1/2^n$. Therefore we have $\rho = 2/2^n$. Note that AES is invoked $q+1$ times in LRW-AES-Sqr. Combining these facts and Theorem 4 proves the theorem. □

LRW-AES-4r: LRW-AES WITHOUT MULTIPLICATION. Both LRW-AES and LRW-AES-Sqr require GF multiplication in order to be able to update a tweak arbitrarily. Here, we provide an interesting alternative to the multiplication: the reduced-round of AES. This idea is basically the same as the recent proposal of AES-based message authentication codes [13]. More precisely, what we use is the 4-round AES, denoted by $\text{AES}^{(4)}_{K_{\text{sub}}}$, where $K_{\text{sub}} \in_{\text{u}} (\Sigma^{128})^3$ consists of the round keys for the last three rounds. The first round key is set to 0. We first generate $V = \text{AES}_K(0)$ and $K_{\text{sub}} \in_{\text{u}} (\Sigma^{128})^3$. For tweak $T \in \Sigma^{128}$, we use the offset function such as $\Delta(T) = \text{AES}^{(4)}_{K_{\text{sub}}}(T \oplus V)$. This scheme, which we call LRW-AES-4r is essentially AES-based while the cost for updating a tweak is less than an AES encryption. XEX mode also has this property (if we fix N to some constant), but a tweak can be updated only in an incremental order.

SECURITY OF LRW-AES-4r. The differential and linear properties of the AES and its reduced-round version have been extensively studied. Particularly,

Table 1. Mean speed of LRW-AES and our improvements for random 2^{20} messages and tweaks on a PC (Pentium III (Coppermine), 1 GHz, 16KB L1 cache). Alg 1 and 2 denote the multiplication algorithms specified in [23]. Preprocessing includes key schedulings for both AES and its inverse, and precomputation for multiplication, and one AES encryption: $V = \mathrm{AES}_K(0)$.

Mode	Preproc (cycles)	Enc (cycle/byte)	Dec (cycle/byte)
LRW-AES (alg 1)	1248	234	241
LRW-AES (alg 2)	289506	155	161
LRW-AES-Sqr (alg 1)	1696	235	241
LRW-AES-Sqr (alg 2)	289966	155	161
LRW-AES-4r	1653	39	45

Keliher proved that the maximum expected differential probability of $\mathrm{AES}^{(4)}_{K_{\mathrm{sub}}}$ was at most 2^{-113} [8], if $K_{\mathrm{sub}} \in_{\mathrm{u}} (\Sigma^{128})^3$. This means that $\mathrm{AES}^{(4)}_{K_{\mathrm{sub}}}(T \oplus V)$ is 2^{-113}-AXU, when (K_{sub}, V) is the key and T is the input.

The security of LRW-AES-4r is proved as follows.

Theorem 6. $\mathtt{Adv}^{\widetilde{\mathtt{sprp}}}_{\text{LRW-AES-4r}}(q, \tau) \leq \mathtt{Adv}^{\mathtt{sprp}}_{\mathrm{AES}_K}(q+1, \tau') + \frac{(2^{16}+2.5)q^2}{2^{128}}$, *where* $\tau' = \tau + O(q)$.

Proof. We have $\epsilon < 2^{-113} = 2^{15}/2^{128}$ from [8]. Moreover, we have $\gamma = 1/2^{128}$. Note that the output of $\mathrm{AES}^{(4)}_{K_{\mathrm{sub}}}$ is completely random and independent of the input, as each round key is XORed to the intermediate message and uniformly distributed. This indicates $\rho = 1/2^{128}$. □

We have to mention that LRW-AES-4r is not an ideal substitute for the LRW-AES. The security of the LRW-AES-4r is moderately degraded compared with the original LRW-AES. That is, LRW-AES-4r has $112/2 = 56$-bit security (i.e., q must be much smaller than 2^{56}), while the original LRW-AES has 63-bit security. This means that the lifetime of key should be slightly shortened. In addition, the key of the LRW-AES-4r is longer (512 bits) than that of the LRW-AES (256 bits), though both require only one AES key scheduling. We implemented our proposals and the original LRW-AES in software. Our implementation was based on the reference AES code [22]. We used two naive algorithms for multiplication in $\mathrm{GF}(2^{128})$ that were specified in the document of LRW-AES [23]. The performance of LRW-AES-4r is quite remarkable, as Table 1 shows.

Acknowledgments

We would like to thank Peng Wang for pointing out the reference on XEX. We also thank Etsuko Tsujihara for the implementation and anonymous reviewers for very useful comments.

References

1. Bellare, M., Desai, A., Jokipii, E., Rogaway, P.: A Concrete Security Treatment of Symmetric Encryption. In: Proceedings of the 38th Annual Symposium on Foundations of Computer Science, FOCS '97, pp. 394–403 (1997)
2. Black, J.: Message Authentication Code. PhD dissertation (2000)
3. Goldreich, O.: Modern Cryptography, Probabilistic Proofs and Pseudorandomness. Springer, Heidelberg
4. Halevi, S., Rogaway, P.: A Tweakable Enciphering Mode. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 482–499. Springer, Heidelberg (2003)
5. Halevi, S., Rogaway, P.: A Parallelizable Enciphering Mode. In: Okamoto, T. (ed.) CT-RSA 2004. LNCS, vol. 2964, pp. 292–304. Springer, Heidelberg (2004)
6. Iwata, T., Kurosawa, K.: On the Universal Hash Functions in Luby-Rackoff Cipher. In: Lee, P.J., Lim, C.H. (eds.) ICISC 2002. LNCS, vol. 2587, pp. 226–236. Springer, Heidelberg (2003)
7. Iwata, T., Kurosawa, K.: OMAC: One-Key CBC MAC. In: Johansson, T. (ed.) FSE 2003. LNCS, vol. 2887, pp. 129–153. Springer, Heidelberg (2003)
8. Keliher, L., Sui, J.: Exact Maximum Expected Differential and Linear Probability for 2-Round Advanced Encryption Standard (AES). IACR ePrint Archive (2005)/321
9. Kilian, J., Rogaway, P.: How to Protect DES Against Exhaustive Key Search. In: Koblitz, N. (ed.) CRYPTO 1996. LNCS, vol. 1109, pp. 252–267. Springer, Heidelberg (1996)
10. Liskov, M., Rivest, R., Wagner, D.: Tweakable Block Ciphers. In: Yung, M. (ed.) CRYPTO 2002. LNCS, vol. 2442, pp. 31–46. Springer, Heidelberg (2002)
11. Maurer, U.: Indistinguishability of Random Systems. In: Knudsen, L.R. (ed.) EUROCRYPT 2002. LNCS, vol. 2332, pp. 110–132. Springer, Heidelberg (2002)
12. Maurer, U., Pietrzak, K.: Composition of Random Systems: When Two Weak Make One Strong. In: Naor, M. (ed.) TCC 2004. LNCS, vol. 2951, pp. 410–427. Springer, Heidelberg (2004)
13. Minematsu, K., Tsunoo, Y.: Provably Secure MACs From Differentially-uniform Permutations and AES-based Implementations. In: Robshaw, M. (ed.) FSE 2006. LNCS, vol. 4047, Springer, Heidelberg (2006)
14. Naor, M., Reingold, O.: On the Construction of Pseudorandom Permutations: Luby-Rackoff Revisited. Journal of Cryptology 12(1), 29–66 (1999)
15. Nyberg, K.: Differentially Uniform Mappings for Cryptography. In: Helleseth, T. (ed.) EUROCRYPT 1993. LNCS, vol. 765, pp. 55–64. Springer, Heidelberg (1994)
16. Pietrzak, K.: Composition Does Not Imply Adaptive Security. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 55–65. Springer, Heidelberg (2005)
17. Rogaway, P., Bellare, M., Black, J., Krovetz, T.: OCB: a block-cipher mode of operation for efficient authenticated encryption. In: ACM Conference on Computer and Communications Security ACM CCS'01, pp. 196–205 (2001)
18. Rogaway, P.: Efficient Instantiations of Tweakable Blockciphers and Refinements to Modes OCB and PMAC (the early version of [19]), http://www.cs.ucdavis.edu/~rogaway/papers/offsets.pdf
19. Rogaway, P.: Efficient Instantiations of Tweakable Blockciphers and Refinements to Modes OCB and PMAC. In: Lee, P.J. (ed.) ASIACRYPT 2004. LNCS, vol. 3329, pp. 16–31. Springer, Heidelberg (2004)
20. Wegman, M., Carter, L.: New Hash Functions and Their Use in Authentication and Set Equality. Journal of Computer and System Sciences 22, 265–279 (1981)

21. Vaudenay, S.: On the Lai-Massey Scheme. In: Lam, K.-Y., Okamoto, E., Xing, C. (eds.) ASIACRYPT 1999. LNCS, vol. 1716, pp. 9–19. Springer, Heidelberg (1999)
22. `http://homes.esat.kuleuven.be/~rijmen/rijndael/rijndael-fst-3.0.zip`
23. Draft Proposal for Tweakable Narrow-block Encryption (2004), `http://www.siswg.org/docs/LRW-AES-10-19-2004.pdf`

A Theorems and Lemmas Proved by Maurer [11]

Let us now describe some of Maurer's results [11] other than Theorem 3. They were used in our analysis.

Lemma 3. *(Lemma 1 (iv) of [11]) Let MESs $\mathcal{A}$ and $\mathcal{B}$ be defined for F and G. Moreover, let X_i and Y_i denote the i-th input and output of F (or G), respectively. Assume $F|\mathcal{A} \equiv G|\mathcal{B}$. If $P^F_{a_i|X^iY^{i-1}a_{i-1}} \le P^G_{b_i|X^iY^{i-1}b_{i-1}}$ for $i \ge 1$, which means $P^F_{a_i|X^iY^{i-1}a_{i-1}}(x^i, y^{i-1}) \le P^G_{b_i|X^iY^{i-1}b_{i-1}}(x^i, y^{i-1})$ holds for all (x^i, y^{i-1}) such that $P^F_{a_{i-1}|X^{i-1}Y^{i-1}}(x^{i-1}, y^{i-1})$ and $P^G_{b_{i-1}|X^{i-1}Y^{i-1}}(x^{i-1}, y^{i-1})$ are positive, then there exists an MES $\mathcal{C}$ defined for G such that $F^{\mathcal{A}} \equiv G^{\mathcal{B}\wedge\mathcal{C}}$.*

Lemma 4. *(Lemma 4 (ii) of [11]) Let F and G be two compatible keyed functions, and $\mathbb{F}$ be the function of F and G (i.e., $\mathbb{F}[F]$ is a function that internally invokes F, possibly several times, to process its inputs). Here, $\mathbb{F}$ can be probabilistic, and if so, we assume $\mathbb{F}$ is independent of F or G. If $F^{\mathcal{A}} \equiv G^{\mathcal{B}}$ holds for some MESs $\mathcal{A}$ and $\mathcal{B}$, we have $\mathbb{F}[F]^{\mathcal{A}'} \equiv \mathbb{F}[G]^{\mathcal{B}'}$. Here, MES $\mathcal{A}' = a'_0 a'_1 \dots$ is defined such that a'_i denotes $\mathcal{A}$-event is satisfied for the time period i. For example, if $\mathbb{F}[F]$ always invoke F c times for any input, then $a'_i = a_{ci}$. $\mathcal{B}'$ is defined in the same way.*

Lemma 5. *(Lemma 6 (ii) of [11]) If $F^{\mathcal{A}} \equiv G^{\mathcal{B}}$, then $\nu(F, \overline{a_q}) = \nu(G, \overline{b_q})$.*

Lemma 6. *(Lemma 6 (iii) of [11]) $\nu(F, \overline{a_q \wedge b_q}) \le \nu(F, \overline{a_q}) + \nu(F, \overline{b_q})$.*

B Proof of Theorem 1

The structure of the proof is almost the same as the proofs of Lemmas 1 and 2. Let $\widetilde{E}$ denote the LRW mode using the offset function Δ, and $\widetilde{\mathsf{P}}$ denote the URP compatible (i.e., the block size and tweak space are the same as those of $\widetilde{E}$) with $\widetilde{E}$. Let M_i and C_i be the i-th plaintext and ciphertext, and let T_i be the i-th tweak. Let S_i be the i-th input to E_K, i.e., $S_i = \Delta(T_i) \oplus M_i$. Similarly, we define $U_i = \Delta(T_i) \oplus C_i$. Note that these variables can be defined for both $\widetilde{E}$ and $\widetilde{\mathsf{P}}$. We use two MESs $\mathcal{A} = a_0 a_1 \dots$ and $\mathcal{B} = b_0 b_1 \dots$ where $a_i \overset{\text{def}}{=} \text{dist}(S^i)$ and $b_i \overset{\text{def}}{=} \text{dist}(U^i)$. An analysis similar to that used in the proof of Lemma 2 provides that the equivalences $\widetilde{E}|\mathcal{A} \equiv \widetilde{\mathsf{P}}|\mathcal{A} \wedge \mathcal{B}$ and $\widetilde{E}^{\mathcal{A}\wedge\mathcal{C}} \equiv \widetilde{\mathsf{P}}^{\mathcal{A}\wedge\mathcal{B}}$ hold for some MES $\mathcal{C} = c_0 c_1 \dots$ defined for $\widetilde{E}$. Thus we have

$$\mathtt{Adv}^{\widetilde{\mathtt{sprp}}}_{\widetilde{E}}(q) = \mathtt{Adv}^{\widetilde{\mathtt{cca}}}_{\widetilde{E},\widetilde{\mathsf{P}}}(q) \le \nu(\widetilde{\mathsf{P}}, \overline{a_q \wedge b_q}) \le \nu(\widetilde{\mathsf{P}}, \overline{a_q}) + \nu(\widetilde{\mathsf{P}}, \overline{b_q}), \tag{25}$$

where the first inequality follows from $\widetilde{E}^{\mathcal{A}\wedge\mathcal{C}} \equiv \widetilde{\mathsf{P}}^{\mathcal{A}\wedge\mathcal{B}}$ and Theorem 3, and the last follows from Lemma 6. It is almost trivial to see that any adaptive strategy against $\widetilde{\mathsf{P}}$ to invoke $\overline{a_q}$ or $\overline{b_q}$ is no better than the best non-adaptive strategy. Therefore, we have

$$\nu(\widetilde{\mathsf{P}}, \overline{a_q}) \leq \max_{m^q, t^q} P^{\widetilde{\mathsf{P}}}(S_i = S_j \text{ for some } 1 \leq i < j \leq q | M^q = m^q, T^q = t^q) \leq \epsilon q^2/2,$$

where $P^{\widetilde{\mathsf{P}}}$ denotes the probability space defined by $\widetilde{\mathsf{P}}$ and the maximum is taken for all q plaintexts and tweaks satisfying $(m_i, t_i) \neq (m_j, t_j)$ for any $i \neq j$. The second inequality follows from the assumption on Δ. Similarly, we obtain $\nu(\widetilde{\mathsf{P}}, \overline{b_q}) \leq \epsilon q^2/2$, and thus, $\nu(\widetilde{\mathsf{P}}, \overline{a_q}) + \nu(\widetilde{\mathsf{P}}, \overline{b_q}) \leq \epsilon q^2$. This concludes the proof.

C An Attack Against OCB1 Using Flawed XEX^{-1}

OCB1 [18] is an authenticated encryption mode for any finite-length message. A ciphertext consists of a nonce, and an encryption of a message, and an authentication tag, which we simply call a tag. The OCB1 defined in [18] and [19] are slightly different, but our attack is applicable to both. For simplicity, we only describe a version of OCB1 defined in [18] for a message of length cn bits for some positive integer c. We also assume that the tag is n-bit. Let the message M be $(M[0], \ldots, M[c-1])$, where each $M[i] \in \Sigma^n$. Let $\widetilde{E}_K$ be an n-bit block strong tweakable block cipher having the tweak space $\{0, 1, \ldots, 2^{n/2}\} \times \{0, 1\} \times \Sigma^n$. To encrypt M with nonce $N \in \Sigma^n$, we first let $C[i] = \widetilde{E}_K(M[i], (i, 0, N))$, where the second argument of $\widetilde{E}_K$ is a tweak, for $i = 0, \ldots, c-2$. The last block, $M[c-1]$, is encrypted such as $C[c-1] = M[c-1] \oplus \widetilde{E}_K(v, (c-1, 0, N))$, where v denotes the bit length of the last block, which is assumed to be n, in some deterministic encoding. Then, we compute the sum of all message blocks, namely $\mathsf{sum} = M[0] \oplus M[1] \oplus \cdots \oplus M[c-1]$. The tag is $\mathsf{tag} = \widetilde{E}_K(\mathsf{sum}, (c-1, 1, N))$, and the ciphertext C is $(N, C[0], \ldots, C[c-1], \mathsf{tag})$. To decrypt it, we compute $\widehat{M}[i] = \widetilde{E}_K^{-1}(C[i], (i, 0, N))$ for $1 \leq i \leq c-2$. For C_{c-1}, we have $\widehat{M}[c-1] = C_{c-1} \oplus \widetilde{E}_K(v, (c-1, 0, N))$ and $\widehat{\mathsf{sum}} = \widehat{M}[0] \oplus \widehat{M}[1] \oplus \cdots \oplus \widehat{M}[c-1]$. Then, we check if $\widetilde{E}_K(\widehat{\mathsf{sum}}, (c-1, 1, N))$ and tag are the same. If they are the same, we say the ciphertext is authenticated, and otherwise it is faked.

XEX^{-1} gives ciphertext $C = E_K^{-1}(M \oplus \Delta(i_1, i_2, N)) \oplus \Delta(i_1, i_2, N)$ where $\Delta(i_1, i_2, N)$ equals $2^{i_1} 3^{i_2} E_K(N)$ for all $(i_1, i_2) \in \{0, 1, \ldots, 2^{n/2}\} \times \{0, 1\}$. Recall that this provides unique representations but does not exclude a reduced-to-1 index vector. Our attack is against the tag-generation part and is based on the information of two ciphertexts. We assume the nonce is set to N at the beginning of the attack.

1. Ask the oracle (who implemented the XEX^{-1}-based OCB1) to encrypt a $2n$-bit message, $M_1 = (M_1[0], M_1[1]) = (0, m)$, for some $m \in \Sigma^n$ and receive the ciphertext $C_1 = (N, C_1[0], C_1[1], \mathsf{tag}_1)$, where $C_1[0] = \widetilde{E}_K(M_1[0], (0, 0, N)) = E_K(N) \oplus N$ and $C_1[1] = m \oplus E_K^{-1}(v \oplus 2 \cdot E_K(N)) \oplus 2 \cdot E_K(N)$. The tag is $\mathsf{tag}_1 = E_K^{-1}(m \oplus 2 \cdot 3 \cdot E_K(N)) \oplus 2 \cdot 3 \cdot E_K(N)$.

2. Ask the oracle to encrypt $M_2 = (M_2[0], M_2[1]) = (0, m')$ for some $m' \in \Sigma^n, m' \neq m$ and receive the ciphertext $C_2 = (N', C_2[0], C_2[1], \mathsf{tag}_2)$, where $C_2[0] = \widetilde{E}_K(M_2[0], (0, 0, N')) = E_K(N') \oplus N'$ and $N' \neq N$. We do not use $C_2[1]$ and tag_2.
3. Then, issue the faked ciphertext $C' = (N, C'[0], C'[1], \mathsf{tag}')$. Here, $C'[0] = C_1[0] \oplus N \oplus N'$, and $C'[1] = C_1[0] \oplus C_1[1] \oplus C_2[0] \oplus N \oplus N'$, and $\mathsf{tag}' = \mathsf{tag}_1$.

The above faked ciphertext will be always accepted as authentic by the oracle, since the decrypted message will be:

$$\widehat{M}'[0] = E_K(C'[0] \oplus E_K(N)) \oplus E_K(N) = E_K(N) \oplus E_K(N') \tag{26}$$

$$\widehat{M}'[1] = C'[1] \oplus E_K^{-1}(v \oplus 2 \cdot E_K(N)) \oplus 2 \cdot E_K(N) = E_K(N) \oplus E_K(N') \oplus m. \tag{27}$$

These equations indicate that $\widehat{\mathsf{sum}}' = \widehat{M}'[0] \oplus \widehat{M}'[1] = m$, and therefore, we have $\mathsf{tag}' = \mathsf{tag}_1$.

Extended Hidden Number Problem and Its Cryptanalytic Applications

Martin Hlaváč[1] and Tomáš Rosa[1,2,⋆]

[1] Department of Algebra, Charles University in Prague, Sokolovská 83, 186 75 Prague 8, Czech Republic
[2] eBanka, a.s., Václavské Náměstí 43, 110 00 Prague 1, Czech Republic
hlavm1am@artax.karlin.mff.cuni.cz, trosa@ebanka.cz

Abstract. Since its formulation in 1996, the Hidden Number Problem (HNP) plays an important role in both cryptography and cryptanalysis. It has a strong connection with proving security of Diffie-Hellman and related schemes as well as breaking certain implementations of DSA-like signature schemes. We formulate an extended version of HNP (EHNP) and present a polynomial time algorithm for solving its instances. Our extension improves usability of HNP for solving real cryptanalytic problems significantly. The techniques elaborated here can be used for cryptographic strength proving, as well. We then present a practically feasible side channel attack on certain implementations of DSA (e.g. OpenSSL), which emphasizes the security risk caused by a side channel hidden in the design of Pentium 4 HTT processor for applications like SSH. During experimental simulations, having observed as few as 6 authentications to the server, an attacker was able to disclose the server's private key.

Keywords: side channel analysis, cache analysis, DSA implementation, hyper-threading, sliding window, lattice.

1 Introduction

In 1996, Boneh and Venkatesan studied the following problem: Fix p and n. Having given $(k_i,\ \mathrm{MSB}_n(xg^{k_i} \,\mathbf{mod}\, p))$ for $1 \le i \le d$, where g is a generator of $\mathbb{Z}_p^*$, $\mathrm{MSB}_n(y)$ satisfies $|y - \mathrm{MSB}_n(y)| < \frac{p}{2^{n+1}}$ and d is a polynomial function of $\log p$, find $x \,\mathbf{mod}\, p$. The particular unknown value of x was called a hidden number and the whole problem was named a Hidden Number Problem (HNP). HNP plays an important role in proving security of most significant bits in Diffie-Hellman key agreement and related schemes [7]. In 1999, Nguyen showed a connection between solving HNP and breaking flawed implementations of DSA [13] attacked sooner by Howgrave-Graham and Smart [10]. It was then extended together with Shparlinski in [14]. Their formulation of HNP followed from [7] with a modification allowing them to disclose the private DSA key provided that they know a sufficiently long block of bits of each nonce for several signatures (cf. DSA description in §5.1). The limitation of the method in [14] was that

⋆ The second author was partly supported by the institutional grant MSM 0021620839.

© Springer-Verlag Berlin Heidelberg 2007

these known bits had to be consecutively placed on the position of either most or least significant bits or they had to occupy a block of bits somewhere in the middle of each nonce. An idea was given on how to overcome this restriction, based on a multidimensional diophantine approximation. Deeper investigation of it then showed that it leads to an unnecessary information loss (cf. remark on §3 below). Furthermore, the method was unaware of direct partial information an attacker may have on the private key itself. A practical cryptanalytic problem may arise, where these factors are limiting. Interestingly, the limiting focus on most significant bits also persisted in other variants derived from the original HNP [21], [6]. In this article, we show an extension of HNP (EHNP) together with a probabilistic algorithm for solving its instances in polynomial time, which relaxes the strict conditions on the form of partial information in HNP significantly. It then allows us to develop, for instance, a successful realistic side channel attack on certain implementations of DSA presented in §5. We consider the EHNP elaboration and the attack presented of independent merit, since we believe that the method presented here can be used as a general tool for solving many other cryptanalytic problems leading to instances of EHNP. Besides (EC)DSA and its variants, we can mention Diffie-Hellman and related schemes such as El Gamal public key encryption, Shamir message passing scheme, and Okamoto conference key sharing studied originally in [7]. Using EHNP allows an analyst to exploit partial information in an "individual bits"-like manner, which is very important with regard to popular side channel attacks. EHNP also allows us to study cryptographic security of individual bits of secrets in these schemes. One can, for instance, extend the results on most significant bits given in [7] to other bits or blocks of bits, as well. It is also possible to implant the ideas elaborated here into other variants of HNP, such as [21], [6]. The practical attack presented, on the other hand, illustrates the significant vulnerability that general widespread applications like SSH server [19] can acquire when using the Pentium 4 HTT processor [11] as a hosting platform.

The rest of the paper is organized as follows: In §2, we review elementary properties of lattices that we use to solve EHNP. We note that instead of a set of diophantine inequalities, we view HNP and its extensions as a special kind of a set of linear congruences of truncated variables. In this way, we get a particular case of the general one studied in [8] (cf. also [4]) which, however, leads to a far easier solution. This idea itself is not new (cf. [16]). However, we show how to exploit it to rigorously prove solvability of (E)HNP without relying on the transforming approximation suggested in [14]. Especially, we exploit the fact that there is a variable (the hidden number) that appears (with a nonzero coefficient) in each congruence from the set.

In §3, we illustrate an evolution of EHNP starting from the approach of [14]. We recall the results of [14] being important for us here in a form of theorems. To stay focused on algorithmic part of the connection in between EHNP and lattices, we omit otherwise very interesting elaboration of distribution conditions relaxation from the theorems. We slightly generalize the method used there to exploit the partial information in the middle of each nonce. We define HNP-2H problem and show how it reduces to HNP to demonstrate the idea of [14] on how to use the information spread across individual bits. Informally speaking, the authors suggested using certain kind of diophantine approximation to convert these problems to HNP. With such an approach, however, the amount of information needed per nonce grows with a multiple of the number of unknown bit blocks (as already mentioned in [14] and [16]). Contrary to this approach, the EHNP solving method presented here is compact, which means that it does not rely on the conversion causing the unnecessary information loss. In this sense, our method is similar to the one presented by Howgrave-Graham and Smart in [10] which, however, contains a lot of heuristic assumptions in comparison with the results presented here. We hope the whole elaboration in §3 is a useful guideline for a cryptanalyst deciding which method to choose by the kind and amount of information she has.

In §4, we define EHNP, present a lattice-based algorithm for searching candidate solution in polynomial time, and investigate correctness of that solution. To give an illustrative presentation of our ideas, we use certain bounds for lattice problems employed that are not very tight. Actually, we rely only on the well known algorithms of Lenstra, Lenstra, Lovász [12], and Babai [3]. Even with these conservative results, we are able to derive practically acceptable bounds for the EHNP solving algorithm. As being widely understood in the area of lattice problems [9], in practice, however, we may reasonably assume that the algorithms (or their equivalents, cf. §2) will behave much better than what would be guaranteed even by more strict bounds for more matured techniques. Therefore, we suggest an experimental verification of the EHNP instance solvability for a particular cryptanalytic problem, even when it seems to be beyond the estimates guaranteed formally.

In §5, we present a practical side channel attack on certain implementations of DSA (including the OpenSSL one) when running on the Pentium 4 HTT platform [11]. Besides being of an independent merit, it serves as an example of using EHNP approach in a real cryptanalytic case. Finally, we conclude in §6.

2 Preliminaries

The main tool of our approach is a lattice. We define a (full rank) lattice $\mathcal{L}$ in $\mathbb{Q}^d$ as the set of lattice vectors

$$\left\{ \sum_{i=1}^{d} \alpha_i \mathbf{b}_i \mid \alpha_i \in \mathbb{Z} \right\}, \tag{1}$$

where $\mathbf{b}_1, \ldots, \mathbf{b}_d \in \mathbb{Q}^d$ are linearly independent and are called basis vectors of lattice $\mathcal{L}$. The matrix whose rows are the basis vectors is called basis matrix of lattice $\mathcal{L}$. There is an algorithmic problem connected with the lattices called Approximate Closest Vector Problem (ACVP). Given d linearly independent basis vectors $\mathbf{b}_1, \ldots, \mathbf{b}_d \in \mathbb{Q}^d$ together with $\mathbf{v} \in \mathbb{Q}^d$, the task is to find a lattice vector $\mathbf{W} \in \mathcal{L}$ satisfying $\|\mathbf{v} - \mathbf{W}\| \leq f(d) \min_{\mathbf{A} \in \mathcal{L}} \|\mathbf{v} - \mathbf{A}\|$. The coefficient $f(d)$ is called the approximation factor. The problem is known to be NP-hard for $f(d) = 1$ for any norm [9]. In practical cryptanalytic cases, however, a weaker approximation still suffices. In particular, we will expect there exists a polynomial time algorithm solving ACVP with $f(d) = 2^{\frac{d}{4}}$, which was the basic assumption in [7], [14] (cf. Babai algorithm in [3]). We can get such an algorithm by using various number-theoretic libraries such as NTL [20]. Furthermore, an attacker will probably choose some of the most recent methods like [15]. Note that Algorithm 1 adapts "automatically" for such modification and the estimates of Theorem 5 can be adjusted very easily by a change of the parameter κ_D (cf. the elaboration in §4 for its precise definition).

To simplify the notation later, it is useful to define symbol $|a|_N = \min_{k \in \mathbb{Z}} |a - kN|$ where $N \in \mathbb{N}$ and $a \in \mathbb{Z}$. It is easy to see that

1. $|a+b|_N = 0 \Leftrightarrow a \equiv -b \ (\mathbf{mod}\, N)$
2. $|a+b|_N \leq |a|_N + |b|_N$
3. $|ab|_N \leq |a|_N |b|_N$
4. $|a|_N = \min\{a \,\mathbf{mod}\, N,\ N - (a \,\mathbf{mod}\, N)\}$
5. $|a|_N \leq |a|$

for all $a, b \in \mathbb{Z}$.

As N is a prime throughout the whole paper, $\mathbb{Z}_N$ is regarded as the finite field $\mathbb{Z}_N(+, \cdot, 0, 1)$. Unless stated otherwise, the elements of $\mathbb{Z}_N$ are represented as integers from the set $\{0, \ldots, N-1\}$.

3 On the Way from HNP to EHNP

Definition 1 (Hidden Number Problem). *Let N be a prime and let x, $x \in \mathbb{Z}_N$ be a particular unknown integer that satisfies d congruences*

$$\alpha_i x + \rho_i k_i \equiv \beta_i \ (\mathbf{mod}\, N), \quad 1 \leq i \leq d,$$

where α_i, $\alpha_i \not\equiv 0 \ (\mathbf{mod}\, N)$, ρ_i and β_i, $1 \leq i \leq d$ are known values. The unknown integers k_i satisfy $0 \leq k_i < 2^{\mu}$, $1 \leq i \leq d$, where μ is a known rational constant. The Hidden Number Problem (HNP) is to find x.

Theorem 1 (Solving HNP). *There exists an algorithm running in polynomial time that solves HNP instance, where α_i and ρ_i, $1 \leq i \leq d$ are uniformly and independently distributed on $\langle 1, N-1 \rangle$, with the probability of success*

$$P > 1 - \frac{2^{d\mu}}{(N-1)^{d-1}} \left(1 + 2^{\frac{d+1}{4}} (1+d)^{\frac{1}{2}}\right)^d.$$

Proof. Based on rephrasing the results from [14].

Definition 2 (HNP with Two Holes). *Let N be a prime and let x, $x \in \mathbb{Z}_N$ be a particular unknown integer that satisfies d congruences*

$$\alpha_i x + \rho_{i,1} k_{i,1} + \rho_{i,2} k_{i,2} \equiv \beta_i \ (\mathbf{mod}\, N), \quad 1 \le i \le d, \tag{2}$$

where α_i, $\alpha_i \not\equiv 0 \ (\mathbf{mod}\, N)$, $\rho_{i,1}$, $\rho_{i,2}$ and β_i, $1 \le i \le d$ are known values. The unknown integers $k_{i,1}$ and $k_{i,2}$ satisfy $0 \le k_{i,1} < 2^{\mu_1}$ and $0 \le k_{i,2} < 2^{\mu_2}$, $1 \le i \le d$, where μ_1 and μ_2 are known rational constants. The Hidden Number Problem with Two Holes (HNP-2H) is to find x.

Theorem 2 (Dirichlet, [9]). *Let $\alpha \in \mathbb{R}$ and $0 < \varepsilon \le 1$ be given values. Then there exist $p, q \in \mathbb{Z}$ such that*

$$1 \le q \le \frac{1}{\varepsilon} \quad \textit{and} \quad \left| \alpha - \frac{p}{q} \right| < \frac{\varepsilon}{q}.$$

Corollary 1. *Let us be given $A, N \in \mathbb{Z}$ and $B \in \mathbb{R}$ satisfying $B \ge 1$ and $N > 0$. Then there exists λ, $\lambda \in \mathbb{Z}$ such that*

$$1 \le \lambda \le B \quad \textit{and} \quad |\lambda A|_N < \frac{N}{B}.$$

Proof. Theorem 2 states there exist $p, q \in \mathbb{Z}$ such that $\left| \frac{A}{N} - \frac{p}{q} \right| < \frac{1}{Bq}$ and $1 \le q \le B$. So $|qA|_N \le |qA - Np| < \frac{N}{B}$. Setting $\lambda = q$ finishes the proof. □

Remark 1. Note that λ promised by Corollary 1 can be easily found in polynomial time using a technique based on the continued fractions expansion (cf. [9], [14]).

Theorem 3 (Solving HNP-2H using Dirichlet's approximation). *There exists an algorithm running in polynomial time that solves HNP-2H, where α_i, $\rho_{i,1}$, and $\rho_{i,2}$, $1 \le i \le d$ are uniformly and independently distributed on $\langle 1, N-1 \rangle$, with the probability of success*

$$P > 1 - \frac{(N 2^{\mu_1 + \mu_2})^{\frac{d}{2}}}{(N-1)^{d-1}} \left(4 + 2^{\frac{d+9}{4}} (1+d)^{\frac{1}{2}} \right)^d.$$

Proof. Let $A_i = (\rho_{i,1})^{-1} \rho_{i,2} \,\mathbf{mod}\, N$, $\gamma_i = k_{i,1} + A_i k_{i,2}$, $\alpha'_i = (\rho_{i,1})^{-1} \alpha_i \,\mathbf{mod}\, N$ and $\beta'_i = (\rho_{i,1})^{-1} \beta_i \,\mathbf{mod}\, N$, $1 \le i \le d$. The congruences (2) in Definition 2 become

$$\alpha'_i x + \gamma_i \equiv \beta'_i \ (\mathbf{mod}\, N), \quad 1 \le i \le d. \tag{3}$$

Given any $B \in \mathbb{R}, B \ge 1$, we can find non-zero integers $\lambda_{i,B}$ satisfying $|\lambda_{i,B} A_i| < \frac{N}{B}$ and $1 \le \lambda_{i,B} \le B$ for $1 \le i \le d$. It holds

$$|\lambda_{i,B} \gamma_i|_N = |\lambda_{i,B} k_{i,1} + \lambda_{i,B} A_i k_{i,2}|_N \le |\lambda_{i,B}|_N k_{i,1} + |\lambda_{i,B} A_i|_N k_{i,2} < B 2^{\mu_1} + \frac{N}{B} 2^{\mu_2}.$$

The choice $B_{min} = N^{\frac{1}{2}} 2^{\frac{\mu_2 - \mu_1}{2}}$ minimizes the upper bound $B 2^{\mu_1} + \frac{N}{B} 2^{\mu_2}$.

We convert HNP-2H to HNP by setting $k'_i = \left(\lambda_{i,B_{min}}\gamma_i + \lfloor N^{\frac{1}{2}}2^{\frac{\mu_1+\mu_2+2}{2}}\rfloor\right)$ $\mathbf{mod}\, N$, $k'_i < N^{\frac{1}{2}}2^{\frac{\mu_1+\mu_2+4}{2}}$. After several modifications of (3), we obtain congruences in one unknown variable k'_i per congruence, i.e.

$$\begin{aligned}(\lambda_{i,B_{min}}\alpha'_i)\,x + \lambda_{i,B_{min}}\gamma_i &\equiv \lambda_{i,B_{min}}\beta'_i \quad (\mathbf{mod}\, N),\\ (\lambda_{i,B_{min}}\alpha'_i)\,x + k'_i &\equiv \lambda_{i,B_{min}}\beta'_i + \left\lfloor N^{\frac{1}{2}}2^{\frac{\mu_1+\mu_2+2}{2}}\right\rfloor \quad (\mathbf{mod}\, N),\\ \alpha''_i x + k'_i &\equiv \beta''_i \quad (\mathbf{mod}\, N), \quad 1 \le i \le d\end{aligned}$$

defining an instance of HNP. Let $\mu' \in \mathbb{Q}$ be such that $k'_i < 2^{\mu'} \le N^{\frac{1}{2}}2^{\frac{\mu_1+\mu_2+4}{2}}$. By Theorem 1, such a problem can be solved in polynomial time with the probability

$$P > 1 - \frac{2^{d\mu'}}{(N-1)^{d-1}}\left(1 + 2^{\frac{d+1}{4}}(1+d)^{\frac{1}{2}}\right)^d \ge 1 - \frac{N^{\frac{d}{2}}2^{\frac{(\mu_1+\mu_2+4)d}{2}}}{(N-1)^{d-1}}\left(1 + 2^{\frac{d+1}{4}}(1+d)^{\frac{1}{2}}\right)^d =$$

$$= 1 - \frac{(N2^{\mu_1+\mu_2})^{\frac{d}{2}}}{(N-1)^{d-1}}2^{2d}\left(1 + 2^{\frac{d+1}{4}}(1+d)^{\frac{1}{2}}\right)^d = 1 - \frac{(N2^{\mu_1+\mu_2})^{\frac{d}{2}}}{(N-1)^{d-1}}\left(4 + 2^{\frac{d+9}{4}}(1+d)^{\frac{1}{2}}\right)^d.$$

□

It is correct to interpret Theorem 3 as saying that we need roughly twice as many information bits to solve HNP-2H compared to the plain HNP case [16]. This is caused by the transforming approximation and it is generally independent on the technique used to solve the transformed HNP. If we continue further this way to define HNP with more "holes" (HNP-xH), we will need to use a multidimensional transforming approximation based e.g. on the scope of the multidimensional Dirichlet's theorem and lattice reduction techniques [9]. What we obtain is then a conjecture stated in [16] that we need at least x-times as many information bits to solve HNP-xH compared to the plain HNP case. However, using the algorithmic and mainly the proving strategies described bellow, it turns out that such a conjecture does not hold generally. We can see it, for instance, by normalizing the probability estimations given above and bellow under the assumption that we have an access to an oracle solving the Closest Vector Problem with an arbitrary precision for the maximum norm. We omit this demonstration here, since it is beyond the scope of the paper. Therefore, the conjecture of [16] is not a property of HNP itself, it is a property of a particular method for solving HNP instead. On the other hand, this is not the only one selection criterion. As is demonstrated bellow, our method does not suffer from the expensive transforming approximation, but, on the other hand, it is more sensitive to the approximation factor of the particular algorithm used for solving the Approximate Closest Vector Problem.

Theorem 4 (Solving HNP-2H as a special case of EHNP). *There exists an algorithm running in polynomial time that solves HNP-2H, where α_i, $\rho_{i,1}$, and $\rho_{i,2}$, $1 \le i \le d$ are uniformly and independently distributed on $\langle 1, N-1\rangle$, with the probability of success*

$$P > 1 - \frac{2^{(\mu_1+\mu_2)d}}{N^{d-1}}\left(1 + 2^{\frac{3d+1}{4}}(1+2d)^{\frac{1}{2}}\right)^{2d+1}.$$

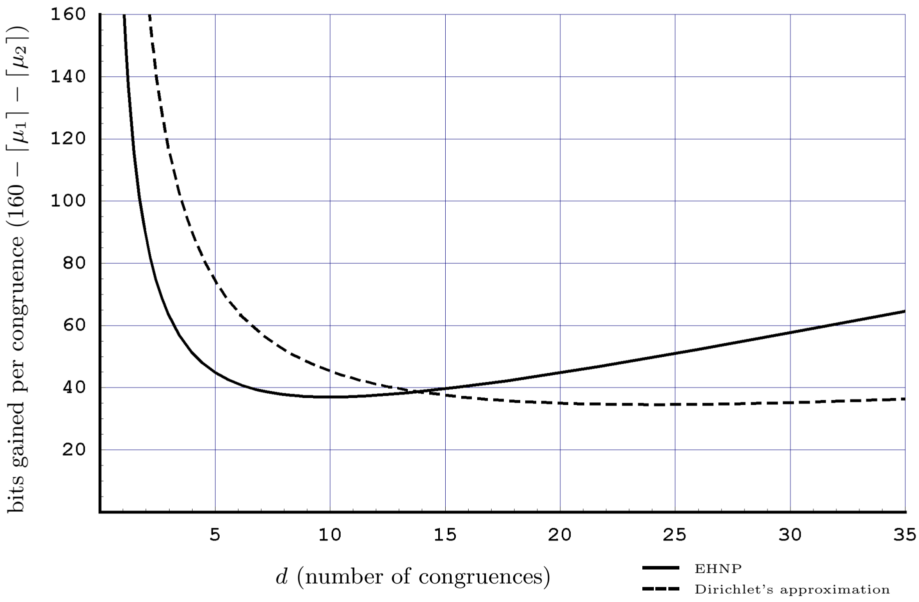


Fig. 1. Dirichlet's approximation vs. EHNP algorithm solving HNP-2H

Proof. The algorithm arises from the solution of EHNP defined and solved in the following section, which HNP-2H is a special case of.

Figure 1 shows a comparison of HNP-2H solving using Dirichlet's approximation and EHNP for 160 bits long modulus N. The graph lines connect the boundary points corresponding to probably solvable combinations of the number of bits gained per congruence and the number of congruences (e.g. signatures of DSA). The EHNP approach is provably preferable in cases with high amount of information available for only few congruences. These are the cases arising, for instance, in fault side channel attacks, where the device may get burnt soon, however, revealing a huge amount of information before being irreversibly damaged. In practice, we may expect a broader preference of EHNP, since the approximation factor will be probably much better than the estimate we used.

4 Extended Hidden Number Problem

Definition 3 (Extended Hidden Number Problem). *Let N be a prime and let x, $x \in \mathbb{Z}_N$, be a particular unknown integer such that*

$$x = \bar{x} + \sum_{j=1}^{m} 2^{\pi_j} x_j, \tag{4}$$

where the integers $\bar{x}$ *and* π_j, $1 \le j \le m$ *are known. The unknown integers* x_j *satisfy* $0 \le x_j < 2^{\nu_j}$, *where* ν_j *are known rational constants* $1 \le j \le m$. *Furthermore, let us be given d congruences*

$$\alpha_i \sum_{j=1}^{m} 2^{\pi_j} x_j + \sum_{j=1}^{l_i} \rho_{i,j} k_{i,j} \equiv \beta_i - \alpha_i \bar{x} \ (\mathbf{mod}\, N), \quad 1 \le i \le d, \tag{5}$$

where α_i, $\alpha_i \not\equiv 0 \ (\mathbf{mod}\, N)$, $1 \le i \le d$, π_j, $1 \le j \le m$, $\rho_{i,j}$, $1 \le i \le d$, $1 \le j \le l_i$ *and* β_i, $1 \le i \le d$ *are known values. The unknown integers* $k_{i,j}$ *satisfy* $0 \le k_{i,j} < 2^{\mu_{i,j}}$, *where* $\mu_{i,j}$ *are known,* $1 \le i \le d, 1 \le j \le l_i$. *We define* $\tau = \sum_{j=1}^{m} \nu_j$, $\xi_i = \sum_{j=1}^{l_i} \mu_{i,j}$, $1 \le i \le d$ *and* $\xi = \sum_{i=1}^{d} \xi_i$.

The Extended Hidden Number Problem (EHNP) is to find (the hidden number) x and its instance is represented by

$$\left(\bar{x}, N, \{\pi_j, \nu_j\}_{j=1}^{m}, \left\{\alpha_i, \{\rho_{i,j}, \mu_{i,j}\}_{j=1}^{l_i}, \beta_i\right\}_{i=1}^{d}\right). \tag{6}$$

Definition 4 (Lattice $\mathcal{L}(\mathcal{I}, \delta)$ and its basis matrix). *For* $\delta > 0$ *and a given instance* $\mathcal{I} = \mathcal{I}_{EHNP}$ *of the EHNP we define* $\mathcal{L}(\mathcal{I}, \delta)$ *as the lattice spanned by the rows of the matrix*

$$\mathbf{B} = \mathbf{B}(\mathcal{I}, \delta) = \begin{pmatrix} N \cdot \mathbf{I}_d & \emptyset & \emptyset \\ \mathbf{A} & \mathbf{X} & \emptyset \\ \begin{matrix} \boldsymbol{\rho}_1^T & & \\ & \ddots & \\ & & \boldsymbol{\rho}_d^T \end{matrix} & \emptyset & \mathbf{K} \end{pmatrix} \in \mathbb{Q}^{D \times D},$$

where we define integers $L = \sum_{i=1}^{d} l_i$ *and* $D = d + m + L$, *vectors*

$$\boldsymbol{\rho}_i = (\rho_{i,1}, \ldots, \rho_{i,l_i}) \in \mathbb{Z}^{l_i}, \quad 1 \le i \le d$$

and matrices

$$\mathbf{A} = (a_{j,i})_{1 \le i \le d, 1 \le j \le m} \in \mathbb{Z}^{m \times d}, \text{ where } a_{j,i} = \alpha_i 2^{\pi_j}$$

$$\mathbf{X} = \mathbf{diag}\left(\frac{\delta}{2^{\nu_1}}, \ldots, \frac{\delta}{2^{\nu_m}}\right) \in \mathbb{Q}^{m \times m}$$

$$\mathbf{K} = \mathbf{diag}\left(\frac{\delta}{2^{\mu_{1,1}}}, \ldots, \frac{\delta}{2^{\mu_{1,l_1}}}, \frac{\delta}{2^{\mu_{2,1}}}, \ldots, \frac{\delta}{2^{\mu_{2,l_2}}}, \ldots, \frac{\delta}{2^{\mu_{d,1}}} \ldots, \frac{\delta}{2^{\mu_{d,l_d}}}\right) \in \mathbb{Q}^{L \times L}.$$

Lemma 1 (Short vectors in $\mathcal{L}$). *Let* $\mathcal{I}$ *be an instance of the EHNP, where* α_i, $1 \le i \le d$ *and* $\rho_{i,j}$, $1 \le i \le d$, $1 \le j \le l_i$ *are uniformly and independently*

Algorithm 1. Finding a solution candidate for EHNP

```
Input:  Instance $\mathcal{I}$ of EHNP
Output: Solution candidate $z \in \mathbb{Z}_N$
1: $\kappa_D \leftarrow \frac{2^{\frac{D}{4}}(m+L)^{\frac{1}{2}}+1}{2}$
2: Choose $\delta$ such that $0 < \kappa_D\delta < 1$
3: $\mathbf{v} \leftarrow \left((\beta_1 - \alpha_1\bar{x}) \,\mathbf{mod}\, N, \ldots, (\beta_d - \alpha_d\bar{x}) \,\mathbf{mod}\, N, \frac{\delta}{2}, \ldots, \frac{\delta}{2}\right)$
4: find $\mathbf{W} \in \mathcal{L} = \mathcal{L}(\mathcal{I}, \delta)$, $\mathbf{W} = (W_1, \ldots, W_D)$ such that $\|\mathbf{W} - \mathbf{v}\| \leq$
   $2^{\frac{D}{4}} \min_{\mathbf{B}\in\mathcal{L}} \|\mathbf{v} - \mathbf{B}\|$          //in polynomial time (§2)
5: for $j = 1$ to $m$ do
6:     $x'_j \leftarrow \frac{W_{d+j}2^{\nu_j}}{\delta}$          //$x'_j \in \mathbb{Z}$
7: end for
8: $z \leftarrow \bar{x} + \sum_{j=1}^{m} 2^{\pi_j} x'_j \,\mathbf{mod}\, N$
9: return $z$
```

distributed on $\langle 1, N-1\rangle$. Let $\delta, \kappa_D \in \mathbb{Q}$ be such that $0 < \delta$, $0 < \kappa_D$ and $\kappa_D\delta < 1$. Then with the probability

$$P > 1 - \frac{(2\kappa_D)^{L+m}\, 2^{\tau+\xi}}{N^d} \tag{7}$$

for each vector $\boldsymbol{\Delta} \in \mathcal{L} = \mathcal{L}(\mathcal{I}, \delta)$ with coordinates $\mathbf{c} = (e_1, \ldots, e_d,\, y_1,\, \ldots,\, y_m,\, t_{1,1}, \ldots, t_{1,l_1},\, \ldots,\, \ldots,\, t_{d,1}, \ldots, t_{d,l_d})$ w.r.t. basis $\mathbf{B} = \mathbf{B}(\mathcal{I}, \delta)$ (i. e. $\boldsymbol{\Delta} = \mathbf{cB}$), satisfying $\|\boldsymbol{\Delta}\|_\infty < \kappa_D\delta$

(i) there exists (a witness index) w, $1 \leq w \leq d$ such that

$$t_{w,j} \equiv 0 \;(\,\mathbf{mod}\, N), \quad 1 \leq j \leq l_w, \tag{8}$$

(ii) $\sum_{j=1}^{m} 2^{\pi_j} y_j \equiv 0 \;(\,\mathbf{mod}\, N)$ holds,
(iii) $\sum_{j=1}^{l_i} \rho_{i,j} t_{i,j} \equiv 0 \;(\,\mathbf{mod}\, N), 1 \leq i \leq d$ holds.

Proof. To be found in Appendix.

Theorem 5 (Correctness of the algorithm solving EHNP). *Let x be the solution of EHNP specified by the instance $\mathcal{I} = \mathcal{I}_{EHNP}$ where N is prime, α_i, $1 \leq i \leq d$ and $\rho_{i,j}$, $1 \leq i \leq d$, $1 \leq j \leq l_i$ are uniformly and independently distributed on $\langle 1, N-1\rangle$. Then with the probability*

$$P > 1 - \frac{(2\kappa_D)^{L+m}\, 2^{\tau+\xi}}{N^d}, \tag{9}$$

where $\kappa_D = \frac{2^{\frac{D}{4}}(m+L)^{\frac{1}{2}}+1}{2}$, Algorithm 1 returns the correct particular solution of instance $\mathcal{I}$.

Proof. Let $\delta \in \mathbb{Q}$ be such that $0 < \kappa_D \delta < 1$. By Definition 3 there exists vector $\mathbf{h} = (c_1, \ldots, c_d,\ x_1, \ldots, x_m,\ k_{1,1}, \ldots, k_{1,l_1},\ \ldots,\ \ldots,\ k_{d,1}, \ldots, k_{d,l_d}) \in \mathbb{Z}^D$ such that

$$c_i N + \alpha_i \sum_{j=1}^{m} 2^{\pi_j} x_j + \sum_{j=1}^{l_i} \rho_{i,j} k_{i,j} = \beta_i - \alpha_i \bar{x}, \qquad 1 \le i \le d, \tag{10}$$

$$0 \le x_j < 2^{\nu_j}, \qquad 1 \le j \le m, \tag{11}$$

$$0 \le k_{i,j} < 2^{\mu_{i,j}}, \qquad 1 \le i \le d,\ 1 \le j \le l_i \tag{12}$$

hold. Let $\mathbf{B} = \mathbf{B}(\mathcal{I}, \delta)$ be the matrix defined in Definition 4 and let

$$\mathbf{H} = \mathbf{hB} \in \mathcal{L},\ \mathcal{L} = \mathcal{L}(\mathcal{I}, \delta),$$
$$\mathbf{v} = ((\beta_1 - \alpha_1 \bar{x}) \mathbf{mod}\, N, \ldots, (\beta_d - \alpha_d \bar{x}) \mathbf{mod}\, N, \frac{\delta}{2}, \ldots, \frac{\delta}{2}) \in \mathbb{Z}^D.$$

Since the vector $\mathbf{H} - \mathbf{v}$ is equal to

$$\Big(0, \ldots, 0, \delta\tfrac{x_1}{2^{\nu_1}} - \tfrac{\delta}{2}, \ldots, \delta\tfrac{x_m}{2^{\nu_m}} - \tfrac{\delta}{2}, \delta\tfrac{k_{1,1}}{2^{\mu_{1,1}}} - \tfrac{\delta}{2}, \ldots, \delta\tfrac{k_{1,l_1}}{2^{\mu_{1,l_1}}} - \tfrac{\delta}{2}, \ldots,$$
$$\ldots, \delta\tfrac{k_{d,1}}{2^{\mu_{d,1}}} - \tfrac{\delta}{2}, \ldots, \delta\tfrac{k_{d,l_d}}{2^{\mu_{d,l_d}}} - \tfrac{\delta}{2}\Big),$$

and the bounds (11), (12) hold, we can write

$$\|\mathbf{v} - \mathbf{H}\|_\infty < \frac{\delta}{2}.$$

A lattice vector $\mathbf{W}$ found in step 4 of the algorithm satisfies

$$\|\mathbf{v} - \mathbf{W}\| \le 2^{\frac{D}{4}} \min_{\mathbf{A} \in \mathcal{L}} \|\mathbf{v} - \mathbf{A}\| \le 2^{\frac{D}{4}} \|\mathbf{v} - \mathbf{H}\| < \frac{2^{\frac{D}{4}} \delta (m+L)^{\frac{1}{2}}}{2}. \tag{13}$$

Let $\boldsymbol{\Delta} = \mathbf{H} - \mathbf{W} \in \mathcal{L}$. Since $\|\mathbf{a}\|_\infty \le \|\mathbf{a}\|$ for all $\mathbf{a} \in \mathbb{Z}^D$, by triangle inequality we have

$$\|\boldsymbol{\Delta}\|_\infty \le \|\mathbf{H} - \mathbf{v}\|_\infty + \|\mathbf{v} - \mathbf{W}\| < \frac{\delta}{2} + \frac{2^{\frac{D}{4}} \delta (m+L)^{\frac{1}{2}}}{2} = \kappa_D \delta. \tag{14}$$

Let $\mathbf{w}$, $\boldsymbol{\gamma}$ be the coordinate vectors of $\mathbf{W}$, $\boldsymbol{\Delta}$, respectively, w.r.t. basis $\mathbf{B}$ and

$$\mathbf{w} = (c'_1, \ldots, c'_d, x'_1, \ldots, x'_m, k'_{1,1}, \ldots, k'_{1,l_1}, \ldots, \ldots, k'_{d,1}, \ldots, k'_{d,l_d})$$
$$\boldsymbol{\gamma} = (e_1, \ldots, e_d, y_1, \ldots, y_m, t_{1,1}, \ldots, t_{1,l_1}, \ldots, \ldots, t_{d,1}, \ldots, t_{d,l_d})$$

and $z = \bar{x} + \sum_{j=1}^{m} 2^{\pi_j} x'_j$ be the candidate returned by Algorithm 1. Since $\mathbf{B}$ is nonsingular, $\boldsymbol{\gamma} = \mathbf{h} - \mathbf{w}$. Then with the probability greater than $1 - \frac{(2\kappa_D)^{L+m} 2^{\tau+\xi}}{N^d}$, guaranteed by Lemma 1, it holds

$$x - z \equiv \left(\bar{x} + \sum_{j=1}^{m} 2^{\pi_j} x_j\right) - \left(\bar{x} + \sum_{j=1}^{m} 2^{\pi_j} x'_j\right) \equiv$$
$$\equiv \sum_{j=1}^{m} 2^{\pi_j} (x_j - x'_j) \equiv \sum_{j=1}^{m} 2^{\pi_j} y_j \equiv 0 \ (\mathbf{mod}\, N).$$

Finally, since $x, z \in \mathbb{Z}_N$, we have $x = z$. □

The distribution conditions in Theorem 5 can be, in a particular cryptanalytic case, further relaxed using techniques like in [14]. For verification of the attack in §5, however, we decided to use an experimental approach instead.

5 Real Scenario - Digital Signature Algorithm

5.1 DSA Key Disclosure Problem

Let us briefly recall the Digital Signature Algorithm (DSA) [1]. The public parameters are specified by triplet (p, q, g), where p and q are prime numbers satisfying $q|p-1$ and $g \in \mathbb{Z}_p^*$ is a generator of the subgroup of order q in $\mathbb{Z}_p^*$. The second revision of [1] requires $2^{1023} < p < 2^{1024}$ and $2^{159} < q < 2^{160}$. The private key $x \in \mathbb{Z}_q$ is chosen uniformly at random from $\mathbb{Z}_q$ and the corresponding public key $y \in \mathbb{Z}_p$ is computed as $y = g^x \,\mathbf{mod}\, p$. The couple (x, y) is called the DSA key pair.

To create a signature (r, s) of a message $m \in \{0, 1\}^*$, the owner of the private key x first generates a random number $k \in \mathbb{Z}_q^*$, which is usually referred to as a nonce (number used once). Then she computes $r = \left(g^k \,\mathbf{mod}\, p\right) \mathbf{mod}\, q$ and $s = k^{-1}(h(m) + xr) \,\mathbf{mod}\, q$, where h is the hash function SHA-1 (see [2]) and $k^{-1}k \equiv 1 \ (\mathbf{mod}\, q)$.

To verify the signature pair (r, s) of a message m, having checked that $0 < r < q$ and $0 < s < q$, one computes $w = s^{-1} \,\mathbf{mod}\, q$, $u_1 = h(m)w \,\mathbf{mod}\, q$, $u_2 = rw \,\mathbf{mod}\, q$ and $v = (g^{u_1} y^{u_2} \,\mathbf{mod}\, p) \,\mathbf{mod}\, q$. She accepts the message signature if and only if $v = r$.

Definition 5 (DSA-KDP problem). *Let (p, q, g) be public DSA parameters and (x, y) DSA key pair. Let*

$$r_i = \left(g^{k_i} \,\mathbf{mod}\, p\right) \mathbf{mod}\, q, \quad 1 \le i \le d \tag{15}$$

$$s_i = k_i^{-1} \left(h(m_i) + x r_i\right) \mathbf{mod}\, q, \quad 1 \le i \le d \tag{16}$$

be known signature pairs of known hashed messages $h(m_i)$. Suppose an additional information about the private key x and the nonces k_i is known, as well, i.e.

$$x = \bar{x} + \sum_{j=1}^{m} 2^{\pi_j} x_j, \ 0 \le x_j < 2^{\nu_j}, \quad 1 \le j \le m \tag{17}$$

$$k_i = \bar{k}_i + \sum_{j=1}^{l_i} 2^{\lambda_{i,j}} k_{i,j}, \ 0 \le k_{i,j} < 2^{\mu_{i,j}}, \quad 1 \le i \le d, 1 \le j \le l_i, \tag{18}$$

where $\bar{x}$, $\{\pi_j, \nu_j\}_{j=1}^{m}$, $\left\{\bar{k}_i, \{\lambda_{i,j}, \mu_{i,j}\}_{j=1}^{l_i}\right\}_{i=1}^{d}$ are known integers satisfying

$$\begin{gathered} \bar{x} \in \mathbb{Z}_q, \ \bar{k}_i \in \mathbb{Z}_q, \quad 1 \le i \le d \\ 2^{\pi_j} \in \mathbb{Z}_q, \quad 1 \le j \le m, \ 2^{\lambda_{i,j}} \in \mathbb{Z}_q, \quad 1 \le i \le d, \ 1 \le j \le l_i \\ 2^{\nu_j} \in \mathbb{Z}_q, \quad 1 \le j \le m, \ 2^{\mu_{i,j}} \in \mathbb{Z}_q, \quad 1 \le i \le d, \ 1 \le j \le l_i \end{gathered}$$

DSA key disclosure problem (DSA-KDP) is to find the private key x and its instance $\mathcal{I}_{DSA}$ is represented by

$$\left(\bar{x}, q, \{r_i, s_i, h(m_i)\}_{i=1}^{d}, \{\pi_j, \nu_j\}_{j=1}^{m}, \left\{\bar{k}_i, \{\lambda_{i,j}, \mu_{i,j}\}_{j=1}^{l_i}\right\}_{i=1}^{d}\right).$$

Lemma 2 (Transition from DSA-KDP to EHNP). *Let x be the particular solution of the DSA-KDP problem specified by the instance $\mathcal{I}_{DSA}$. Let*

$$\begin{aligned} N &= q, \\ \alpha_i &= r_i, \quad 1 \le i \le d \\ \rho_{i,j} &= \left(-s_i 2^{\lambda_{i,j}}\right) \mathbf{mod}\, N, \quad 1 \le i \le d, 1 \le j \le l_i, \\ \beta_i &= \left(s_i \bar{k}_i - h(m_i)\right) \mathbf{mod}\, N, \quad 1 \le i \le d. \end{aligned}$$

Then x can be found as the solution of EHNP specified by the instance

$$\left(\bar{x}, N, \{\pi_j, \nu_j\}_{j=1}^{m}, \left\{\alpha_i, \{\rho_{i,j}, \mu_{i,j}\}_{j=1}^{l_i}, \beta_i\right\}_{i=1}^{d}\right). \tag{19}$$

Proof. The lemma follows directly when (17) and (18) are substituted into (16) in Definition 5. □

5.2 Hyper-threading Technology

In [18], the author explores a side channel hidden in certain processors design employing the Hyper-Threading Technology (HTT). The processors affected are Intel Pentium 4, Mobile Pentium 4 and Xeon.

The size of L1 cache memory in hyper-threaded Pentium 4 is 8 KB (or 16 KB)[1]. It is divided into 32 cache address classes consisting of 4 (or 8) cache lines of 64 bytes. When a memory line is requested by a process, the processor first checks whether the line is already loaded in L1 cache in the corresponding cache class. In case it is, we say a cache hit occurs, contrary to a cache miss when the line has to be loaded to L1 cache. Therefore, a cache miss takes a much longer time than a cache hit and that can be recognized by the process.

A hyper-threaded Pentium 4 multiplexes two independent instruction streams over one set of execution resources creating two virtual processing cores that share certain physical resources, namely the L1 cache memory. This is an architectural design that leaves the Confinement Problem [5] unsolved leading to a vital side channel. To the operating system, this platform appears as two logical CPUs, thus it can schedule two processes to be run at the same time. One process cannot read the data of the other process, however, it can determine whether the process running on the second virtual core used certain line of the L1 cache or not by measuring the amount of time it takes to repeatedly read several data blocks from the same cache address class.

In this way, we get a side information which can be used to discriminate between two different operations performed by the process being spied [17], [18].

[1] Depending on actual type.

When the two operations are different enough with respect to the memory access they induce, we can easily identify them basing on a different "footprint" left in the access time measurements (cf. Figure 2).

In [18], this side information was used to break an implementation of RSA scheme. Here, we show that by using the EHNP approach described before, we can break certain implementation of DSA algorithm, as well. In practice, this attack threatens, for instance, applications like SSH [19], when the server runs on an unsecured HTT platform and uses DSA for the server authentication. When the attacker logs on the server, she can run the spy process on one processor core, while she opens another SSH session with the server, hoping that it will run on the second core. In this way, she gains the side information about DSA signature computation when the server authenticates itself for the newly opened connection. Collecting several such measurements, she can get enough information to be able to solve the associated EHNP. From here, she gets the server's private key allowing her to impersonate the server.

5.3 Sliding Window Exponentiation

An OpenSSL-based SSH server uses the sliding window (SW) exponentiation algorithm (cf. Algorithm 2)[2] in the process of DSA authentication when computing $r = g^k \, \mathbf{mod}\, p$. Two operations to be discriminated on the HTT platform by the aforesaid technique are squaring (S) and multiplication (M). Being able to identify the SW algorithm execution, the attacker obtains a sequence $\mathcal{S} \in \{S, M\}^*$.

To convert $\mathcal{S}$ to an information suitable for EHNP, one sets $k' = 0$ as the first approximation of the nonce k. Then, she takes the next operation from $\mathcal{S}$ and multiplies k' by 2 if the operation is S or adds 1 and adds a new "hole" for M. Finally, the holes of zero length are filtered out from the output sequence. This procedure, described by Algorithm 3, outputs the decomposition $k = k' + \sum_{j=1}^{l} 2^{\lambda_j} k_j$, $0 \leq k_j < 2^{\mu_j}$. On average case, the algorithm provides us with 78 known bits separated by 31 holes for the exponent size of 160 bits with the sliding window length set to 4 (to match its definition in OpenSSL 0.9.7e).

In Figure 2, we can see the execution of SW algorithm from the spy process viewpoint. The rows, each representing one of 32 memory classes in L1 cache, change color from white standing for a very fast read operation, to black for slow data reloads. To emphasize the different footprints, the extra top row displays the average gray level for the selected cache classes (in our case 24th, 25th and 26th class). This row allows us to identify squaring and multiplication operations easily.

5.4 Practical Experiments

Algorithm 1 was implemented in C++ employing Shoup's NTL library [20]. Several experiments were run for different number of signatures d. Each experiment consisted of 10 instances of DSA-KDP with random public parameters,

[2] Valid for the versions up to 0.9.7g. As from version 0.9.7h, OpenSSL uses fixed window modular exponentiation by default for RSA, DSA, and DH private key operations to prevent cache timing attacks.

Algorithm 2. Sliding window (SW) exponentiation; s is the SW length

```
Input: $g$, $e = (e_t e_{t-1} \dots e_0)_2$, $e_t = 1$, $s \geq 1$
Output: $g^e$
 1: $g_1 \leftarrow g$, $g_2 \leftarrow g^2$
 2: for $i = 1$ to $2^{s-1} - 1$ do
 3:     $g_{2i+1} \leftarrow g_{2i-1} g_2$
 4: end for
 5: $A \leftarrow 1$, $i \leftarrow t$
 6: while $i \geq 0$ do
 7:     if $e_i = 0$ then
 8:         $A \leftarrow A^2$                                    //squaring
 9:         $i \leftarrow i - 1$
10:     else
11:         find longest string $(e_i e_{i-1} \dots e_l)$ such that $i - l + 1 \leq s$ and $e_l = 1$
12:         $A \leftarrow A^{2^{i-l+1}} g_{(e_i e_{i-1} \dots e_l)_2}$         //$i - l + 1$ squarings, 1 multiplication
13:         $i \leftarrow l - 1$
14:     end if
15: end while
16: return $A$
```

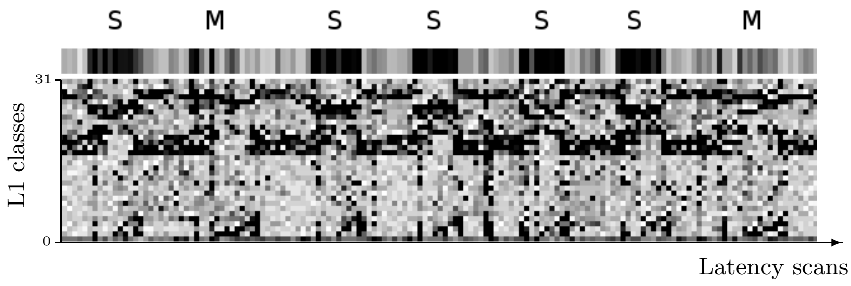


Fig. 2. SW exponentiation observed through the side channel of L1 cache

random key pair and the signature pairs for d random messages. The results of the experiments are displayed in Figures 3 and 4. The computing platform employed was running GNU/Linux Debian on AMD Opteron 844. We used the side channel emulation in these computations. Its real existence and usability was successfully verified by technical experiments with an SSH server powered by OpenSSL 0.9.7e on an unprotected Pentium 4 HTT platform, as well.

The bases were reduced by LLL reduction with `delta = 0.99` (`LLL_XD()`, marked “LLL”). If such reduction did not lead to the key disclosure, stronger Block Korkin-Zolotarev reduction with Givens rotations (`G_BKZ_XD()`, marked “BKZ”) was employed with `delta = 0.99`, `BlockSize = 40` and `Prune = 15`.

We ran several experiments with random DSA-KDP instances with the size of DSA prime q set to 256 bits (which is the highest size allowed by the draft

Algorithm 3. Conversion of sequence from $\{S, M\}^*$ to the decomposition of k; s is the SW length; (N is the upper bound on k)

```
Input: $\mathcal{S} \in \{S, M\}^*$, $s \geq 1$, ($N > 1$)
Output: $\bar{k}, l, \{\lambda_j, \mu_j\}_{j=1}^{l}$
 1: $\bar{k} \leftarrow 0$, $L \leftarrow 0$, $\Lambda_0 \leftarrow 1$, $w_0 \leftarrow s - 1$      //L, Λ, and w are internal only
 2: $A \leftarrow \mathrm{first}(\mathcal{S})$
 3: repeat
 4:    if $A = S$ then
 5:       $\bar{k} \leftarrow 2\bar{k}$
 6:       for $j = 0$ to $L$ do
 7:          $\Lambda_j \leftarrow \Lambda_j + 1$                          //shift existing holes
 8:       end for
 9:    else
10:       $\bar{k} \leftarrow \bar{k} + 1$
11:       $L \leftarrow L + 1$, $\Lambda_L \leftarrow 1$
12:       $w_L \leftarrow \min(s - 1, \Lambda_{L-1} + w_{L-1} - s - 1)$    //new, possibly empty, hole
13:    end if
14: until $A \leftarrow \mathrm{next}(\mathcal{S})$
15: if $N$ is defined and $\Lambda_1 + w_1 > \lceil \log_2 N \rceil$ then
16:    $w_1 = \lceil \log_2 N \rceil - \Lambda_1$                          //adjust the first hole
17: end if
18: $l \leftarrow 0$
19: for $j = 1$ to $L$ do
20:    if $w_j > 0$ then
21:       $l \leftarrow l + 1$, $\lambda_l \leftarrow \Lambda_j$, $\mu_l \leftarrow w_j$       //keep the nonempty holes
22:    end if
23: end for
```

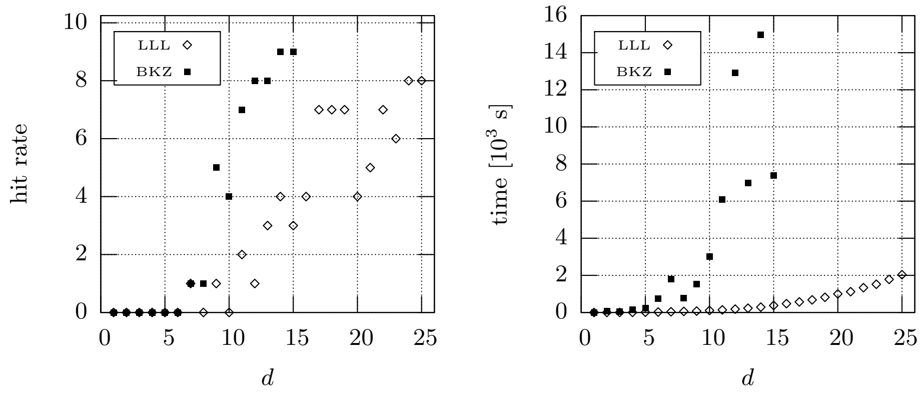


Fig. 3. The hit rate and the average duration of EHNP algorithm solving 10 random instances of DSA-KDP, each derived from a simulated side channel leakages during the signing operation with $\lceil \log_2 q \rceil = 160$

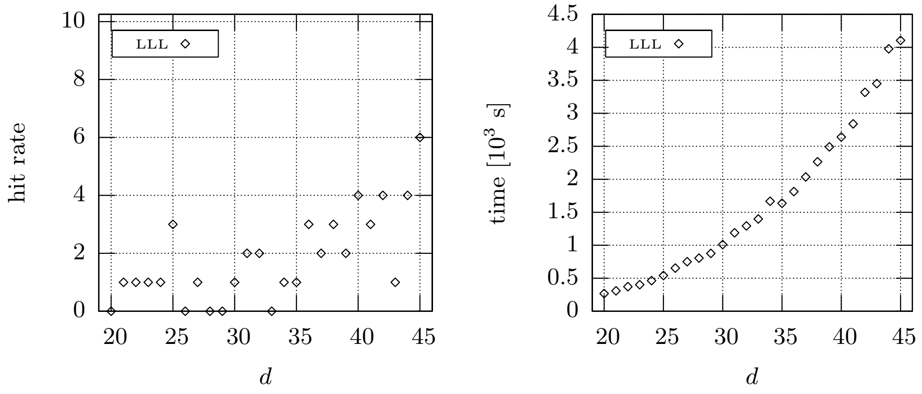


Fig. 4. Hit rate and the average duration of EHNP algorithm solving 10 random instances of DSA-KDP, each derived from a simulated side channel leakages during the signing operation with $\lceil \log_2 q \rceil = 256$. Blocks of known bits shorter than 5 in length are ignored (to reduce running time)

of Third revision of FIPS 186). The dimension of lattices associated is higher resulting in longer running time, however, as shown in Figure 4, the private key can be extracted for these instances, as well.

6 Conclusion

The Hidden Number Problem (HNP) was originally presented as a tool for proving cryptographic security of Diffie-Hellman and related schemes [7]. It was then shown in [13] and [14] that HNP can be used for solving cryptanalytic problems arising around DSA, as well. However, it still retained certain properties that limited its suitability for real cryptanalytic attacks. In this article, we showed how to overcome these limitations by formulation of the Extended Hidden Number Problem (EHNP) that covers significantly broader area of practical cases. Algorithm for solving EHNP together with its formal analysis were presented. We employed EHNP to develop a practically feasible side channel attack on the OpenSSL implementation of DSA. This part of the paper is of independent merit showing an inherent insecurity of the Pentium 4 HTT processor platform for applications like SSH. For instance, having observed only 6 authentications to the OpenSSL-powered SSH server, an attacker was able to disclose the server's private key.

Acknowledgment

The authors would like to thank Colin Percival for kindly providing them the source code for his experiments and the anonymous referees for their valuable comments. The first author thanks Jozef Juríček for helpful discussions covering several topics in probability theory. The second author appreciates eBanka a.s. supporting these research activities.

References

1. Digital signature standard. National Institute of Standards and Technology, Washington (Note: Federal Information Processing Standard 186-2) (2000), URL: http://csrc.nist.gov/publications/fips/
2. Secure hash standard. National Institute of Standards and Technology, Washington (Note: Federal Information Processing Standard 180-2) (2002), URL: http://csrc.nist.gov/publications/fips/
3. Babai, L.: On Lovász' lattice reduction and the nearest lattice point problem. In: Mehlhorn, K. (ed.) STACS 85. LNCS, vol. 182, pp. 13–20. Springer, Heidelberg (1984)
4. Bellare, M., Goldwasser, S., Micciancio, D.: "Pseudo-Random" number generation within cryptographic algorithms: The DDS case. In: Kaliski Jr., B.S. (ed.) CRYPTO 1997. LNCS, vol. 1294, pp. 277–291. Springer, Heidelberg (1997)
5. Bishop, M.: Computer Security: Art and Science. Addison-Wesley, Reading (2003)
6. Boneh, D., Halevi, S., Howgrave-Graham, N.: The modular inversion hidden number problem. In: Boyd, C. (ed.) ASIACRYPT 2001. LNCS, vol. 2248, pp. 36–51. Springer, Heidelberg (2001)
7. Boneh, D., Venkatesan, R.: Hardness of computing the most significant bits of secret keys in Diffie-Hellman and related schemes. In: Koblitz, N. (ed.) CRYPTO 1996. LNCS, vol. 1109, pp. 129–142. Springer, Heidelberg (1996)
8. Frieze, A.M., Hastad, J., Kannan, R., Lagarias, J.C., Shamir, A.: Reconstructing truncated integer variables satisfying linear congruences. SIAM Journal of Computing 17(2), 262–280 (1988)
9. Groetschel, M., Lovász, L., Schrijver, A.: Geometric Algorithms and Combinatorial Optimization, 2nd edn. Springer, Heidelberg (1993)
10. Howgrave-Graham, N.-A., Smart, N.P.: Lattice attacks on digital signature schemes. Design, Codes and Cryptography 23, 283–290 (2001)
11. Intel Corporation. Intel(R) Pentium(R) 4 Processor supporting Hyper-Threading Technology. URL: http://www.intel.com/products/processor/pentium4/index.htm
12. Lenstra, A.K., Lenstra Jr., H.W., Lovász, L.: Factoring polynomials with rational coefficients. Mathematische Ann. 261, 513–534 (1982)
13. Nguyen, P.Q.: The dark side of the hidden number problem: Lattice attacks on DSA. In: Proc. of the Workshop on Cryptography and Computational Number Theory (CCNT '99), Basel, CH, pp. 321–330. Birkhäuser (2001)
14. Nguyen, P.Q., Shparlinski, I.: The insecurity of the digital signature algorithm with partially known nonces. J. Cryptology 15(3), 151–176 (2002)
15. Nguyen, P.Q., Stehlé, D.: Floating-point LLL revisited. In: Cramer, R.J.F. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 215–233. Springer, Heidelberg (2005)
16. Nguyen, P.Q., Stern, J.: The two faces of lattices in cryptology. In: Silverman, J.H. (ed.) CaLC 2001. LNCS, vol. 2146, pp. 146–180. Springer, Heidelberg (2001)
17. Osvik, D.A., Shamir, A., Tromer, E.: Cache attacks and countermeasures: The case of AES. In: Pointcheval, D. (ed.) CT-RSA 2006. LNCS, vol. 3860, pp. 1–20. Springer, Heidelberg (2006)

18. Percival, C.: Cache missing for fun and profit (2005), URL: http://www.daemonology.net/papers/htt.pdf
19. OpenBSD project members. OpenSSH Suite. URL: http://www.openssh.com/
20. Shoup, V.: NTL: A Library for doing Number Theory. URL: http://www.shoup.net/ntl/
21. Shparlinski, I., Winterhof, A.: A hidden number problem in small subgroups. Mathematics of Computation 74(252), 2073–2080 (2005)

Appendix

Lemma 3 (Short solutions). *Given prime N and $s \in \mathbb{Z}_N$, let $\rho_1, \ldots, \rho_l$ be uniformly and independently distributed on $\mathbb{Z}_N$. Then the probability that the congruence*

$$\sum_{j=1}^{l} \rho_j x_j \equiv s \ (\mathbf{mod}\, N) \tag{20}$$

has a "short" non-trivial solution $(t_1, \ldots, t_l) \in (\mathbb{Z}_N)^l$ satisfying $|t_j|_N < T_j$, $1 \le j \le l$ is lower than

$$\frac{2^l \prod_{j=1}^{l} T_j}{N}. \tag{21}$$

Proof. Let $\mathbf{t} = (t_1, \ldots, t_l)$ be a non-zero l-tuple in $(\mathbb{Z}_N)^l$ with $t_k \neq 0$. There exist exactly N^{l-1} l-tuples

$$(\rho_1, \ldots, \rho_l) = \left(\rho_1, \ldots, \rho_{k-1}, (t_k)^{-1}\left(s - \sum_{j=1, j\neq k}^{l} \rho_j t_j\right) \mathbf{mod}\, N, \rho_{k+1}, \ldots, \rho_l\right)$$

such that $\mathbf{t}$ is a solution of (20). Consequently, there exist no more than

$$N^{l-1}\left(\left(\prod_{j=1}^{l}(2T_j - 1)\right) - 1\right) < N^{l-1} 2^l \prod_{j=1}^{l} T_j$$

l-tuples $(\rho_1, \ldots, \rho_l)$ such that "short" non-trivial solution of (20) exists. Since N^l is total number of all l-tuples on $\mathbb{Z}_N$, the lemma follows. □

Proof (of Lemma 1 on Short vectors in $\mathcal{L}$). Let $\boldsymbol{\Delta} \in \mathcal{L}$ be such that $\|\boldsymbol{\Delta}\|_\infty < \kappa_D \delta < 1$. Then

$$\left| e_i N + \alpha_i \sum_{j=1}^{d} 2^{\pi_j} y_j + \sum_{j=1}^{l_i} \rho_{i,j} t_{i,j} \right| = |\Delta_i| < 1, \quad 1 \le i \le d \tag{22}$$

$$\left| \frac{\delta}{2^{\nu_j}} y_j \right| = |\Delta_{d+j}| < \kappa_D \delta, \ 1 \le j \le m \tag{23}$$

$$\left| \frac{\delta}{2^{\mu_{i,j}}} t_{i,j} \right| = \left| \Delta_{d+m+j+\sum_{u=1}^{i-1} l_u} \right| < \kappa_D \delta, \ \begin{matrix} 1 \le i \le d \\ 1 \le j \le l_i \end{matrix} \tag{24}$$

respectively implying

$$\begin{aligned}
\left|\alpha_i \sum_{j=1}^{d} 2^{\pi_j} y_j + \sum_{j=1}^{l_i} \rho_{i,j} t_{i,j}\right|_N &= \quad 0, && 1 \le i \le d && (25)\\
|y_j| &< \ \kappa_D 2^{\nu_j}, && 1 \le j \le m && (26)\\
|t_{i,j}| &< \kappa_D 2^{\mu_{i,j}}, && 1 \le i \le d, 1 \le j \le l_i, && (27)
\end{aligned}$$

since the expression on the left-hand side of (22) is an integer. Furthermore, (25) is equivalent to the congruence

$$\sum_{j=1}^{l_i} \rho_{i,j} t_{i,j} \equiv -\alpha_i \sum_{j=1}^{d} 2^{\pi_j} y_j \ (\mathbf{mod}\, N), \quad 1 \le i \le d. \tag{28}$$

To prove (i), we have to show the probability P_F that for all i, $1 \le i \le d$ there exists j, $1 \le j \le l_i$ such that $t_{i,j} \not\equiv 0 (\mathbf{mod}\, N)$ is bounded above as $P_F < \frac{(2\kappa_D)^{L+m} 2^{\tau+\xi}}{N^d}$. Regarding the algorithmic viewpoint of the whole paper, it is worth emphasizing the following probability elaboration is focused on the event that an "unwanted" vector does exist in the lattice at all, rather than investigating the properties of a particular vector chosen. The algorithmic interpretation is then that, obviously, the particular vector computed cannot have the properties that no such vector in the lattice $\mathcal{L}(\mathcal{I}, \delta)$ has.

Fix an m-tuple $(y_1, \ldots, y_m) \in \mathbb{Z}^m$ and define $R_i = -\alpha_i \sum_{j=1}^{d} 2^{\pi_j} y_j \,\mathbf{mod}\, N$. The substitution to congruence (28) gives

$$\sum_{j=1}^{l_i} \rho_{i,j} t_{i,j} \equiv R_i \ (\mathbf{mod}\, N), \quad 1 \le i \le d. \tag{29}$$

Lemma 3 states non-trivial solution of (29) satisfying the bounds (27) exists with the probability

$$p_i(y_1, \ldots, y_m) < \frac{2^{l_i} \prod_{j=1}^{l_i} \kappa_D 2^{\mu_{i,j}}}{N} = \frac{(2\kappa_D)^{l_i} 2^{\xi_i}}{N}. \tag{30}$$

For a fixed m-tuple $(y_1, \ldots, y_m)$, the probability that (29) and (27) can be non-trivially satisfied for all i, $1 \le i \le d$ is

$$p(y_1, \ldots, y_m) \le \prod_{i=1}^{d} p_i(y_1, \ldots, y_m) < \prod_{i=1}^{d} \frac{(2\kappa_D)^{l_i} 2^{\xi_i}}{N} = \frac{(2\kappa_D)^{L} 2^{\xi}}{N^d}. \tag{31}$$

There is no more than $\prod_{j=1}^{m} 2\kappa_D 2^{\nu_j} = (2\kappa_D)^m 2^\tau$ m-tuples $(y_1, \ldots, y_m)$ that satisfy (26), therefore

$$P_F \le \sum_{\substack{\mathbf{y} = (y_1, \ldots, y_m) \\ \mathbf{y} \text{ satisfies (26)}}} p(y_1, \ldots, y_m) < \frac{(2\kappa_D)^{L+m} 2^{\tau+\xi}}{N^d}. \tag{32}$$

To prove the congruence (ii) holds, it suffices to substitute $t_{w,j} \equiv 0 \ (\mathbf{mod}\, N)$, $1 \leq j \leq l_w$ from (i) to (28), i.e.

$$\sum_{j=1}^{m} 2^{\pi_j} y_j \equiv -(\alpha_w)^{-1} \sum_{j=1}^{l_w} \rho_{w,j} t_{w,j} \equiv 0 \ (\mathbf{mod}\, N), \tag{33}$$

since $(\alpha_w)^{-1} \,\mathbf{mod}\, N$ exists, because $\alpha_w \not\equiv 0 \,\mathbf{mod}\, N$ and N is a prime.

Finally, substituting (ii) to the congruence (28), i.e.

$$\sum_{j=1}^{l_i} \rho_{i,j} t_{i,j} \equiv -\alpha_i \sum_{j=1}^{m} 2^{\pi_j} y_j \equiv 0 \ (\mathbf{mod}\, N), \tag{34}$$

finishes the proof of (iii). □

Changing the Odds Against Masked Logic

Kris Tiri[1] and Patrick Schaumont[2]

[1] Trusted Platform Laboratory, Intel Corporation, USA
kris.tiri@intel.com
[2] ECE Department, Virginia Tech, USA
schaum@vt.edu

Abstract. Random switching logic (RSL) has been proposed as an efficient countermeasure to mitigate power analysis. The logic style equalizes the output transition probabilities using a random mask-bit. This manuscript, however, will show a successful attack against RSL. The single mask-bit can only add one bit of entropy to the information content of the overall power consumption variations and can very easily be deduced from the power consumption. Once the mask-bit is known, the a posteriori probabilities of the output transitions are not equal anymore and a power analysis can be mounted. A threshold filter suffices to remove the additional bit of information.

1 Introduction

Side-channel attacks (SCAs) do not attack the mathematical properties of an encryption algorithm. Instead, they use information that is leaked by the device on which the algorithm has been implemented. Variations in power consumption, but also in time delay and electromagnetic radiation, have all successfully been exploited. In [7] for instance, a power analysis extracts the full 128-bit key of an ASIC AES implementation in less then three minutes.

A SCA works as follows: it compares an estimation of the side-channel leakage with a measurement of the side-channel leakage. In a power-based SCA, measured power traces are compared with power consumption estimations. The correct key is found by identifying the best match between the measurements and the possible estimations. Furthermore, by limiting the side-channel leakage estimation to only a small piece of the algorithm, the computational complexity is reduced compared to a brute-force attack. A single AES secret key byte can be found by estimating the power consumption of only a single state register byte.

Countless SCA mitigations have been put forward. In case of a power analysis, they range from decoupling the power supply or adding noise generators to masking data bearing signals or using custom logic cells. In [5],[6], Suzuki et al. combine the two ideas of masking data bearing signals and using custom logic cells into random switching logic (RSL). RSL unites the advantage of the former of being a theoretically proven countermeasure and the advantage of the latter of being an algorithmic independent countermeasure.

© Springer-Verlag Berlin Heidelberg 2007

Yet as shown in [2],[3], theoretically proven mitigations do not always hold in practice. In this manuscript, we will successfully attack random switching logic. In fact, we will show that RSL only improves the power analysis resistance by a factor of two.

Masking decorrelates the data from the power consumption. It equalizes the transition probabilities of the data bearing signals. If all signal transitions, i.e. 0 to 0, 0 to 1, 1 to 0 and 1 to 1, are equally likely and independent of the state of the circuit, a power analysis will be unsuccessful. In RSL, each and every signal is masked by xor-ing the output of all logic gates with a random mask-bit and consequently the output transition, and thus the power consumption, of the logic gate is independent of the state of the gate.

Masking is only effective if the mask-bit itself remains secret. If the value of this bit can somehow be derived from the measurements, the output transitions are not equally likely anymore. Therefore, knowledge of the mask-bit value will re-enable a normal power analysis.

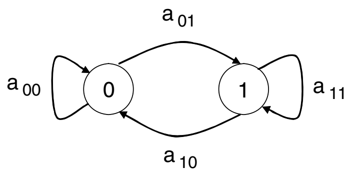
A single mask-bit can only impact the power consumption in a binary fashion. Even though all signal transitions are equally likely, the transition probability of a gate output is higher if the mask-bit changes. Indeed, the mask-bit can be seen as a data signal distributed to all gates. A change of its value will have an effect on all of the gates. It will result in a proportionally larger number of output transitions than when its value would remain constant. By filtering out the high power peaks, caused by these additional output transitions, we keep the events in which the mask-bit remains constant, thus in which the masking operation was actually absent.

The main contributions of this paper are that we point out that a single mask-bit only adds one bit of entropy, which is not sufficient to protect against a power analysis and that we show how to successfully attack random switching logic. Additionally, compared with [3], we show that masking at the gate level can also be attacked when glitches are not present.

The remainder of this paper is organized as follows. The next section presents the information theory model of masking. It first introduces masking, probability and entropy and then applies it to RSL. In section 3, an experiment is setup in which, based of the findings of section 2, a test circuit implemented in RSL is successfully attacked. Finally a conclusion will be formulated.

2 Changing the Odds Using a Posteriori Probabilities

The power consumption of a CMOS circuit is dependent on circuit and architecture characteristics such as capacitance, supply voltage, leakage current and clock frequency. In addition, it is also dependent on signal activity. In the following, we will focus on the impact of this signal activity on information leakage. We will define some information theory concepts, and in particular analyze the entropy of masked signals. Using these concepts we will describe our proposed attack on RSL, which is discussed in the second subsection.



$$A_{00} = a_0.a_{00}$$
$$A_{01} = a_0.a_{01}$$
$$A_{10} = a_1.a_{10}$$
$$A_{11} = a_1.a_{11}$$

Fig. 1. Markov Model for signal a

2.1 Conditional Transition Probability and Entropy of Random Digital Signals

The dynamic power consumption of digital circuits is characterized in terms of the switching activity of the signals in the design. One defines the *activity factor* p_t of a signal a as the probability for a power-consuming transition per clock cycle of that signal [1]. A clock signal has $p_t = 1$. In static CMOS logic, glitch-free data signals have a p_t smaller than 0.5. Often the activity factor is expressed as a percentage, and typical digital circuits show activity factors of 5% to 15% [9]. This means that a signal will show a power-consuming transition (i.e. 0→1 for CMOS) during 5% to 15% of the clock cycles.

For a signal a, we denote the probability of a zero as a_0 and of a one as a_1.

$$P(a=1) = a_1, \quad P(a=0) = a_0 = 1 - a_1 \tag{1}$$

We now establish a relation between p_t and the probability characteristics of a. We denote the absolute probability that a will make a transition from 0 to 1 as A_{01}. By our choice of p_t, it must be that $A_{01} = p_t$. We can now derive other relevant probabilities based on the probability a_0 and the activity factor p_t.

We will make use of a Markov model as illustrated in figure 1. In a Markov model, the random signal a is characterized in terms of its transition probabilities over a number of clock cycles. For each possible transition there is a corresponding probability defined as a_{ij}. For example, a_{01} is the probability that a becomes 1 under the condition that one clock cycle earlier it has the value 0. An important observation is that the conditional transition probabilities a_{ij} are bigger than absolute transition probabilities A_{ij}. In other words, the knowledge of the value of a influences our knowledge on the transitions made by a. This leads to the concept of *a posteriori probability*: the transition probability when the value of a is known.

The long-term probabilities of the signal being zero (a_0) or one (a_1) are determined by the Markov chain in (2). This leads to the conclusion that random signals have an equal amount of up-going and down-going edges: $A_{01} = A_{10} = p_t$.

$$\left.\begin{array}{l} \begin{bmatrix} a_{11} & a_{01} \\ a_{10} & a_{00} \end{bmatrix} \begin{bmatrix} a_1 \\ a_0 \end{bmatrix} = \begin{bmatrix} a_1 \\ a_0 \end{bmatrix} \\ \Sigma a_i = \Sigma a_{1i} = \Sigma a_{0i} = 1 \end{array}\right\} \Rightarrow a_0.a_{01} = a_1.a_{10} \Rightarrow A_{01} = A_{10} \tag{2}$$

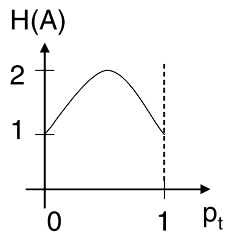


Fig. 2. Transition Entropy

Entropy expresses the information content of a signal. For an event E with probability q, the entropy is equal to $H(E)$ = - q . $\log_2(q)$. A random signal thus has a single bit of entropy:

$$H(a) = H(a_0) + H(a_1) = -½.\log_2(½) - ½.\log_2(½) = 1 \tag{3}$$

The entropy of the transitions of a is:

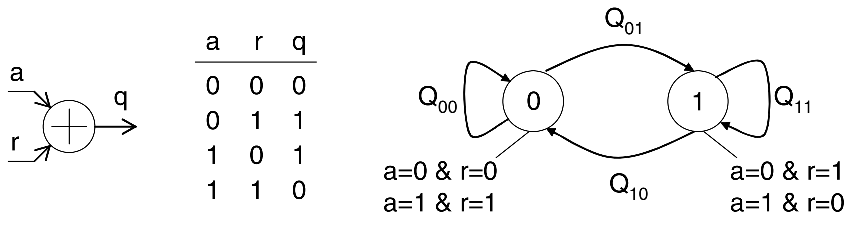
$$\begin{aligned} H(A) &= H(A_{00}) + H(A_{01}) + H(A_{10}) + H(A_{11}) \\ H(A) &= -p_t \log_2(p_t) - (a_0 - p_t)\log_2(a_0 - p_t) \\ &\quad - (a_1 - p_t)\log_2(a_1 - p_t) - p_t \log_2(p_t) \end{aligned} \tag{4}$$

When $a_0 = a_1 = 0.5$, the entropy of the transitions is between 1 and 2 bit, depending on the value of p_t.

$$H(A)\Big|_{a_0=a_1=½} = -p_t(\log_2(p_t) - 1) - (1 - p_t)(\log_2(1 - p_t) - 1) \tag{5}$$

For $p_t = 0.5$, $H(A)$ reaches a maximum of 2 bits, as shown in figure 2. For other values in $0 < p_t < 1$, $H(A)$ is never smaller than 1 bit, even though $H(A)$ is derived from a single-bit random signal. Indeed, $H(A)$ is the result of observing two bits from a random stream. When these bits are uncorrelated (i.e. when the $p_t = 0.5$), we receive two bits of information per observation.

We now examine the transfer of information through logic gates using the above concepts. Consider first a simple xor gate, with two signals at the input,



a	r	q
0	0	0
0	1	1
1	0	1
1	1	0

Fig. 3. Markov chain for an xor gate with inputs r and a

one called r and another one called a. We will characterize the properties of the output signal q in terms of these two input signals.

The transition probabilities of q can be expressed in terms of the transition probabilities of r and a. For example, the transition probability Q_{01} requires a and r to change from equal to different value. There are four combinations that have this effect, which results in Q_{01} having four terms.

$$\begin{aligned}
Q_{01,xor} &= A_{00}R_{01} + A_{11}R_{10} + A_{10}R_{11} + A_{01}R_{00} \\
Q_{11,xor} &= A_{00}R_{11} + A_{11}R_{00} + A_{10}R_{01} + A_{01}R_{10} \\
Q_{10,xor} &= A_{00}R_{01} + A_{11}R_{01} + A_{10}R_{00} + A_{01}R_{11} \\
Q_{00,xor} &= A_{00}R_{00} + A_{11}R_{11} + A_{10}R_{10} + A_{01}R_{01}
\end{aligned} \tag{6}$$

If r is an uncorrelated random signal (i.e. $R_{01} = R_{10} = R_{11} = R_{00} = 0.25$), then q will be an uncorrelated random signal as well. As a result, the entropy $H(Q_{xor})$ will be two bit, the same as $H(R)$. Only the xor gate has this property; an xor gate does not loose information and allows the signal a to be restored if signal r is known. In contrast, and and or gates destroy information, and their $H(Q)$ will be less than 2 bit. For example, assume an and gate with two input signals a and r, each with activity 50%, then it can be shown that

$$Q_{00,and} = 9/16, \quad Q_{01,and} = Q_{10,and} = 3/16, \quad Q_{11,and} = 1/16 \tag{7}$$

which leads to an entropy of $H(Q_{and}) = 1.66$ bit.

In the next section, we will apply the ideas of conditional probabilities and entropy to the observation of the power consumption of RSL gates. This will show that an apparently random signal can still have non-random a posteriori probabilities.

2.2 Random Switching Logic

The RSL nor and nand gates are defined as follows [5]:

$$\begin{aligned}
nor_{rsl} &: z = \overline{\bar{e} + x.y + (x+y).\bar{r}} \\
nand_{rsl} &: z = \overline{\bar{e} + x.y + (x+y).r}
\end{aligned}$$

$$with \begin{cases} x = a \oplus r, \quad y = b \oplus r, \quad z = q \oplus r \\ q\big|_{nor_{rsl}} = \overline{a+b}, \quad q\big|_{nand_{rsl}} = \overline{a.b} \end{cases} \tag{8}$$

Signal r is the random mask-bit, which equalizes the transition probability according to formula 6. Signal e is an enable signal, which suppresses transient hazards. It prevents glitches which have been shown to make a power analysis on masked gates possible [2]. The signal only enables the gate when input signals x, y and r are stable. For this purpose, signal e must meet stringent requirements such that for each gate its arrival time is later than the arrival times of the output of already enabled gates. For the rest of this manuscript, we will assume that the enable signal e is one and that glitches do not occur. Please note that for the experimental results, a cycle accurate simulator is used which does not simulate glitches.

In a design implemented with RSL, only the global input signals are explicitly masked. The internal nets are masked because of the RSL gates. A gate expects masked inputs and produces a masked output, which serves as the masked input of the next gate. Note that to implement this functionality, r is still distributed to and used by all gates.

The signal r modifies the functionality of the RSL gates as follows. When r equals zero, we can derive that:

$$\begin{aligned} nor_{rsl}\big|_{r=0} &= \overline{x+y} \\ nand_{rsl}\big|_{r=0} &= \overline{x.y} \end{aligned} \tag{9}$$

and when r equals one:

$$\begin{aligned} nor_{rsl}\big|_{r=1} &= \overline{x.y} \\ nand_{rsl}\big|_{r=1} &= \overline{x+y} \end{aligned} \tag{10}$$

Whenever the mask-bit value changes, each gate that previously functioned as a nand gate, modifies its functionality to a nor gate and vice versa. A design implemented with RSL will thus switch between two dual configurations, which will require energy in addition to the energy for calculating the design's output.

Table 1 presents the transition probabilities Z_{ij} of an RSL nand gate in function of the random mask-bit transition. The table has been calculated with the unmasked signals a and b having a typical activity factor of 10%. The summation per transition event of z shows that all transitions are equally likely (i.e. $Z_{01} = Z_{10} = Z_{11} = Z_{00} = 0.25$). This is the basis for the power analysis resistance of RSL.

The table, however, also shows that the a prosteriori probabilities are not equally likely. For instance, if the mask-bit remains at one, with high probability the output z will remain at zero. Indeed, when r is one, the RSL nand gate functions as a nor gate, for which it is sufficient that one input is one to have a zero output.

Table 1 shows that when the mask-bit r remains constant, it is very likely that the output z remains constant ($Z_{11} + Z_{00} >> Z_{01} + Z_{10}$ for r_{0-0} and r_{1-1}) while when the mask-bit changes, it is very likely that the output changes ($Z_{01} + Z_{10} >> Z_{11} + Z_{00}$ for r_{0-1} and r_{1-0}). There will thus be a large difference in power consumption between the two events. This will be easy to filter out, no matter what kind of design has been implemented. For example, if a typical design has a 10% activity factor, it means that the probability of a power transition of q would be 10% ($p_t = Q_{01} = Q_{10} = 0.1$) and it means also that the non-switching probability is 80% ($Q_{00} = Q_{11} = 0.4$). Now when r

Table 1. Transition Probabilities of an RSL nand gate with $A_{01} = 0.1$, $B_{01} = 0.1$

r \ z	$0 \rightarrow 0$	$1 \rightarrow 0$	$0 \rightarrow 1$	$1 \rightarrow 1$
$0 \rightarrow 0$	0.0400	0.0225	0.0225	0.1650
$1 \rightarrow 0$	0.0225	0.0400	0.1650	0.0225
$0 \rightarrow 1$	0.0225	0.1650	0.0400	0.0225
$1 \rightarrow 1$	0.1650	0.0225	0.0225	0.0400
Σ	0.2500	0.2500	0.2500	0.2500

Table 2. A posteriori transition probabilities of RSL nand gate

r \ z	$0 \rightarrow 0$	$1 \rightarrow 0$	$0 \rightarrow 1$	$1 \rightarrow 1$
$0 \rightarrow 0$	0.0400	0.0225	0.0225	0.1650
$1 \rightarrow 1$	0.1650	0.0225	0.0225	0.0400
2Σ	0.4100	0.0900	0.0900	0.4100

switches, these two groups are exchanged, and the circuit gets an activity factor of 40%.

Table 2 shows the transition probabilities after the constant mask-bit transitions have been selected. The a posteriori transition probabilities of the RSL nand gate are not equally likely anymore. They are asymmetric ($Z_{00} = Z_{11} = 0.41 \neq Z_{10} = Z_{01} = 0.09$) and a power analysis should be possible.

3 Experimental Results

3.1 Power Measurements

The power measurements are simulated with toggle counts. A toggle count reports how many signals have a switching event in a clock cycle. By restricting the toggle count to positive signal transitions, they report the power consuming transitions. This is a first order approximation of the power consumption. More accurate power consumption measurements can be obtained by using weight factors based on the estimated capacitance attached to the switching nets. For our purpose, raw un-weighted toggle counts are sufficient. If an implementation is not DPA proof with raw toggle counts it will not be DPA resistant with a more accurate model either. Please note that the inverse would not be true.

We obtained the toggle counts from our test circuit by simulation with the GEZEL cycle-based simulator (`http://rijndael.ece.vt.edu/gezel2`). This simulator supports two modes of toggle counting. In one mode, it obtains toggle counts from a test circuit for all intermediate nets as a function of time. In a second mode, it obtains toggle counts per net over all clock cycles. These counts are obtained by evaluating the Hamming distance of all signal transitions. The

```
1. $option "profile_toggle_upedge_cycles" // count 0->1 transitions
2.
3.  ipblock rng(out q : ns(32)) {
4.    iptype "rngblock";
5.  }
6.
7.  dp rsl_nand(in  blank : ns(1);           // masking bit
8.              in  a, b  : ns(1);           // inputs
9.              out q     : ns(1)) {         // output
10.   always {
11.     q = ~((a & b) | ((a | b) & blank));
12.   }
13. }
```

Fig. 4. Circuit input description for GEZEL cycle-based simulator

partial netlist, shown in figure 4, illustrates a circuit input description for this cycle-based simulator.

Line 1 instructs the simulator to count up-going transitions in the circuit and to report the result per clock cycle.

Lines 3-5 create a random generator module by means of the ipblock construct. These ipblock are user-defined simulation primitives. They are described in C++, and are easy to add to the simulator. The advantage of such user-defined primitives for this application is that they do not contribute to the toggle count. Instead, ipblock primitives are black-box descriptions.

Lines 7-13 show the example model of an RSL-nand gate. This gate will be evaluated once per clock cycle during the simulation. The GEZEL simulator uses a pure cycle-based algorithm and does not simulate glitches. Besides the modeling of combinational logic, GEZEL also supports sequential logic, control modeling, and structural hierarchy. GEZEL also has a code generation backend to convert circuit descriptions into C++ as well as into synthesizable VHDL. The conversion into C++ is useful to generate ipblock descriptions automatically from existing circuit descriptions.

3.2 Device Under Test Setup

Figure 5 shows the block diagram of the test circuit implemented in RSL. The test circuit consists of the AES substitution followed by the key addition. This is a sufficient subset of the AES algorithm on which a SCA can be mounted. Furthermore, a side-channel attack on AES will in general find the 128-bit secret key byte per byte by estimating the side-channel leakage of exactly the circuit shown in figure 5.

The masking is done as follows. The random mask-bit r is inserted at the input, by xor-ing the input *in* and an 8-bit repetition of r. Likewise, the mask is inserted at the other input *key* and extracted at the output *out*. Additionally, the signal r is fed to the RSL gates forming the substitution box and the key addition.

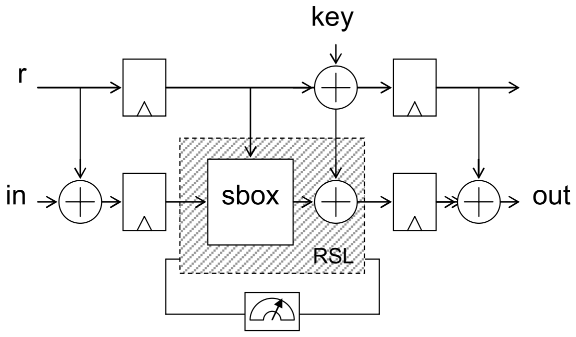


Fig. 5. RSL test circuit: AES sbox and key addition

The power measurements, i.e. the toggle counts, are restricted to the substitution box and the key addition. The switching of the registers storing signals *r*, *in*, and *out*, and the switching of the xor-gates masking signals *in*, *key* and *out* are not included in the power measurements. Only the toggle counts of the logic gates within the dotted line of figure 5 are reported. This has been done to exclude any side-channel leakage of unmasked signals from the measurements.

Additionally, even though the mask-bit *r* is a global signal distributed to all RSL gates it has only weight one. It has the same contribution into the toggle count as a local signal confined between two adjacent gates. In reality, signal *r* behaves as a clock signal, which typically consumes a large fraction –30% to 40% according to [5]– of the total system power. The signal *r*, by itself, will thus cause a large power spike when switching. The observability of a mask-bit transition only increases with more accurate weight factors especially given that the signal *r* is distributed to all logic gates, 871 in our test circuit, while the clock is only distributed to the registers and latches, 18 in our test circuit.

The following power analysis has been carried out. The toggle count measurements are correlated with the number of changing bits between two subsequent values of the signal *in*. The number of changing bits serves as the attacker's estimate of the power consumption. The values of *in* are calculated from the signal *out*, which is known, and a guess on the signal *key*. The guess that results in the highest correlation coefficient is the correct secret key. Please note the power estimation is based on *in* and not on the actual state of the circuit $in \oplus r$.

3.3 Power Based SCA Results

The outcome of the power analysis is shown in figure 6. The results are based on 100,000 toggle count acquisitions, which is more than enough to disclose any side-channel leakage if present. In [7], less than 10,000 measurement acquisitions were required to extract the key in an actual measurement setup suffering from measurement errors and power dissipation of peripheral elements on the die.

The figure shows that the measurements and the estimations of all key hypotheses are uncorrelated. The correlation coefficients are very small and similar

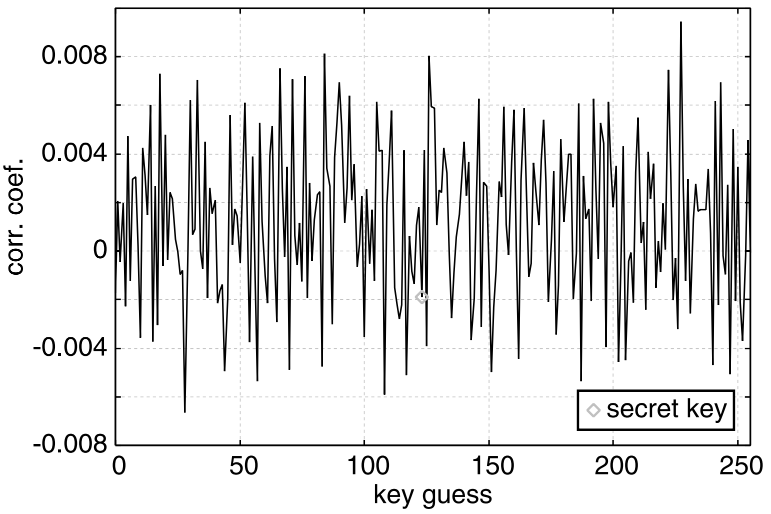


Fig. 6. Power side-channel analysis using 100,000 acquisitions

in size. Not one hypothesis really stands out and the attack did not expose the secret key. Based on these results, one could indeed conclude that RSL is successful in mitigating a power analysis.

The mask-bit value, however, can be derived from a simple power analysis. Figure 7, which shows the toggle count transient together with the value of the mask-bit for 100 clock cycles, confirms the derivations of section 2.2. The toggle count is higher than average whenever the mask-bit changes and smaller than average whenever the mask-bit remains constant.

The separation between the high and the low toggle counts corresponding to a switching and a non-switching mask-bit respectively is even more apparent in the toggle count histogram shown in figure 8. The figure shows that the single mask-bit only impacts the power consumption in a binary fashion. The entropy of the probability density function derived from the toggle count histogram of the test circuit fed with a constant mask-bit and a random mask-bit is 6.24 and 7.24 respectively. The single mask-bit adds exactly one bit of entropy to the information content of the overall power consumption variations.

Note that the toggle count values correspond closely with what would be obtained based on the transition probabilities of table 1. Based on the table, the test circuit, which consists of 388 nand gates and 483 nor gates, would have a toggle count of 380 whenever the mask-bit changes and a toggle count of 80 whenever the mask-bit remains constant.

The toggle count numbers also confirm that gate level masking is an expensive operation. Recall that whenever the mask-bit changes, the configuration switches to its dual mode. Each gate that previously functioned as a nand gate, changes it functionality to a nor gate and vice versa. About 4 times the normal toggle count is required to switch between both configurations. The un-weighted

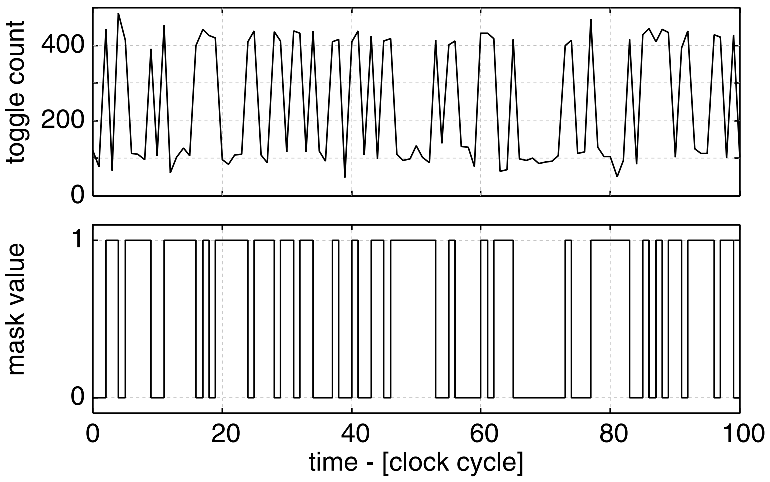


Fig. 7. Toggle count (top); and random mask-bit (bottom) transient

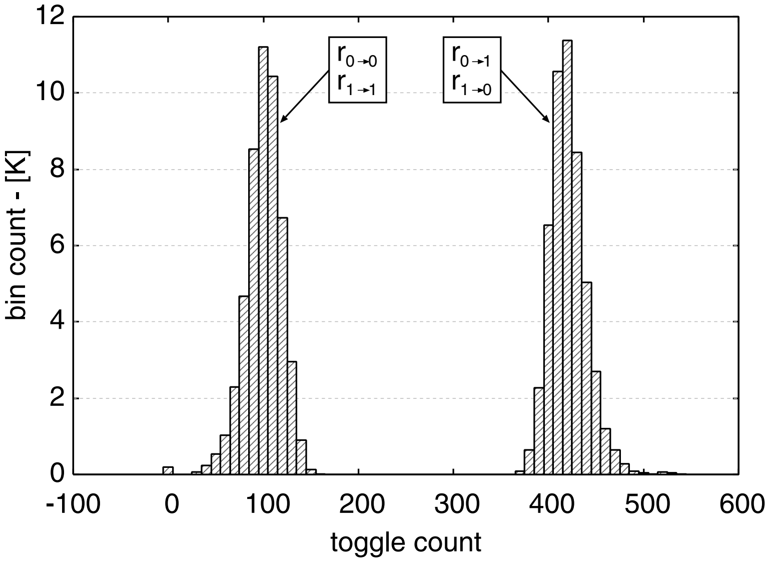


Fig. 8. Toggle count histogram based on 100,000 acquisitions

toggle count power model, which essentially neglects the power dissipation of the global masking signal r and the global enable signal e, estimates the average power consumption of RSL to be 2.5 times the power consumption of a regular implementation.

A threshold filter, which filters out the large toggle counts, retains the events of interest while at same time throws away the unwanted events. The operation

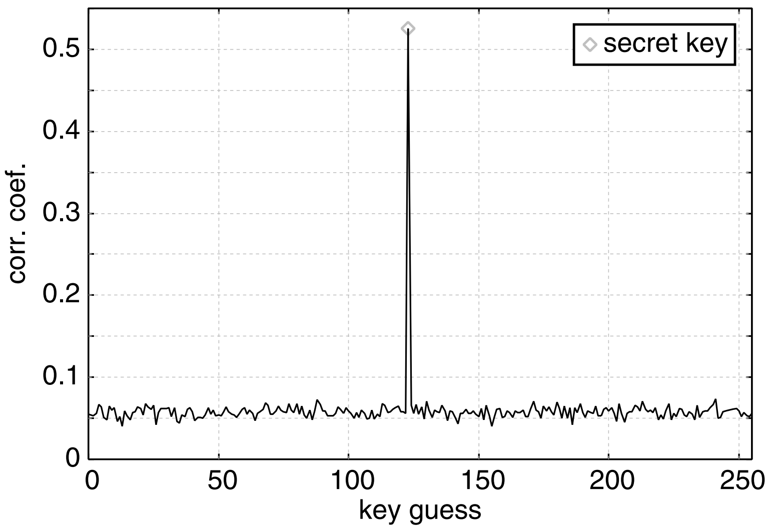


Fig. 9. Power side-channel analysis after threshold filtering operation on 100,000 acquisitions

only keeps the events in which the mask-bit remains constant, thus in which the masking operation was absent. The remaining toggle counts can come from the test circuit in a stable nand-nor configuration or from the test circuit in a stable nor-nand configuration. The fact that these are two different configurations is not important. It is sufficient for the power analysis that both have a power consumption profile proportional to the Hamming distance of two subsequent states. They do not need to have exactly the same power consumption profile.

The outcome of the power analysis after a threshold filter has removed the samples with a toggle count larger than 250 is shown in figure 9. The power analysis successfully exposes the secret key. In fact, 100 acquisitions (yielding approximately 50 samples after the filter) are sufficient to disclose the correct secret key. These experimental results show that RSL is not an efficient countermeasure to mitigate power analysis.

4 Conclusions

Masking is only effective if the mask-bits remain secret. Once the mask-bits are known, the output transitions are not random anymore. A single mask-bit can only impact the power consumption in a binary fashion. For RSL, the mask-bit can easily be derived from the power measurements. A switching mask-bit causes the energy consumption of a typical design to increase four-fold. Once the low energy counts have been separated from the high energy counts, random switching logic can successfully be attacked.

References

1. Chandrakasan, A., Sheng, S., Brodersen, R.: Low Power CMOS Design. IEEE Journal of Solid-State Circuits (JSSC) 27(4), 473–484 (1992)
2. Mangard, S., Popp, T., Gammel, B.: Side-Channel Leakage of Masked CMOS Gates. In: Menezes, A.J. (ed.) CT-RSA 2005. LNCS, vol. 3376, pp. 351–365. Springer, Heidelberg (2005)
3. Peeters, E., Standaert, F., Donckers, N., Quisquater, J.: Improved Higher Order Side-Channel Attacks with FPGA experiments. In: Rao, J.R., Sunar, B. (eds.) CHES 2005. LNCS, vol. 3659, pp. 309–323. Springer, Heidelberg (2005)
4. Moyer, B.: Low-power design for embedded processors. Proceedings of the IEEE 89(11), 1576–1587 (2001)
5. Suzuki, D., Saeki, M., Ichikawa, T.: Random Switching Logic: A Countermeasure against DPA based on Transition Probability. Cryptology ePrint Archive, Report 2004/346 (2004)
6. Suzuki, D., Saeki, M., Ichikawa, T.: DPA Leakage Models for CMOS Logic Circuits. In: Rao, J.R., Sunar, B. (eds.) CHES 2005. LNCS, vol. 3659, pp. 366–382. Springer, Heidelberg (2005)
7. Tiri, K., Hwang, D., Hodjat, A., Lai, B., Yang, S., Schaumont, P., Verbauwhede, I.: Prototype IC with WDDL and Differential Routing - DPA Resistance Assessment. In: Rao, J.R., Sunar, B. (eds.) CHES 2005. LNCS, vol. 3659, pp. 354–365. Springer, Heidelberg (2005)
8. Weste, N., Harris, D.: Principles of CMOS VLSI Design, 3rd edn. Addison-Wesley, Reading (2005)

Advances on Access-Driven Cache Attacks on AES[*]

Michael Neve[1] and Jean-Pierre Seifert[2,3]

[1] Intel Corporation, CTG STL Trusted Platform Laboratory,
2111 NE 25th Avenue, Hillsboro Oregon 97124, USA
michael.neve.de.mevergnies@intel.com
[2] Applied Security Research Group
The Center for Computational Mathematics and Scientific Computation
Faculty of Science and Science Education
University of Haifa
Haifa 31905, Israel
[3] Institute for Computer Science
University of Innsbruck
6020 Innsbruck, Austria
jeanpierreseifert@yahoo.com

Abstract. An access-driven attack is a class of cache-based side channel analysis. Like the time-driven attack, the cache's timings are under inspection as a source of information leakage. Access-driven attacks scrutinize the cache behavior with a finer granularity, rather than evaluating the overall execution time. Access-driven attacks leverage the ability to detect whether a cache line has been evicted, or not, as the primary mechanism for mounting an attack. In this paper we focus on the case of AES and we show that the vast majority of processors suffer from this cache-based vulnerability. Our best results are indeed performed on a processor without the multi-threading capabilities — in contrast to previous works in this area that had suggested that multi-threading actually improved, or even made possible, this class of attack.

Despite some technical difficulties required to mount such attacks, our work shows that access-driven cache-based attacks are becoming easier to understand and analyze. Also, when such attacks are mounted against systems performing AES, only a very limited number of encryptions are required to recover the whole key with a high probability of success, due to our last round analysis from the ciphertext.

1 Introduction

Side channels have been studied for many years in the context of smart cards and embedded systems. Recently some researches demonstrated that microprocessors are also vulnerable to side channels [2,13,14,17], by showing that the

[*] This work has first been presented during the rump session of Crypto 05 by E. Brickell.

© Springer-Verlag Berlin Heidelberg 2007

cache mechanism induces a variability in the execution time due to the different memory accesses. This represents a threat for cryptographic software, since the cache accesses are dependent on the inputs of the software, namely the plaintext and the key. Hence, the analysis of the execution time provides information about the key.

The cache has been mentioned earlier as a potential vulnerability regarding covert channels [6,9,19] and side channels [7,8]. Tsunoo *et al.* demonstrated in [17] the first practical results on DES. They also mentioned results on AES but did not provide further details. In [2], Bernstein showed results of side channel analysis against AES, based on the first round. [10] and [11,12,13] independently provided an analysis of the second round of AES. These attacks belong to the class of *time-driven* cache-based attacks as they analyze the overall execution time.

Moreover [11,12,13] detailed also techniques to perform *access-driven* attacks on AES, where a process *spies* on another one using the cache accesses. [14] used a similar technique against RSA implementations. [3] recently provided software mitigation strategies for AES that reduce the cache leakage.

In this paper, we detail a new and very efficient access-driven cache-based side channel attack. We focus on 128-bit AES implementations that uses four 1KB precomputed SBox tables (such as OpenSSL [1]) and we show that an analysis of the ciphertexts can lead to the recovery of the entire secret key.

Previous attacks [14,12,11,13] exploited the hardware-assisted multi-threading capability of some microprocessors, cf. [15], in order to run a spy process quasi parallel to a crypto process. However, today most processors are single-threaded, therefore, this paper investigates and demonstrates that one can successfully perform such attacks also on this common class of processors.

In practical cases, many processes are quasi-parallel executed at the same time as our crypto and spy processes. Those other processes generate noise in the measurements of the spy process. In this paper however, we are interested in deriving expected numbers of measurements necessary to disclose the full key with perfect measurements, *i.e.* without noise. Nevertheless, we will discuss the consequence of noise on our strategies. Moreover, we will also elaborate on the number of snapshots that the spy process can take per encryption. In addition we will discuss the impact of the measurement resolution upon the quality of the attack.

The present paper is organized as follows. The next Section briefly recalls some facts about AES and cache-based side-chanel attacks. In Section 3, we detail how access-driven attacks can be mounted on single-threaded processors and we demonstrate our practical success by showing a snapshot of a cache activity on such a processor. In common implementations of AES the last round uses a separate SBox table from the other rounds. We show in Section 3 how this information can be combined with the ciphertexts in order to deduce the key. In Section 4 we compute the expected number of cache lines accessed during the last round. We discuss the different cases of attack resolution in Section 5, and two different analysis methods are described in Section 6. Finally we provide our conclusions in Section 7.

2 Definitions and Preliminaries

AES. AES is a popular and commonly used block cipher. We only recall here the particular features of AES that we use in this paper. Refer to [4] for full details. AES operates in a succession of identical rounds, where four operations are performed on the state (*i.e.* the temporary value): an SBox permutation `SubBytes`, a byte transposition `ShiftRows`, a collumn by column permutation `MixColumn` and a sub key addition `AddRoundKey`. The last round however is slightly different since the `MixColumn` operation is skipped.

The key schedule `ExpandedKey` derives the sub keys $K^{(i)}$ from the secret key k. The non-linearity is given by the mean of `SubBytes`. `ExpandedKey` is invertible and, in the case of 128-bit AES, it is possible to derive the secret key from any single sub key. We will use this property in our attack.

Efficient software implementations take advantage of precomputed SBox tables to reach high performances. In OpenSSL for example there are five 1KB tables (T_0, T_1, T_2, T_3, T_4) necessary for the encryption part. All rounds but the last one use 4 of them (T_0 to T_3) whereas the last round and the key schedule use the special fifth one (T_4).

Cache-Based Side Channel Attacks. Access-driven side channels consider that two (or more) processes are executed quasi-parallel on the processor. One process (called here the crypto process) is performing a cryptographic function (*i.e.* AES in this case) involving a secret key. As aforementioned, precomputed values are involved in the execution of the crypto process and their accesses are done through the memory hierarchy. On each data request, the cache checks whether it holds the data, or not. If it does, a cache-hit occurs and the data is immediately transmitted to the processor. Otherwise, a cache-miss occurs and the data must be fetched from a higher memory level, with a longer access time.

A second process, called a spy process, *spies* on the cache accesses of the crypto process. It continuously loads a table S of the size of the cache. From time to time, the crypto process is executed and it inevitably evicts some parts of S by accessing particular data. Therefore, the next time that the spy process is executed, the access time of each part of S (*i.e.* the time necessary to reload a given part of S) indicates which part has been evicted by the crypto process during the last execution of the crypto process.

Thus, the cache is leaking information about the crypto process's memory accesses. Since the software implementation is known, an attacker can infer partial knowledge of the secret key. It is however worth underlining the fact that the spy process cannot diretly access the data of the crypto process; it only observes the cache activity generated by the crypto process and deduces (partial) information from this activity.

3 Exploiting OS Scheduling Instead of Simultaneous Multithreading

Recall that previously described attacks [12, 11, 13, 14] take advantage of the multi-threading capacity of certain processors. It allows them to have two processes running *quasi* parallel on the same processor, as if there were two logical processors [15,16]. In this manner some logical elements are shared, while the quasi parallelism enables one process to *spy* on the other through the use of the shared logic elements. The cache architecture is one such example of a shared element. Although hardware-assisted multi-threading seems to be mandatory at first sight, we show in the rest of this section that it is not.

Although single-threaded processors run threads/processes serially, the OS manages to execute several programs also in a quasi parallel way, only at a coarser resolution, cf. [16]. The OS basically decomposes an application into a series of short threads that are ordered with other application threads. The processor's resources are thus temporally shared according to the OS's ascribed prioritization.

In order to transfer the (hardware-assisted) multi-threaded processor attacks from [12,11,13,14] to single-threaded processors, one has to leave the comfort of hardware-assistance and exploit subtle OS particularities — which may vary from OS to OS. While this seems quite possible for attacks such as [14], the very fast execution time of AES seems to require the aforementioned hardware-assistance in order to efficiently switch between the spy and the crypto process. Indeed, the objective is to ensure is that the crypto thread runs only for a small amount of time between any two runs of the spy thread, or in other words we are able to implement the following strategy:

spy: Continuously watches the cache usage of the parallel crypto thread.
crypto: Runs only for a small amount of time between any two runs of spy.

Interestingly enough, the basic idea is already pointed out in one of the fundamental papers on cache-based side channel attacks, cf. Hu [6], and can adapted to today's OS to stretch the AES execution time over several OS quantums, cf. [6,16]. According to cf. [6], the so called *preemptive scheduling* property, cf. [16], "...allows a process to control when it yields the CPU to another process without waiting until the end of the quantum." Therefore, using the Linux command `sleep` instead of the VAX security kernel command `WAIT` and the following repetitive spy process paradigm we are able to achieve an implementation of the above attack strategy:

– Watch the cache usage.
– Spend most of the OS quantum.
– Yield the CPU to another process via an appropriate `sleep` near to the quantum end.

This paradigm uses the fact that the OS will reschedule the (very short) remaining quantum part to the crypto thread which will be able to execute a few instructions, after which the OS will quickly reschedule to the spy thread, allowing him to spy on the recently used memory accesses. As the above paradigm and all its subtle implementation details heavily depend on the underlying OS, CPU type and frequency, etc. we will not deepen further this technical details here.

Figure 1 shows the successful implementation of the above strategy and was actually created by observing an unmodified AES implementation through the cache accesses. In this Figure, there are 80 columns. Each column represents a single cache line. The 80 columns are divided into five tables of 16 cache lines, each table representing an SBox (starting with T_0 at the left and continuing to T_4 at the right). Each row in this figure indicates a different measuring time (the uppermost being the first measurement). Each point in a row displays the activity of the particular cache line that it represents. The brighter the point, the longer the time it takes to access an element in the cache line. It is important to understand that we get no information about the order of the accesses within one measurement. By using this kind of picture, an attacker can follow the activity of one or more specific cache lines. We obtained similar patterns on another OS and we believe that any multitasking OS could lead to the same access results.

Figure 1 depicts 4 successive AES encryptions. In this particular example, each encryption is repeated 5 times. The time resolution enables us to perform a few measurements per encryption. However, we do not have any distinction between the AES rounds. We only know that they are interrupted several times by the spy program at some points during the encryption.

SBox T_4 [1] plays a particular role here, as it is invoked only on the last round. Therefore, the SBox T_4 accesses indicate the end of an encryption, and all lines within the SBox T_4 accesses are then linked to a single encryption.

4 Analysis of the Last Round

Previously cited attacks use the information about the first or the first and second rounds of one encryption[2]. However, we focus here on the accesses of the last round. Indeed, if the time resolution of the spy process enables us to see the accesses of one encryption, SBox T_4 will also appear clearly.

The ciphertext is now under investigation in order to take advantage of the last round accesses. Recall from our introduction, the last round of AES is particularly of interest in the sense that the `MixColumns` operation is never applied. And for that particular reason, OpenSSL uses SBox T_4, especially for the last round. With $c := E_{AES}(p, k)$ being the ciphertext, we have the following relations linking c and the last round:

$$c = K^{(10)} \oplus \texttt{ShiftRows}[\texttt{SubBytes}[x^{(9)}]],$$

where $x^{(9)}$ is the initial state of round 10 (*i.e.* the output of round 9 and input to round 10). Since round 10 uses SBox T_4, we denote the actual access to T_4 by [SBox T_4 outputs]. The relation becomes:

$$c = K^{(10)} \oplus [\text{SBox } T_4 \text{ outputs}].$$

[1] Recall that T_4 has a size of 1KB and therefore it is represented by the last set of 16 points in each row, with 64-byte cache lines.

[2] But the accesses contain the cache activity of all rounds and the analysis of the first round(s) is disturbed by the access of the other rounds.

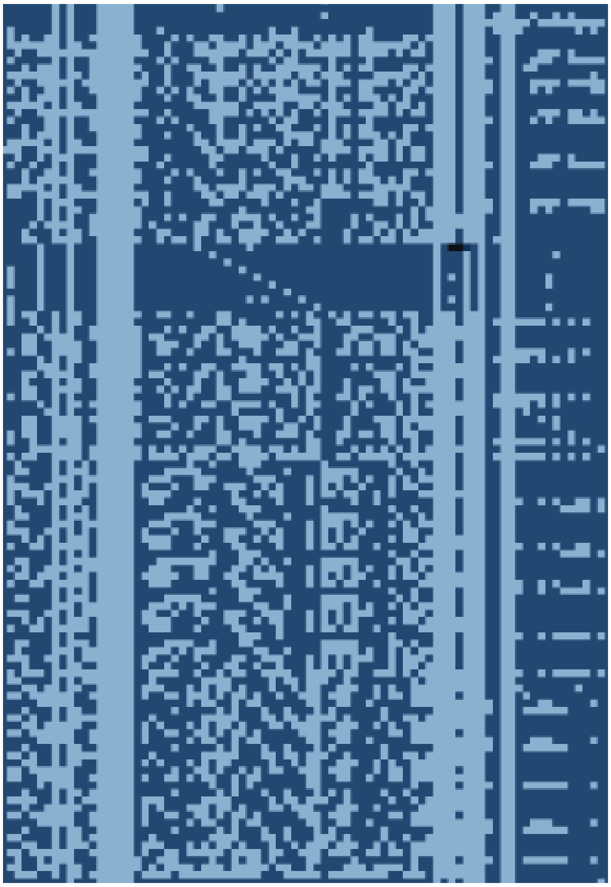


Fig. 1. Evolution of the cache *versus* time, displaying several AES encryptions. Each horizontal line represents the state of the cache lines (represented by a point) at a given time. The brighter, the longer the time to access its corresponding cache line.

Therefore, we derive a relation defining $K^{(10)}$:

$$K^{(10)} = c \oplus [\text{SBox } T_4 \text{ outputs}],$$

from which it is easy to deduce the value of k from $K^{(10)}$ — see our brief recall of AES in section 2.

However, [SBox T_4 outputs] represents the result of *all* the accesses of the last round, *i.e.* for all bytes of the T_4 input $x^{(9)}$. Moreover, the cache accesses only point out the accessed cache lines, but not the individual elements in those lines. Although the next section will give more details on the last points, we refer the reader to Handy [5] for a thorough review of cache architectures.

Table 1. Expected number of cache lines of SBox T_4 accessed in t last rounds for $t > 1$ and $m = 16$

t	2	3	4	5	6	7	> 7
$\mathbb{E}(P(t \cdot 16))$	13.97	15.28	15.74	15.91	15.97	15.99	≈ 16

5 Average Number of Accesses for the Last Round

Let us first introduce some notations. Let $\delta = 2^o$ be the cache line size (in byte) and $m = 2^l$ be the number of cache lines of SBox T_4. Let also $p(b)$ be the probability that one specific cache line is accessed in b T_4 accesses, and $P(b)$ its corresponding random variable. Likewise, $p_n(b)$ the probability that one specific cache line is *not* accessed during b T_4 accesses, and $P_n(b)$ its corresponding random variable. Also, let us assume that the accesses to T_4 are independent and uniformly distributed. We now want to compute the expected number of different cache accesses into T_4. Using that $p(1) = 1/m$, or $p_n(1) = 1 - 1/m$ and the last assumption yields

$$p_n(16) = \left(1 - \frac{1}{m}\right)^{16}.$$

Therefore, the expected number of cache lines *not* accessed in a last round is given by

$$\mathbb{E}(P_n(16)) = 16 \cdot \left(1 - \frac{1}{m}\right)^{16}.$$

In the case of caches with 64 bytes per cache line (*i.e.* $\delta = m = 16$), we get $\mathbb{E}(P_n(16)) = 5.70$ and thus $\mathbb{E}(P(16)) = 10.30$ as the expected number of cache lines accessed during a last round.

6 Resolution

On Figure 1, the T_4 accesses are all visible within a few vertical lines. Let the resolution factor t be defined as

$$t := \frac{\text{\# of ciphertexts}}{\text{\# of measurements}},$$

which yields the following different resolution cases:

– *low resolution*: One measurement covers t encryptions (with $t > 1$) and therefore several last round accesses are overlaid. Then $\mathbb{E}(P(t \cdot 16)) = 16 \cdot (1 - 1/m)^{t \cdot 16}$. Table 1 shows that $\mathbb{E}(P(t \cdot 16))$ rapidly gets close to its limits.
– *one line resolution*: The frequency of measurements isolates one last round per measurement, *i.e.* $t = 1$. We already computed this case. Then $\mathbb{E}(P(16))$ equals 10.30 for $m = 16$.

– *high resolution*: There are several measurements ($1/t$) occurring during the last round, *i.e.* $t < 1$. The observation of the evolution of the accesses gives a notion of the order in which the accesses have taken place and therefore narrows down the possible accesses per byte.

For now, we consider *one line resolution* to detail the analysis of the accesses. We return to this in Section 8 and discuss the impact of the resolution in the analysis's results.

7 Non-elimination and Elimination Methods

We detail here how to deduce the secret key from cache accesses of SBox T_4 and the ciphertexts.

The first method is directly inferred from the relation obtained above:

$$\mathbf{K}^{(10)} = \mathbf{c} \oplus [\text{SBox } T_4 \text{ outputs}].$$

This states that $\mathbf{K}^{(10)}$ is computed with the ciphertext $\mathbf{c}$ and *some* SBox outputs resulting from the SBox T_4 accesses. Each access to a particular line outputs one out of 16 values and we try to discover which one it is, from many ciphertext/accesses pairs. This finally leads to the value of $\mathbf{K}^{(10)}$, when applied in a byte-wise fashion.

The second method is based on the inverse relation:

$$\mathbf{K}^{(10)} \neq \mathbf{c} \oplus \neg[\text{SBox } T_4 \text{ outputs}],$$

where $\neg$[SBox T_4 outputs] refers to the non-accessed cache lines. The relation simply means that the bytes obtained by the addition of $\mathbf{c}$ and the non-accessed cache lines can be discarded as candidates for $\mathbf{K}^{(10)}$. This method, as we are about to see, requires less ciphertext/accesses pairs than the first one.

Let us call those methods respectively *Non-elimination* and *Elimination* methods, since they share the same philosophy as Tsunoo's methods [18]. Let us further suppose that we have a large number of clear measurements of the cache accesses over the last round and the corresponding ciphertexts. We will now detail each method individually.

7.1 Non-elimination Method

This method is separated into three steps. All three steps must be applied for all of the 16 bytes of the key. Suppose we attack byte i, $0 \leq i \leq 15$.

1. *Selection of the ciphertext*: The ciphertext/accesses pairs are sorted according to the value of byte i of the ciphertext. Since the key is constant, it is clear that if the ith byte of different ciphertexts have the same value, all the accesses corresponding to those ciphertexts must contain an access to one common cache line[3].

[3] Since the ciphertexts can be considered random, the other bytes will have random accesses to T_4. We seek the constant access among the random ones.

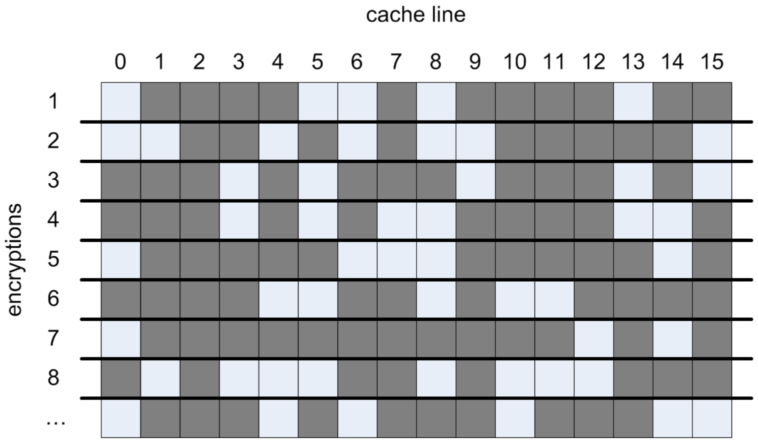


Fig. 2. Cache line accesses for ciphertexts with a constant value for byte i. The dark boxes represent accessed cache lines.

Consider for example Figure 2 as the accesses for a constant value of byte i (say x 00).

2. *Discovery of the correct access*: The access corresponding to the value of byte i is found by taking the unique access present on *every* encryption (cfr. Figure 3 where byte i = x 00 is found to be linked to cache line 2. We define in this case false positives as the wrong candidates present along with the correct candidate: *e.g.* the number of false positives on this example is
   - 10 on encryption 1,
   - 6 on encryption 2,
   - 5 on encryption 3,
   - 3 on encryptions 4 and 5,
   - 1 on encryption 6 and further
   - 0 on encryption 7 and further.

   The probability of a false positive accessed for k successive encryptions is $\left(1 - ((m-1)/m)^{15}\right)^k$ and this gives less than 4 percents when $k = 7$.
3. *Application of the difference*: The bitwise difference of the selected values of byte i must also link two elements in the corresponding access of T_4. Operation (2) showed that byte i = x 00 is linked to cache line 2. Let us assume that the same operation was being executed on a different value of byte i (*e.g.* x 01) and the corresponding cache line was 5. Therefore the bitwise difference of the values for byte i is x 00 $\oplus$ x 01 = x 01 = 1. Hence we only need to find, in the cache lines 2 and 5, output values presenting the same difference. The two lines are shown below:

```
...
2    b7 fd 93 26 36 3f f7 cc 34 a5 e5 f1 71 d8 31 15
...
5    53 d1 00 ed 20 fc b1 5b 6a cb be 39 4a 4c 58 cf
...
```

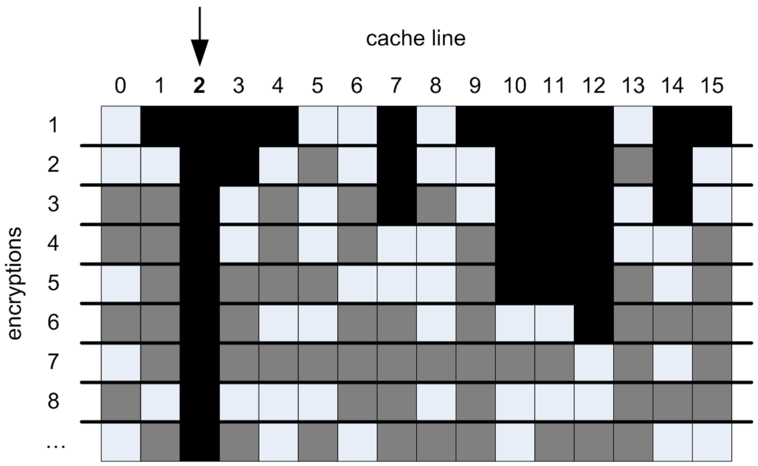


Fig. 3. Highlight of the constant access. The dark boxes represent accessed cache lines and the black boxes show the evolution of the possible candidates.

The only pair having a bitwise difference of 1 are `xfd` and `xfc`, when byte i is equal to respectively `x00` and `x01`. Therefore, the byte of $\mathbf{K}^{(10)}$ corresponding to byte i has a value of $\texttt{xfd} \oplus \texttt{x00} = \texttt{xfc} \oplus \texttt{x01} = \texttt{xfd}$. In the unlikely event of more than one match, the operation (2) must be repeated to establish other bitwise differences.

The expected number of pairs to find the correct key byte is

$$\sum_{n=2}^{\infty} p_{fp}(n) \cdot N(n) \approx 186,$$

with $p_{fp}(n)$ and $N(n)$ respectively being the probability of having a false positive after n pairs and the average number of pairs necessary to get two values repeated. The other bytes of $\mathbf{K}^{(10)}$ are found the same way, by considering another byte number.

7.2 Elimination Method

Here, all bytes can be treated at the same time. We consider the case of byte i for the sake of clarity; it is straightforward to apply the method to the other ones. Let $\mathcal{V}$ be the set of all possible key byte values. Initially, $\mathcal{V}$ is composed of all 256 values a byte can take: $\mathcal{V} = \{j : 0 \leq j \leq 255|j\}$. At the end, we want that $\mathcal{V} = \{k_i^{(10)}\}$. Consider for example that the ciphertext's byte i c_i equals `x2c` and the corresponding accesses are the ones displayed in Figure 4.

The accessed cache lines are

1 2 3 5 6 7 9 10 12 14 15

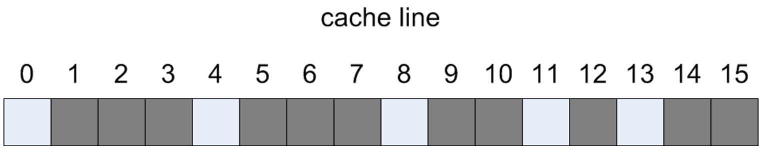


Fig. 4. Example of accessed cache lines. The dark boxes represent accessed cache lines.

and the non-accessed ones are

0 4 8 11 13.

This method focuses on the latter list of cache lines. Let $\widetilde{\mathcal{A}}$ represent this subset of the cache lines and $n_{\widetilde{\mathcal{A}}}$ be the number of elements of $\widetilde{\mathcal{A}}$:

$$\widetilde{\mathcal{A}} = \{0, 4, 8, 11, 13\}, \qquad n_{\widetilde{\mathcal{A}}} = \left|\widetilde{\mathcal{A}}\right| = 5.$$

By the elimination relation, $\mathbf{K}^{(10)} \neq \mathbf{c} \oplus \neg[T_4 \text{ outputs}]$, each non-accessed cache line enables us to remove all key candidates corresponding to this access. In our example, this means that for the first element of $\widetilde{\mathcal{A}}$ we have:

$$\begin{aligned} K_i^{(10)} &\neq c \oplus [\text{cache line } 0] \\ &\neq \mathtt{x\,2c} \oplus {}_{\mathtt{x}}\{\mathtt{63,7c,77,7b,f2,6b,6f,c5,30,01,67,2b,fe,d7,ab,76}\} \\ &\neq {}_{\mathtt{x}}\{\mathtt{4f,50,5b,57,de,47,43,e9,1c,2d,4b,07,d2,fb,87,5a}\} \\ &= \mathcal{V}_e, \end{aligned}$$

where $\mathtt{x}\ldots$ and ${}_{\mathtt{x}}\{\ldots\}$ represent hexadecimal values. All values of $\mathcal{V}_e$ can then be eliminated from $\mathcal{V}$:

$$\mathcal{V} \leftarrow \{j : 0 \leq j \leq 255\} \backslash \mathcal{V}_e$$

Then we go to the next element of $\widetilde{\mathcal{A}}$ (*i.e.* cache line 4) and apply the same technique. The cache line bytes are

$${}_{\mathtt{x}}\{\mathtt{09,83,2c,1a,1b,6e,5a,a0,52,3b,d6,b3,29,e3,2f,84}\}$$

added with $\mathtt{x\,2c}$ gives new candidates to eliminate. Then $\mathcal{V}$ is updated as:

$$\mathcal{V} \leftarrow \mathcal{V} \backslash \{\mathtt{25,af,00,36,37,42,76,8c,7e,17,fa,9f,05,cf,03,a8}\}.$$

This is then repeated for the three other cache lines in $\widetilde{\mathcal{A}}$.

For one given ciphertext/accesses pair, each cache line ends up eliminating 16 different values from the byte candidates: the ciphertext byte is constant and the SBox outputs are all different from each other. In our example 80 candidates have been eliminated with the pair under consideration. Then, another ciphertext/access pair is analyzed and the same technique is applied with the non-accessed cache lines of that pair. The ciphertext's byte i and the non-accessed cache lines are probably different from the previous analyzed pair. Therefore the

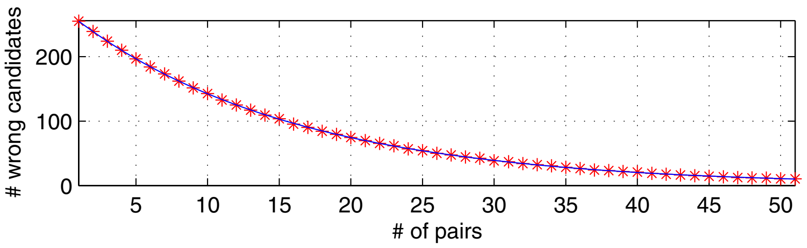


Fig. 5. Experimental verification of the reduction formula, yielding the number of wrong key candidates per pairs. The continuous line is the theoretical formula and the stars are the simulated data.

subset of wrong candidates deduced from this pair should only present a few collisions with the one of previous pairs.

However, there will be more and more collisions as the number of wrong key candidates becomes closer to one. Consider for example the following table showing the reduction of the number of wrong key candidates, for a practical case.

#	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\lvert\mathcal{V}\rvert$	255	175	119	77	55	33	21	15	6	6	2	2	2	1	1	0

Then other pairs are analyzed until there is only one key byte remaining[4]. It is important to note that this method allows to work on all the bytes, at the same time. In this case, we need 16 subsets (one per byte) keeping the count of all candidates:

$$\mathcal{V}_0, \ldots, \mathcal{V}_{15}.$$

Let us first compute the number of pairs needed to distinguish the right key candidate, for one byte. We can now define s, the number of candidates eliminated by the analysis of one ciphertext/access pair. At the beginning, we have 255 wrong candidates. With the first pair, we eliminate s of them and the number of wrong candidates is 255-s. However, starting at the second pair, collisions can occur. Therefore, we approximate the number of remaining wrong key candidates after n pairs by $255 \cdot (1 - s/255)^n$. This formula is validated by simulation with $s = 16$ (see Figure 5).

We then apply the formula with the expected number of non-accessed cache lines (*i.e.* 5.70, for 64-byte cache lines). Hence, substituting $s = 5.70 \cdot 16$ into $255 \cdot (1 - s/255)^n$ we get the following results[5] (cfr. Table 2).

After approximatively 14 pairs, the number of wrong candidates for one byte should be close to 0. This shows that less than 20 ciphertext/accesses pairs are

[4] Note that one can stop anytime and run an exhaustive search on the remaining key byte candidates.

[5] The data differ from the ones in the practical case because there are 5 non-accessed cache lines whereas 5.70 are considered in Table 2.

Table 2. Theoretical results of wrong key candidates, per pair ciphertext/accesses, for $m = 5.70 \cdot 16$

# pairs	$\lvert\mathcal{V}\rvert$	# pairs	$\lvert\mathcal{V}\rvert$
0	255	8	7
1	164	9	5
2	105	10	3
3	68	11	2
4	43	12	1
5	28	13	0.8
6	18	14	0.5
7	12	15	0.3

needed to recover the whole $\mathbf{K}^{(10)}$ subkey and therefore also the secret key $\mathbf{k}$. This method gives a much better performance than the non-elimination one.

8 Practical Considerations

Let us now re-elaborate the question of the measurement resolution.

- *Low resolution*: Table 1 highlighted that the expected number of accessed cache lines rapidly approaches to 16, when the number of encryptions between two measurements increases. However, even if the leakage gets smaller, every ciphertext/accesses pair with at least one non-accessed cache line carries information. Moreover, low resolution implies multiple ciphertexts for a single cache information (*i.e.* one line combines all the accesses corresponding to the ciphertexts). In this case the analysis must integrate the different possible ciphertext values and statistically derive the most likely key bytes.
- *One line resolution*: As detailed above, 5.70 cache lines are not accessed. The analysis does not need to deal neither with the multiple ciphertexts issue nor with the order inside the accesses.
- *High resolution*: Both methods are still possible. But the leakage also gives some information about the order of the accesses. One can then increase the performances of the analysis and therefore reduce the number of required pairs, by correlation of the byte accesses in the AES program and the accesses visible in the measurements. For $t \leq 1/16$, one can clearly identify the byte accesses. Two to three pairs only makes it possible to find the correct candidates for all key bytes[6].

Finally, we considered in this paper that the cache accesses were exempted of any measurements noise. However practical attacks must deal with noise in the measurements. Consider for example Figure 1 which presents vertical stripes and a diagonal line in the upper half. The presence of noise in the measurements

[6] The elimination and non-elimination methods then presents the same performances.

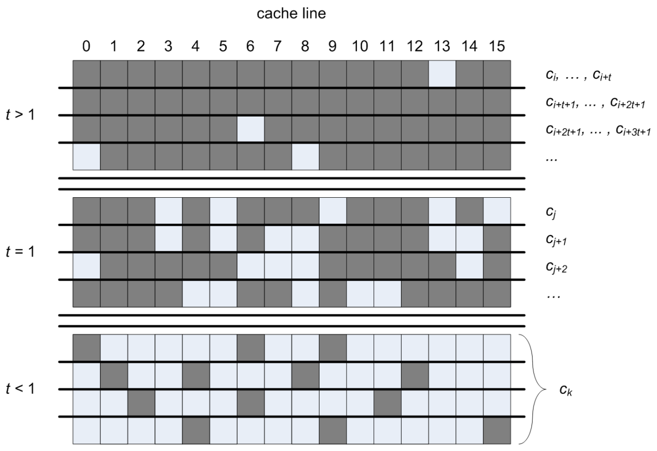


Fig. 6. Different resolutions for access-driven cache-based attacks. The resolution factor t defines the ratio *# of ciphertexts / # of measurements*.

increases the number of accessed cache lines. However, the techniques that we detailed here can still be exploited, by taking into account the noise[7].

We gave above the minimum expected number of measurements to perform the attacks for $t = 1$. As this boundary is precious and has been practically confirmed it should used to evaluate the efficiency and security of current software implementations which are hardened by corresponding countermeasures.

9 Summary

In this paper, we detailed advances on recent processor-oriented side channels. Our contribution is two-fold: we detailled a software method to achieve snapshots of cache accesses on single-threaded processors and we showed that the analysis of the last round of AES enables the full disclosure of an 128-bit AES key with less than 20 encryptions. Where previous studies focused exclusively on a minority of processors, we investigated the access-driven cache-based attacks on single-threaded processors. We explained our strategy and why it is solely depending on software engineering. Moreover, we chose the challenging case of AES: its short execution time (compared to RSA's) demonstrates the fine granularity of our cache accesses' snapshots. Our software strategy can easily be adapted and combined with previously reported access-driven attacks on any single-threaded processor. Moreover, on common implementations the last round is performed with the help of a special precomputed table. Through this feature, we achieved to infer more information than with other strategies. We gave expected numbers of measurements, depending on the granularity and

[7] Also, the location of the vertical stripes is variable between different runs of the setup.

noise of the access-driven measurements. This contribution sets new boundaries for countermeasures against cache-based attacks. For example, some software mitigations proposed to apply masking techniques and to renew the mask every 256 encryptions. We showed in this paper that this number might have to be reconsidered.

Acknowledgment

We would like to thank the anonymous reviewers for their useful comments and also for this sentence "Figure 1 should be framed on the wall in front of every crypto software programmer".

References

1. Openssl: the open-source toolkit for ssl / tls. Available online at http://www.openssl.org/
2. Bernstein, D.J.: Cache-timing attacks on AES (2004), Available onlineat http://cr.yp.to/papers.html#cachetiming
3. Brickell, E., Graunke, G., Neve, M., Seifert, J.-P.: Software mitigations to hedge aes against cache-based software side channel vulnerabilities. Cryptology ePrint Archive, Report, 2006/052 (2006), Available online at http://eprint.iacr.org/
4. Daemen, J., Rijmen, V.: The design of Rijndael, AES - The Advanced Encryption Standard. In: Information Security and Cryptology, Springer, Heidelberg (2001)
5. Handy, J.: The cache memory book (2nd ed.): the authoritative reference on cache design. Academic Press, Inc., Orlando, FL, USA (1998)
6. Hu, W.-M.: Lattice scheduling and covert channels. In: Proceedings of the IEEE Symposium on Security and Privacy, vol. 25, pp. 52–61 (1992)
7. Kelsey, J., Schneier, B., Wagner, D., Hall, C.: Side channel cryptanalysis of product ciphers. Journal of Computer Security 8(2/3) (2000)
8. Kocher, P.C.: Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In: Koblitz, N. (ed.) CRYPTO 1996. LNCS, vol. 1109, pp. 104–113. Springer, Heidelberg (1996)
9. Lampson, B.W.: A note on the confinement problem. Communications of the ACM 16(10), 613–615 (1973)
10. Neve, M., Seifert, J.-P., Wang, Z.: A refined look at Bernstein's AES side-channel analysis. In: Proceedings of AsiaCCS 2006 (2006)
11. Osvik, D.A., Shamir, A., Tromer, E.: Cache atacks and countermeasures: the case of AES (extended version) (2005), Available online at http://www.wisdom.weizmann.ac.il/ tromer/
12. Osvik, D.A., Shamir, A., Tromer, E.: Cache attacks and countermeasures: the case of aes. Cryptology ePrint Archive, Report, 2005/271, (2005) Available online at http://eprint.iacr.org/2005/271.pdf
13. Osvik, D.A., Shamir, A., Tromer, E.: Cache attacks and countermeasures: The case of aes. In: Pointcheval, D. (ed.) CT-RSA 2006. LNCS, vol. 3860, pp. 1–20. Springer, Heidelberg (2006)
14. Percival, C.: Cache missing for fun and profit (2005), Available online at http://www.daemonology.net/hyperthreading-considered-harmful/

15. Shen, J., Lipasti, M.: Modern Processor Design: Fundamentals of Superscalar Processors. McGraw-Hill, New York (2005)
16. Silberschatz, A., Gagne, G., Galvin, P.B.: Operating system concepts, 7th edn. John Wiley and Sons, Inc., USA (2005)
17. Tsunoo, Y., Saito, T., Suzaki, T., Shigeri, M., Miyauchi, H.: Cryptanalysis of des implemented on computers with cache. In: D.Walter, C., Koç, Ç.K., Paar, C. (eds.) CHES 2003. LNCS, vol. 2779, pp. 62–76. Springer, Heidelberg (2003)
18. Tsunoo, Y., Tsujihara, E., Minematsu, K., Miyauchi, H.: Cryptanalysis of block ciphers implemented on computers with cache. In: Proceedings of International Symposium on Information Theory and Its Applications, pp. 803–806 (2002)
19. Wray, J.C.: An analysis of covert timing channels. Journal of Computer Security 1(3-4), 219–232 (1992)

Blind Differential Cryptanalysis for Enhanced Power Attacks

Helena Handschuh[1] and Bart Preneel[2]

[1] Spansion,
7 Avenue Georges Pompidou,
92593 Levallois-Perret Cedex, France
helena.handschuh@spansion.com

[2] Katholieke Universiteit Leuven, Dept. Electrical Engineering-ESAT/COSIC,
Kasteelpark Arenberg 10,
B-3001 Leuven-Heverlee, Belgium
bart.preneel@esat.kuleuven.be

Abstract. At FSE 2003 and 2004, Akkar and Goubin presented several masking methods to protect iterated block ciphers such as DES against Differential Power Analysis and higher-order variations thereof. The underlying idea is to randomize the first few and last few rounds of the cipher with independent masks at each round until all intermediate values depend on a large number of secret key bits, thereby disabling power attacks on subsequent inner rounds. We show how to combine differential cryptanalysis applied to the first few rounds of the cipher with power attacks to extract the secret key from intermediate unmasked (unknown) values, even when these already depend on all secret key bits. We thus invalidate the widely believed claim that it is sufficient to protect the outer rounds of an iterated block cipher against side-channel attacks.

Keywords: differential cryptanalysis, power analysis, side channel attacks, Hamming weights, combined cryptanalysis, blind cryptanalysis.

1 Introduction

In 1998, Kocher et al. introduced Differential Power Attacks on block ciphers and digital signature algorithms [10]. These attacks allow to recover secrets used in cryptographic computations even if these are executed inside tamper-resistant devices such as smart cards. Kocher noted that these devices leak information which is directly correlated to the secret data being manipulated inside the device. The information may be recovered for example by measuring the power consumption of the device and the correlation of its variation with the secret data. Differential Power Analysis exploits the fact that computing a given output bit of a non-linear S-box requires different power consumption when this bit is set to zero or to one; correlation analysis extends this by correlating the power consumption with the key dependent power consumption predicted by a model. Since 1998, these techniques have been generalized to other side-channels such as timing information, electro-magnetic radiation, and even sound waves; research

© Springer-Verlag Berlin Heidelberg 2007

has also focused on how to protect tamper-resistant computations against these attacks. Countermeasures are applied at the hardware, software and protocol level. At the hardware level, power consumption scramblers and ad-hoc noise introduction via random execution delays or random operation execution are the preferred methods. At the software level, the most useful technique against first order differential attacks are randomization techniques. In essence, the intermediate values are blinded using some randomized masks in order to decorrelate them from the actual values which would reveal information to the opponent. These techniques include random masking methods, randomized exponentiation techniques, and randomized execution paths or integer representation. At the protocol level, fast key refreshing has been a useful countermeasure. For further references on side channel attacks and countermeasures, see for example [13,5,9,23,8,20].

Some of these methods have since been shown to be vulnerable to higher order differential attacks (see Messerges [14]) in which an opponent can measure information at different places in a single power consumption curve. Kunz-Jacques et al. [11] have shown how to improve higher order attacks on DES by combining them with the Davies-Murphy attack. For block ciphers, new masking methods have been proposed in which an independent random mask is applied at each round, thus preventing an attacker to take advantage of the repeating mask in a higher-order differential attack. However, these protection methods require very large quantities of volatile memory and pre-computation time, which is typically cost-prohibitive in secure embedded devices. Therefore only a few rounds are eventually masked against power analysis attacks and the inner rounds of the cipher are left unmasked (see Akkar et al. [3,2]). In their paper on cache-based attacks, Osvik et al. [18] independently suggested that differential attacks could be used to bypass protection in the outer rounds, but no details are provided.

Our Contribution. In this paper we show how to attack secure implementations of iterated block ciphers which apply reduced-round masking methods to protect their secret keys against side-channel attacks such as power attacks. We are able to mount key recovery attacks based on differential cryptanalysis techniques [4] and power traces providing only the Hamming weight of the internal variables used throughout the computation. Compared to differential cryptanalysis, the main difference is that our technique is *blind* in the sense that we do not *see* the actual values at the output of the differential path since the path stops somewhere in the middle rounds of the cipher, but can only *derive* them from their measured Hamming weight. As an example we explain how the technique works on the Unified Masking Method applied to the DES.

Organization of the Paper. Section 2 explains the unique masking method and its extensions which formed the inspiration of this attack. In Sect. 3 we explain our blind differential attack for the specific case of DES with the outer four rounds masked. In Sect. 4 we present our simulation results. Section 5 discusses improvements and generalizations and Sect. 6 concludes the paper.

2 Extended Unique Masking Method

The Unique Masking Method (UMM) described below was proposed by Akkar and Goubin [3] and applies to Feistel ciphers such as DES [15] and Substitution Permutation Networks such as AES [16]. We use here the first type as an example. In Feistel ciphers, the plaintext M is split into two halves L_0 and R_0 such that $M = L_0||R_0$ and a round function f is applied to the right half of its input before the result is exored to the left half. Next, both halves are swapped and the procedure iterates for r rounds:

$$L_{i+1} := R_i \quad \text{and} \quad R_{i+1} := L_i \oplus f(R_i) \quad 0 \leq i \leq r-1 \ .$$

The ciphertext is equal to $C = R_r||L_r$ (in the last round, the halves are not swapped). The DES round function comprises a key addition operation, followed by an expansion operation E and substitution through a layer of 8 tables or S-boxes S (each mapping 6 bits to 4 bits); next a bit-level permutation P is applied to the result. UMM proposes to mask the outer rounds of a Feistel block cipher such as DES with different independent masks at each round for at least four rounds in order to decorrelate the calculations from the actual intermediate data. In order to achieve this, the S-boxes S are replaced with different S-boxes, the input and output of which are masked with random data. Since these S-boxes are the only non-linear part of the cipher, new S-boxes need to be generated dynamically at each execution of the algorithm to account for the different input and output translations introduced by the random masks. UMM uses two sets of S-boxes.

Let S_1 and S_2 denote the following two new functions based on the original DES S-boxes S, where α represents the 32-bit input and output mask used during one execution of the algorithm:

$$\begin{cases} \forall\, x \in \{0,1\}^{48}, & S_1(x) = S(x \oplus E(\alpha)) \\ \forall\, x \in \{0,1\}^{48}, & S_2(x) = S(x) \oplus P^{-1}(\alpha) \ . \end{cases}$$

These two new functions are logically combined such that one output mask synchronizes with the input mask of the next round automatically as shown in Fig. 1. Akkar et al. show in [2] that their initial UMM method does not achieve the desired goal as the second round output remains unmasked. Therefore they propose to use a third independent set of S-boxes S_3 such that $\forall\, x \in \{0,1\}^{48}$, $S_3(x \oplus E(\alpha)) = S(x) \oplus P^{-1}(\alpha)$ and completely mask all intermediate data up to the fourth round of DES as shown in Fig. 2. A different value for α should be used at each execution of the algorithm. This scheme uses S_3 in both round 2 and 3; one could avoid this by introducing an additional mask β and by defining S_3' that transforms a mask α into a mask β and S_3'' that does the opposite. Since these masking techniques require the generation of an independent translated S-box for each round, which represents a large cost in terms of volatile memory (256 bytes of RAM each), only four rounds are masked until each intermediate data bit depends on all the key bits of the block cipher. By combining Power Attacks with Differential Cryptanalysis, we will show that this scheme is still not secure even if four or more independent S-box layers are chosen in each run of the algorithm.

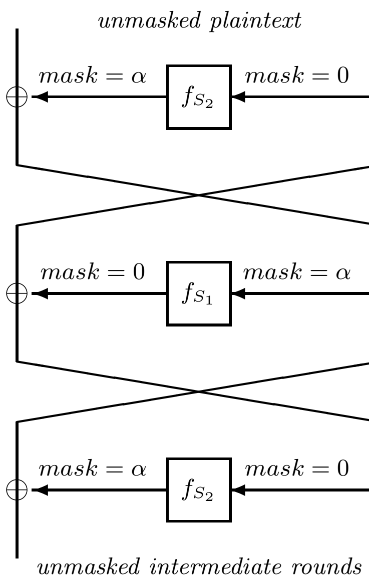


Fig. 1. Example application of Akkar and Goubin's initial masked f-function chaining method with three masked rounds [3]

3 Mounting a Blind Differential Attack on 4-Round DES

Differential attacks as first described by Biham and Shamir [4] are chosen plaintext attacks in which an adversary chooses pairs of plaintexts with given differences and tries to deduce information from the corresponding pairs of ciphertexts. With a certain probability, a given plaintext difference follows a pre-determined path throughout the encryption operations and results in a given ciphertext difference. An input pair that results in the correct intermediate and output differences is called a right pair. When the adversary finds a right pair, he can deduce information on the last round key from the pre-determined differential path. For more details we refer the reader to the original paper describing this technique [4]. We apply differential cryptanalysis to four-round DES using Biham's original four-round differential characteristics. The innovation in our attack is that we cannot observe the differences directly in the ciphertext pair at the output of the cipher as these differences only appear in internal rounds of the encryption process. Therefore we call our method 'blind differential cryptanalysis.'

3.1 Enhanced Power Attacks

Now Power Attacks come into play. In power attacks, a generally admitted model is that the adversary can measure side-channel information which leaks a linear function of the individual data bits, for example the Hamming weight of the data (see for example [1,6,7] for a discussion of this model). In this setting, we can

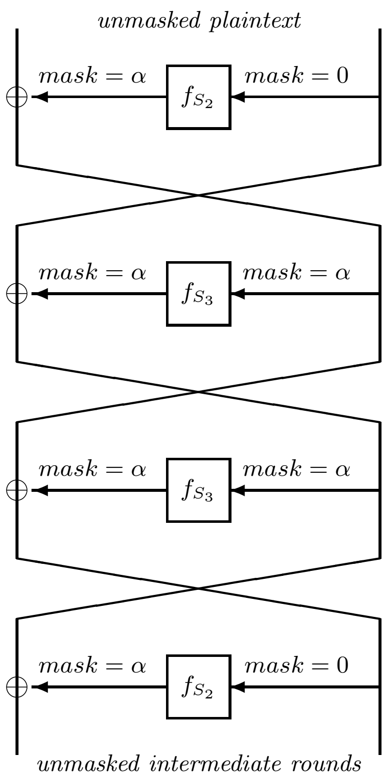


Fig. 2. Example application of Akkar, Bévan and Goubin's improved masked f-function chaining method with four masked rounds [2]

combine the expected value of the difference at round four of the DES and the Hamming weight observations of the power attack. In our four-round differential represented in Fig. 3, several difference bytes are equal to zero, meaning that the corresponding data bytes are equal. This in turn implies that the Hamming weights are equal. Note that the converse is not necessarily true. Thus, our power measurements will both enable us to *see* when two bytes are potentially equal and will provide a large number of false alarms at the same time, since equal Hamming weights do not imply equal data bytes. However, filtering out the pairs which *do* reveal the same Hamming weight on the required data bytes, we can now apply our blind differential key recovery attack to recover part of the secret key of the fourth round. Note that a collision technique (i.e., searching for identical values and thus Hamming weights) has been used by Schramm et al. to improve power analysis attacks [21]. Ledig et al. [12] showed that this attack can be further enhanced by exploiting the slow increase of Hamming distances in the rounds following a collision. However, in our attack we explicitly make use of right pairs for arbitrary characteristics and we show how the key can be recovered even if no collisions occur at all.

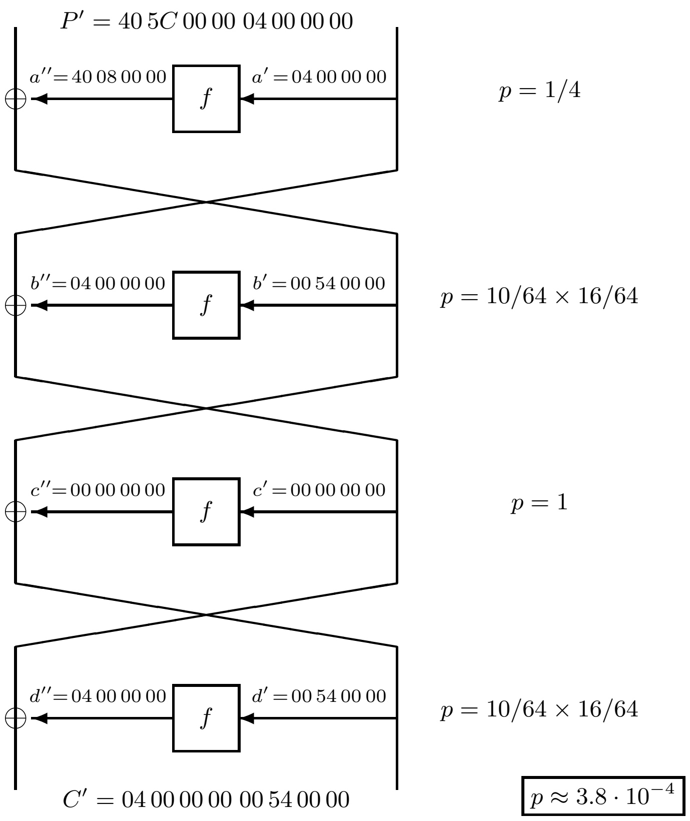


Fig. 3. Biham and Shamir's 4-round differential characteristic for DES and the associated probabilities [4]

3.2 Blind Key Recovery

Once we have filtered out the right pairs using equal Hamming weights on the 6-bit S-box inputs, we need to find a technique which allows to recover the secret key bits involved in this round without knowing the actual intermediate values. Recall that we also have access to the absolute value of the Hamming weight of the data before key addition. For every DES S-box it is easy to construct the difference distribution table of DES and to determine which input pairs yield the right output difference. These actual input values will now be used in the following way.

Consider an active S-box in the fourth round, that is, an S-box with a non-zero input difference δ and output difference Δ. Denote the 6-bit input of the fourth round corresponding to this S-box by x_i, the 6-bit key by k, the 6-bit input of the S-box by y_i and the 4-bit output of the S-box by z_i. Clearly one has $y_i = x_i \oplus k$. In classical differential cryptanalysis applied to a Feistel cipher, one finds a number of right pairs and then deduces candidate values for k from the values (y_i, y'_i) that correspond to the characteristic and from the known values of (x_i, x'_i) (note that $x_i \oplus x'_i = y_i \oplus y'_i = \delta$). Ohta and Matsui [17] and Preneel et al. [19] have extended this attack to the case where not all bits of the x_i and y_i are known in order to attack CBC-MAC and the CFB mode of DES (or reduced-round variants of DES).

In our new attack only the Hamming weights of the intermediate plaintexts (x_i, x'_i) are known. At first sight it seems rather easy to deduce candidate values for k by generating all the 6-bit values with the correct Hamming weight and eliminating those which are not compatible with the characteristic. The remaining candidate intermediate plaintexts then suggest several values for the key k; by considering multiple right pairs, the correct value of k should appear. Unfortunately this attack does not work, since all or almost all 64 key values are suggested by each right pair. Therefore we have developed a new approach.

Consider the set $Y = \{(y_i, y'_i) \mid y_i \oplus y'_i = \delta \text{ and } z_i \oplus z'_i = \Delta\}$. Consider a fixed value of k. Define the set X_k as follows:

$$X_k = \{(x_i, x'_i) = (y_i \oplus k, y'_i \oplus k) \mid (y_i, y'_i) \in Y\} .$$

This set can be partitioned according to the value $(\mathrm{hwt}(x_i), \mathrm{hwt}(x'_i))$. Note that due to the constraint $x_i \oplus x'_i = \delta$ not all combinations of these integers can occur. It is easy to see the following cases:

$\mathrm{hwt}(\delta) = 1$**:** then $\mathrm{hwt}(x_i) = \mathrm{hwt}(x'_i) \pm 1$
$\mathrm{hwt}(\delta) = 2$**:** then $\mathrm{hwt}(x_i) = \mathrm{hwt}(x'_i) \pm 2$ or $\mathrm{hwt}(x_i) = \mathrm{hwt}(x'_i)$
$\mathrm{hwt}(\delta) = 3$**:** then $\mathrm{hwt}(x_i) = \mathrm{hwt}(x'_i) \pm 1$ or $\mathrm{hwt}(x_i) = \mathrm{hwt}(x'_i) \pm 3$

We now define the Hamming weight profile PP_k of the key k as follows:

$$\mathrm{PP}_k[i,j] = |\{(x_i, x'_i) \mid (x_i, x'_i) \in X_k \text{ and } \mathrm{hwt}(x_i) = i, \mathrm{hwt}(x'_i) = j\}| ,$$

or in words: $\mathrm{PP}_k[i,j]$ is the number of input pairs to an active S-box for which the round inputs have Hamming weight i and j respectively.

The attack proceeds as follows:

1. Collect a sufficient number of right pairs; note that since our filtering mechanism based on Hammming weights is not perfect, not all the retained pairs will be right pairs.
2. Compute an estimate for the Hamming weight profile $\hat{\mathrm{PP}}_k[i,j]$ based on the measurements.
3. Perform a matching between the observed profile and the profiles of all the values for k. We propose the use of a mean square error (MSE) as a matching criterion
$$\mathrm{MSE}_{k^*} = \sum_{[i,j]} \left(\mathrm{PP}_{k*}[i,j] - \hat{\mathrm{PP}}_k[i,j]\right)^2 .$$
The idea of the attack is that for the correct value of k the Euclidean distance between the two profiles will be very small (and typically smaller than that for other keys).

Note that due to the symmetry property, we have that $\mathrm{PP}_k[i,j] = \mathrm{PP}_{k\oplus\delta}[i,j]$, hence we cannot distinguish between k and $k \oplus \delta$.

Overall, a few dozen right pairs and the corresponding power measurements provide enough information about the Hamming weights of intermediate data before the S-boxes to recover two candidates for the secret 6-bit key. Note however that this technique only allows to recover key elements corresponding to the active S-boxes. Therefore we need to use several *different* four-round characteristics to recover all 6-bit elements of the secret key. Fortunately, there are many four-round characteristics available for differential cryptanalysis of DES, and we do not have to use only the best one in our attack. Once sufficient key bits have been obtained, the remaining bits can be recovered by exhaustive search.

4 Simulations

We have performed some experiments in software on a PC to validate the analysis in Sect. 3. We assume that we can measure Hamming weights in a reliable way, that is, our simulated measurements are noise free. This assumption may not hold in practice, in particular if additional noise is added as a countermeasure. However, we believe that our methods are sufficiently robust to also work (with an increased number of measurements) under noisy conditions.

The probability p of the characteristic we use is $3.8 \cdot 10^{-4}$ (see Fig. 3); it has two active S-boxes. We use the Hamming weights of the left half output of the fourth round (which is also the input to the next round) to filter out right pairs. There are seven passive S-boxes in the fifth round which allows to almost uniquely identify right pairs as being those which follow the differential path. A very rough estimate shows that the probability p_f to obtain equal Hamming weights on seven 6-bit elements in the pair is approximately $3 \cdot 10^{-5}$; it can be computed as follows:

$$p_f = (p^*)^7 \quad \text{with } p^* = \frac{1}{2^{12}} \cdot \sum_{i=0}^{6} \binom{6}{i}^2 .$$

This filtering function has roughly the same probability of success as regular differential filters, and as noted in Sect. 3.1, we can guarantee that all right pairs are correctly identified, and few false alarms appear. A right pair in the sense of differential cryptanalysis will automatically yield a right pair in the sense of Hamming weights. The Hamming weight difference for a wrong pair is not uniformly distributed – it is more likely to be equal to that of a right pair. In our simulations we have noted that our Hamming weight filter is about a factor of two worse than the above rough estimate, but this is still more than sufficient for the attack to work. As an example, for about 2^{14} random plaintext pairs, we obtain 3 right pairs for differential cryptanalysis and one false alarm. For 2^{20} random plaintext pairs, we obtain 408 right pairs for differential cryptanalysis and 69 false alarms, i.e., about 15%; note that the rough estimate suggests $3 \cdot 10^{-5}/3.8 \cdot 10^{-4} \approx 7.8\%$.

Next, for every possible 6-bit key entering S-box S3 in the fourth round, we computed the Hamming weight profile of the key according to the difference distribution table for that S-box. There are 10 differential pairs which follow our characteristic for S-box S3, and thus the profile distributes these 10 possible input pairs according to the key value as described in the previous section. Note that the profiles only depend on the S-box (namely its difference distribution table and the associated differential pairs) and the key value.

Now taking our real data, we try to match the observed distribution of Hamming weight profiles at the input of S-box S3 in round four with the theoretical profiles we have using a mean square error matching criterion. With 3 right pairs and one false alarm, the right key ends up in second position. However there are many indistinguishable keys at this stage. With as few as 26 right pairs and 4 false alarms (derived from 2^{16} plaintext pairs) the right key ends up in first position and there are only 2 (indistinguishable) keys left. Thus a 20-30 right pairs suffice to recover the 6-bit key element corresponding to S-box S3. Note that if four keys survive at this stage, the overall attack only requires a small extra exhaustive search step. The experiments have been repeated for several keys, with similar or better results.

Now that we recover a few candidates for the first 6-bit key element, we continue with the adjacent S-box S4. Next we change our differential characteristic to get different active S-boxes in round four and recover the whole key piece by piece.

Since the probability of the four-round differential is about $p \approx 3.8 \cdot 10^{-4}$ and there are approximately 15% false alarms, the whole attack requires $\mathcal{O}(1/p) \approx$ 30 000 plaintext pairs with their associated power traces and Hamming weight measurements. Blind differential cryptanalysis completely bypasses any type of random data masking on the four first rounds. The computational complexity of the attack is negligible; it requires only a few seconds on a regular PC.

5 Improvements and Generalizations

So far we have only explained a rather straightforward approach and we have illustrated it with an example as a proof of concept. There are several ways in which our attack can be further optimized and improved. First, some keys are clearly easier to recover than others; we need to analyze this phenomenon in more detail in order to assess the entropy reduction of the key that can be obtained. Second we can optimize the differential characteristics for this type of attack – the example we have used is a good characteristic for a regular differential attack, but it is plausible that we can find characteristics that are better suited for a blind differential attack. Third, the attack could be expanded to taking into account measurement noise and other side channel leakage models (such as leakage of the Hamming weight transitions in the registers). Finally, the attack is independent of the masking technique and only builds on the difference distribution tables of the cipher as well as the Hamming weight profiles of the differential pairs for a given key. Hence it is clear that it can be extended to other

Feistel ciphers (including 2-key and 3-key triple-DES) but also to substitution permutation network ciphers such as the AES.

The technique we describe applies to four or more initial masked rounds, as long as high probability characteristics can be found for this reduced number of rounds of the cipher. Since for all modern block ciphers, resistance against differential cryptanalysis is achieved only after sufficiently many rounds, independent masks should be applied to just as many internal rounds. Power attacks provide enough information about Hamming weights of the intermediate values to put external round-masking techniques at risk when side-channel analysis is combined with differential cryptanalysis. Note that our attacks simply start measuring side-channel information on the first unmasked internal round. Again we stress that our techniques are completely independent from the underlying masking technique and apply to any block cipher for which well-chosen reduced-round differentials with high probability and with several colliding bytes can be found.

6 Conclusion

We have introduced the notion of blind differential cryptanalysis where an attacker uses internal differentials to by-pass outer round masking against power attacks. This technique retrieves the secret key of the target block cipher given only Hamming weight measurements on selected internal values. We therefore invalidate the widely claimed belief that only outer rounds need be protected from power attacks. Our method easily generalizes to different block cipher structures and requires a reasonable amount of plaintext-ciphertext pairs and power measurements. We believe other powerful combinations of side-channel attacks and traditional cryptanalysis will provide for interesting future developments in the area of secure embedded tokens.

Acknowledgements. We would like to thank the anonymous referees for the constructive comments.

References

1. Akkar, M.-L., Bevan, R., Dischamp, P., Moyart, D.: Power Analysis, What Is Now Possible.... In: Okamoto, T. (ed.) ASIACRYPT 2000. LNCS, vol. 1976, pp. 489–502. Springer, Heidelberg (2000)
2. Akkar, M.-L., Bevan, R., Goubin, L.: Two Power Analysis Attacks against One-Mask Methods. In: Roy, B.K., Meier, W. (eds.) FSE 2004. LNCS, vol. 3017, pp. 332–347. Springer, Heidelberg (2004)
3. Akkar, M.-L., Goubin, L.: A Generic Protection against High-Order Differential Power Analysis.. In: Johansson, T. (ed.) FSE 2003. LNCS, vol. 2887, pp. 192–205. Springer, Heidelberg (2003)
4. Biham, E., Shamir, A.: Differential Cryptanalysis of DES-like Cryptosystems. J. Cryptology 4(1), 3–72 (1991)

5. Koç, Ç.K., Naccache, D., Paar, C. (eds.): CHES 2001. LNCS, vol. 2162. Springer, Heidelberg (2001)
6. Chari, S., Jutla, C.S., Rao, J.R., Rohatgi, P.: Towards Sound Approaches to Counteract Power-Analysis Attacks. In: Wiener [24], pp. 398–412
7. Goubin, L., Patarin, J.: DES and Differential Power Analysis (The "Duplication" Method). In: Koç, Ç.K., Paar, C. (eds.) CHES 1999. LNCS, vol. 1717, pp. 158–172. Springer, Heidelberg (1999)
8. Joye, M., Quisquater, J.-J. (eds.): CHES 2004. LNCS, vol. 3156. Springer, Heidelberg (2004)
9. Kaliski Jr., B.S., Koç, Ç.K., Paar, C. (eds.): CHES 2002. LNCS, vol. 2523. Springer, Heidelberg (2003)
10. Kocher, P.C., Jaffe, J., Jun, B.: Differential Power Analysis. In: Wiener [24], pp. 388–397
11. Kunz-Jacques, S., Muller, F., Valette, F.: The Davies-Murphy Power Attack. In: Lee, P.J. (ed.) ASIACRYPT 2004. LNCS, vol. 3329, pp. 451–467. Springer, Heidelberg (2004)
12. Ledig, H., Muller, F., Valette, F.: Enhancing collision attacks. In: Joye and Quisquater [8], pp. 176–190
13. Messerges, T.S.: Securing the AES Finalists Against Power Analysis Attacks. In: Schneier, B. (ed.) FSE 2000. LNCS, vol. 1978, pp. 150–164. Springer, Heidelberg (2001)
14. Messerges, T.S.: Using Second-Order Power Analysis to Attack DPA Resistant Software. In: Paar, C., Koç, Ç.K. (eds.) CHES 2000. LNCS, vol. 1965, pp. 238–251. Springer, Heidelberg (2000)
15. National Institute of Standards and Technology (NIST) FIPS Publication 46-3: Data Encryption Standard (1999)
16. National Institute of Standards and Technology (NIST). FIPS Publication 197: Advanced Encryption Standard (AES) (2001)
17. Ohta, K., Matsui, M.: Differential Attack on Message Authentication Codes. In: Stinson [22], pp. 200–211
18. Osvik, D.A., Shamir, A., Tromer, E.: Cache attacks and countermeasures: The case of AES. In: Pointcheval, D. (ed.) CT-RSA 2006. LNCS, vol. 3860, pp. 1–20. Springer, Heidelberg (2006)
19. Preneel, B., Nuttin, M., Rijmen, V., Buelens, J.: Cryptanalysis of the CFB Mode of the DES with a Reduced Number of Rounds. In: Stinson [22], pp. 212–223
20. Rao, J.R., Sunar, B. (eds.): CHES 2005. LNCS, vol. 3659. Springer, Heidelberg (2005)
21. Schramm, K., Leander, G., Felke, P., Paar, C.: A Collision-Attack on AES: Combining Side Channel- and Differential-Attack. In: Joye and Quisquater [8], pp. 163–175
22. Stinson, D.R. (ed.): CRYPTO 1993. LNCS, vol. 773. Springer, Heidelberg (1994)
23. Walter, C.D., Koç, Ç.K., Paar, C. (eds.): CHES 2003. LNCS, vol. 2779. Springer, Heidelberg (2003)
24. Wiener, M.J. (ed.): CRYPTO 1999. LNCS, vol. 1666. Springer, Heidelberg (1999)

Efficient Implementations of Multivariate Quadratic Systems*

Côme Berbain, Olivier Billet, and Henri Gilbert

France Télécom R&D
38–40, rue du Général Leclerc
92794 Issy les Moulineaux Cedex 9 — France
forname.lastname@orange-ftgroup.com

Abstract. This work investigates several methods to achieve efficient software implementations of systems of multivariate quadratic equations. Such systems of equations appear in several multivariate cryptosystems such as the signature schemes SFLASH, Rainbow, the encryption scheme PMI$^+$, or the stream cipher QUAD. We describe various implementation strategies. These strategies were combined to implement the public computations of three asymmetric schemes as well as the stream cipher QUAD. We conducted extensive benchmarks on our implementations which are exposed in the final section of this paper. The obtained figures support the claim that when some care is taken, multivariate schemes can be efficiently implemented in software.

Keywords: multivariate systems, quadratic equations, efficient software implementation.

1 Introduction

Multivariate cryptography is a quickly expanding research branch of cryptology. Its development, initiated by the seminal work of T. Matsutomo and H. Imai [7,10,11] and J. Patarin [13,14,15], was mainly motivated by the search for alternatives to arithmetic asymmetric cryptosystems such as RSA. Multivariate cryptography exploits the intractability of solving a multivariate system of low degree equations (typically quadratic equations) over a small finite field. Many multivariate asymmetric schemes for encryption, signature, or authentication have been proposed over the past years and a restricted number of them (e.g. the SFLASH and UOV signature schemes [1,8]) have successfully resisted cryptanalysis so far.

The development of multivariate cryptography has recently taken another path with the proposal of a symmetric multivariate algorithm, the stream cipher

* The work described in this paper has been supported by the French Ministry of Research RNRT X-CRYPT project and by the European Commission through the IST Program under Contract IST-2002-507932 ECRYPT.

© Springer-Verlag Berlin Heidelberg 2007

QUAD [2]. This cipher takes advantage of the specific characteristics of multivariate systems of equations in order to provide some provable security properties (an extremely unusual feature in the context of symmetric cryptography) at the expense of a moderate performance penalty.

What renders multivariate cryptography attractive from an implementation point of view is that intractable multivariate problems can be extremely compact. Consequently, the performance of multivariate schemes' implementations is often intermediate between the typical performance of asymmetric schemes and the typical performance of symmetric schemes. But efficient implementations of multivariate schemes have not been systematically investigated so far, and there is a lack of figures to serve as reference when comparing multivariate schemes to other systems in terms of practicality. In the case of the asymmetric multivariate signature scheme SFLASH for instance, considerable optimization efforts [1] were made to produce a highly efficient implementation of the secret key computations and to establish that unlike RSA, the SFLASH signature algorithm can be efficiently embedded in 8-bit smart cards without coprocessor. However there was no effort to optimize the public key computations which are typically done on a server. Another example where optimization takes place on the secret's holder side and does not relate to any public computations is the study of public key generation presented in [17].

The main issue one is faced when implementing multivariate schemes in software is to achieve an efficient computation of (at least apparently) random systems of quadratic equations over a small finite field GF(q). This issue arises for instance in software implementations of the public computations in the setting of asymmetric schemes (for signature verification, encryption, or entity authentication purposes), or for the implementation of symmetric schemes (like in the case of the stream cipher QUAD). Efficient implementations of such multivariate quadratic systems of equations without using any specific structure will thus benefit all multivariate schemes as it is not tight to any particular cryptosystem.

This paper is organized as follows. Section 2 describes several methods that can be used to efficiently implement such generic systems. We discuss the special cases of GF(2), GF(2^4), and GF(2^8) which are in practice the most suitable ground fields in most multivariate cryptosystems, and focus on parameter sizes (like the number of unknowns and the number of polynomials in the system) directly arising from real cryptosystems outlined in Section 3. Section 4 exhibits a comprehensive set of benchmarks showing the performance of our various C ANSI implementations of asymmetric schemes like SFLASH, PMI$^+$, and Rainbow public keys as well as QUAD's internal system of equations. We finally draw our conclusions.

2 Implementation Strategies

This section describes various strategies we investigated in order to efficiently implement the computation of quadratic systems in several cases of cryptographic significance. We specify along these descriptions which strategy seems best suited for a particular setting. The computation of any multivariate system of m quadratic equations in n unknowns over a finite field $\mathrm{GF}(2^p)$ can obviously be split into two phases: generating the value of each of the degree two monomials, and actually computing the value of the output polynomials. While for both steps there are rather academic ways to perform the computations, they have to be tuned to the context of use. All the algorithms described hereafter have the same asymptotic complexity, namely $O(n^2)$ to generate the monomials and $O(mn^2)$ to compute the polynomials. However because we target real cryptographic schemes, the values of m and n lie in some range and as we show in the following it is possible to achieve big speedups. This is especially true when, as is the case in our practical cryptographic examples, the values of n and m are the same order of magnitude as the machine word size w. Moreover, fine tuning the implementation in order to take into account the available amount of L2 cache plays an essential role in the overall efficiency.

2.1 Generating All Monomials

There are many ways to compute the values of all degree two monomials for a given set of n variables over a finite field $\mathrm{GF}(2^p)$. However, their respective efficiency highly depends on the ground field size. We hereafter focus on the natural cases $p \in \{1, 4, 8\}$.

The Naïve Way. The most naïve way to compute a set of monomials is obviously to consider every pair (x_i, x_j) of variables in turn and to generate the corresponding monomial $x_i x_j$. This method is efficient provided one has direct access to each variable, which is the case for instance when working over $\mathrm{GF}(2^8)$. On the opposite, when binary variables are packed in big words, the overhead of accessing the variables is prohibitive.

Rotations. Another intuitive way which may seem particularly attractive in the case of a binary ground field (since the total number of operations is reduced by a factor of the machine word size) is to consider the set of variables as a vector of machine words and to perform w multiplications in parallel using bitwise `and`s between cyclically rotated versions of this vector. However this is not the most efficient strategy over $\mathrm{GF}(2)$ as is shown in the sequel.

Bitslice Multiplications. This strategy can be rather efficient in the case of intermediate ground fields such as $\mathrm{GF}(2^4)$ provided we implement a bitslice multiplication to replace the bitwise multiplications of the binary case. Such a bitslice implementation is described for instance in [9]. It basically requires storing the set of n variables over $\mathrm{GF}(2^4)$ in p vectors of size $\lceil n/w \rceil$ words and performing the computations on these vectors directly, hence avoiding bit level manipulations. (In the existing cryptographic schemes, the number n of variables is about the size w of a machine word, and so the vectors typically consist of a small number of machine words.)

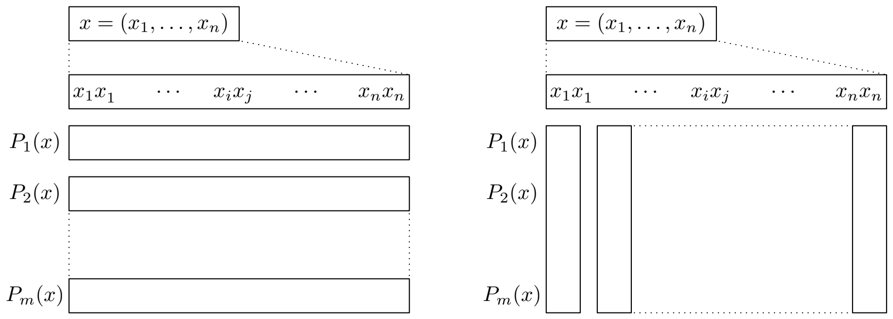


Fig. 1. Left: blocking with bitslice is too slow. Right: using lookup tables.

2.2 Computing the Polynomials

Once all monomials have been generated, one has to compute the value of every polynomial of the equation system. Recall that we are considering polynomials of the special form:

$$P_k(x_1, \ldots, x_n) = \sum_{1 \le i \le j \le n} \alpha^k_{i,j} x_i x_j + \sum_{1 \le l \le n} \beta^k_l x_l + \gamma^k.$$

Given the value of every monomial, a straightforward computation of any polynomial would require $n(n + 1)/2$ field multiplications between coefficients and monomials and the same amount of additions in order to accumulate the result. In this section, we show how to do this more efficiently depending on the context.

Blocking is not Enough. A natural way of implementing the computation of the polynomials is to perform a kind of matrix/vector product, where vector entries holds the value of the monomials and matrix rows represent the value of the polynomials' coefficients just as shown on the left side of Fig. 1. However, even when multiplications are implemented in a bitsliced fashion, this method appears to be rather slow in practice.

Lookup Tables and Field Multiplications. To compute the value of the polynomials one basically has to multiply each monomial with the corresponding coefficient in each polynomial. Of course, such a multiplication is costly and it is worth trying to avoid it; a standard way to do this is by implementing lookup tables. In our case, this strategy amounts to precomputing the contribution of a monomial to *all* polynomials simultaneously, thus saving a factor of m in the number of table lookups. This requires a lookup table with $2^p - 1$ entries of m/w machine words for all of the $n(n + 1)/2$ monomials. (There is no need to store the contribution of zero, and hence there are $2^p - 1$ entries.)

Obviously, the memory required to store all these lookup tables is of crucial importance since they have to fit the processor's cache. In the special case of GF(2^4) with $n = 40$ and $m = 80$ for instance, the amount of space required is 492 K bytes, which fits the L2 cache of most processors. On the opposite, in the special case of $\mathrm{GF}(2^8)$ with $n = 20$ and $m = 40$ the required space now is strictly more than 2 M bytes and will not fit any L2 cache.

To solve this issue, it is possible to split the contribution of any monomial to the polynomials into two parts. This method may be thought of as analogous to extension towers representation of finite fields. Basically, the idea is to perform the multiplication $x \times \alpha$ as $x_{\mathrm{low}} \times \alpha_0 \oplus x_{\mathrm{up}} \times \alpha_1$ where x_{low} and x_{up} are respectively the $p/2$ less significant bits and the $p/2$ most significant bits and α_0 and α_1 are values of $\mathrm{GF}(2^p)$ derived from the value of the coefficient α. This very simple trick dramatically decreases the size of the lookup tables: they have $2^{p/2} - 1$ entries instead of $2^p - 1$. For instance, the previous example with $\mathrm{GF}(2^8)$, $n = 20$ and $m = 40$ now requires 252 K bytes of storage which thus fit most of current processors' L2 cache. A drawback of this technique is that the implementation becomes vulnerable to side channel attacks like cache attack [12].

2.3 The Special Case of GF(2)

There are several benefits of working in the boolean setting. Obvious remarks are that multiplications are readily handled by bitwise ands and that since $x_i^2 = x_i$, we only have to handle homogeneous monomials of degree two. Less obviously, the fact that a monomial now has probability $\frac{3}{4}$ to be zero leads to the optimization described in the first paragraph of this section.

Generating only the Non Zero Monomials. On the average, any variable has probability $\frac{1}{2}$ being zero. Hence any monomial of degree two is zero with probability $\frac{3}{4}$. We can take advantage of this simple fact by first computing the list of indices of non-zero variables, and then generating all pairs of such indices. The number of pairs of non-zero monomials being $n(n+1)/8$ on the average, this significantly decreases the number of lookups to the tables storing the contribution of those monomials to the polynomials and also decreases the overhead during the accumulation process. Moreover, there is no need to extract data at the bit level since all the required information can be discovered with the help of tiny auxiliary lookup tables.

A Differential Trick. To push the previous advantage one step further, the following property appears to be very useful. Every multivariate quadratic polynomial Q has the property that for any $\underline{x} = (x_1, \ldots, x_n)$ and any $\underline{\delta} = (\delta_1, \ldots, \delta_n)$, $Q(\underline{x}) \oplus Q(\underline{x} \oplus \underline{\delta}) = L_{\underline{\delta}}(\underline{x})$, where $L_{\underline{\delta}}(\underline{x})$ is linear with respect to x for any fixed value of $\underline{\delta}$. We now show how this fact can be used to amplify the cost saving achieved in the previous paragraph. Indeed, in order to compute a system $S(\underline{x})$ of multivariate quadratic equations, we first precompute the corresponding linear system $L_{\underline{\eta}}$ in $\underline{x}$ corresponding to the specific $\underline{\eta} = (1, \ldots, 1)$. For instance, the k-th row of $L_{\underline{\eta}}$ is given by:

$$P_k(\underline{x}) \oplus P_k(\underline{x} \oplus \underline{\eta}) = \sum_{1 \le i \le j \le n} \alpha_{i,j}^k \left(x_i \oplus x_j \oplus 1\right) + \mathrm{cst}.$$

Then, depending on the weight of $\underline{x}$, we either perform the computation of $S(\underline{x})$ by evaluating $S(\underline{x})$ in case the Hamming weight of $\underline{x}$ is lower or equal to $\frac{n}{2}$, or we actually compute the mathematically equivalent function $S(\underline{x} \oplus \underline{\eta}) \oplus L_{\underline{\eta}}(\underline{x})$ in case the Hamming weight of $\underline{x}$ is bigger than $\frac{n}{2}$.

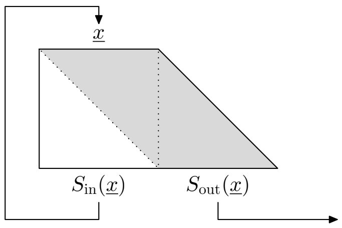
This differential trick can be pushed a little bit further with the use of an error correcting code. Considering a binary linear code, it is possible to compute $S(\underline{x})$ with $S(\underline{x} \oplus \underline{\eta}) \oplus L_{\underline{\eta}}(\underline{x})$, where $\underline{x} \oplus \underline{\eta}$ has a hamming weight lower than some value d determined by the code. However the counterpart is that we have to store for each of the code word a linear system and for each x we have to find the closest code word. For a small d, the number of code words becomes large and the memory required becomes prohibitive.

3 Some Multivariate Cryptosystems

We briefly describe in this section the multivariate schemes we implemented and for which we made extensive benchmarks on a variety of architectures.

3.1 QUAD Stream Cipher

The stream cipher QUAD is a practical stream cipher with some provable security which was introduced [2] by C. Berbain, H. Gilbert, and J. Patarin. The security proof reduces the distinguishability of the keystream generated by QUAD to the hard problem of solving randomly generated quadratic systems over finite field GF(2).


$\underline{x}$
$S_{\text{in}}(\underline{x})$
$S_{\text{out}}(\underline{x})$

The keystream generation makes use of two systems S_{in} and S_{out} of multivariate quadratic equations both sharing the same n unknowns over GF(q), as is described by the figure on the left. The first system S_{in} is used to update the internal state and thus contains n equations, whereas the second system S_{out} produces the keystream and contains $m - n$ equations. As explained in [2], the quadratic systems S_{in} and S_{out}, though randomly generated, are both publicly known.

We will restrict our study to the conservative case $m = 2n$, that is both systems S_{in} and S_{out} contain n quadratic equations in the n bits of the internal state. Given an n-bit internal state $\underline{x} = (x_1, \ldots, x_n)$, the generation amounts to iterating the following steps:

– compute $\big(S_{\text{in}}(\underline{x}), S_{\text{out}}(\underline{x})\big) = \big(Q_1(\underline{x}), \ldots, Q_{2n}(\underline{x})\big)$, from the internal state $\underline{x}$;
– output the sequence $S_{\text{out}}(\underline{x}) = \big(Q_{n+1}(\underline{x}), \ldots, Q_{2n}(\underline{x})\big)$ of n keystream bits;
– update the internal state $\underline{x}$ with the sequence $S_{\text{in}}(\underline{x}) = \big(Q_1(\underline{x}), \ldots, Q_n(\underline{x})\big)$.

The parameters recommended by the authors are $m = 2n$ and $n = 160$ over field GF(2). We made implementations for this parameters and over fields

GF(2^4) and GF(2^8). This allows us to study the impact of changing the size of the field over the performances. However there is no security arguments over fields larger than GF(2), since the security proof of [2] is only done over GF(2). In particular over GF(2^8) the number of variables becomes two small to provides a security of 2^{80}.

3.2 SFLASH Signature Scheme

The signature scheme SFLASH proposed in [1] (sometime referred to as SFLASH v2) was selected as a finalist of the NESSIE project and has resisted attacks so far. It is build around the C^* scheme of T. Matsutomo and H. Imai [7] with $\mathcal{K} = \mathrm{GF}(2^7)$ as ground field, but where some of the public equations have been removed. The secret key consists of two invertible linear transformations L_1 and L_2 defined over $\mathcal{K}^{37}$ together with an isomorphism $\varphi : \mathcal{K}^{37} \to \mathcal{L}$, where $\mathcal{L}$ is an extension of degree 37 of $\mathcal{K}$ defined by $\mathcal{L} = \mathcal{K}[y]/(y^{37} + y^{12} + y^{10} + y^2 + 1)$.

The verification algorithm recognizes $\sigma = (\sigma_1, \ldots, \sigma_{37})$ as the signature of a message $\mu = (\mu_1, \ldots, \mu_{26})$ if and only if equation

$$\mu = L_2\left(\varphi^{-1}\left[\varphi\big[L_1(\sigma)\big]^{128^{11}+1}\right]\right)$$

holds. Since the exponentiation $x \mapsto x^{128^{11}+1}$ is $\mathcal{K}$-quadratic, the public key which gives the values of $\mu_1, \ldots, \mu_{26}$ in terms of the variables $\sigma_1, \ldots, \sigma_{37}$ is nothing but a system of 26 multivariate quadratic equations in 37 unknowns over the finite field GF(2^7).

3.3 Rainbow Signature Scheme

Rainbow is a signature scheme proposed in [5] which is intended to rival SFLASH. However from a security point of view, Rainbow has been recently broken [3]. The public key of Rainbow consists of a set of 27 multivariate quadratic polynomials $\bar{F}_1, \ldots, \bar{F}_{27}$ in 33 unknowns over the finite field GF(2^8). The general problem of solving such a set of multivariate polynomials being hard, those polynomials are constructed in a special way using the UOV construction several times in an embedded manner to build a trapdoor.

The Rainbow signature scheme has four UOV layers and parameters $v_1 = 6$, $v_2 = 12$, $v_3 = 17$, $v_4 = 22$, and $v_5 = 33$. Each layer k is made of a as a set $\mathcal{P}_k$ of polynomials of the special form:

$$\sum_{1 \le j \le v_k < i \le v_{k+1}} \alpha_{i,j}\, x_i x_j + \sum_{1 \le i,j \le v_k} \beta_{i,j}\, x_i x_j + \sum_{1 \le i \le v_{k+1}} \gamma_i\, x_i + \delta.$$

Such polynomials are Oil and Vinegar polynomials since no monomial of degree two has both variables coming from the set $O_k = \{x_{v_k+1}, x_{v_k+2}, \ldots, x_{v_{k+1}}\}$, whereas there are monomials of degree two where both variables come from the set $V_k = \{x_1, \ldots, x_{v_k}\}$. Hence, variables from the set O_k are called oil variables of layer k, and variables from the set V_k are called vinegar variables of layer k.

The first layer of Rainbow is made of 6 polynomials randomly chosen from $\mathcal{P}_1$, the second layer is made of 5 polynomials randomly chosen from $\mathcal{P}_2$, the third layer is made of 5 polynomials randomly chosen from $\mathcal{P}_3$, and the last layer is made of 11 polynomials randomly chosen from $\mathcal{P}_4$. So the resulting internal map of Rainbow is:

$$\begin{array}{rrcl} \mathcal{F}: & \mathrm{GF}(2^8)^{33} & \longrightarrow & \mathrm{GF}(2^8)^{27}, \\ & (x_1, \ldots, x_{33}) & \longmapsto & \big(F_1(x_1, \ldots, x_{33}), \ldots, F_{27}(x_1, \ldots, x_{33})\big). \end{array}$$

Once again, the public key $\bar{\mathcal{F}}$ is obtained by applying to $\mathcal{F}$ a randomly chosen change of variables L_1 of $\mathrm{GF}(2^8)^{33}$ as well as a bijective linear output mixing L_2 of $\mathrm{GF}(2^8)^{27}$, eventually obtaining the multivariate quadratic system:

$$\bar{\mathcal{F}}(z_1, \ldots, z_{33}) = L_2 \circ \mathcal{F} \circ L_1(z_1, \ldots, z_{33}).$$

3.4 PMI$^+$ Encryption Scheme

The PMI$^+$ [4] is a doubly perturbed C^* scheme. As with C^*, there is an exponentiation $F : x \mapsto x^{2^4+1}$ defined over a finite field $\mathrm{GF}(2^{84})$, and two invertible linear transformations L_1 and L_2 respectively defined over $\mathrm{GF}(2^{84})$ and $\mathrm{GF}(2^{98})$. Let us denote by $(f_1, \ldots, f_{84})$ the binary component of the quadratic system in the 84 binary unknowns defined by F.

Additionally, randomly chose 14 quadratic polynomials $q_1, \ldots, q_{14}$ in the 84 unknowns $x_1, \ldots, x_{84}$ defined over $\mathrm{GF}(2)$ and a linear application Z of rank 6 from the 84 unknowns to six binary variables $z_1, \ldots, z_6$. Also randomly chose 98 quadratic polynomials $\rho_1, \ldots, \rho_{98}$ in 6 binary unknowns.

The public key is given by the expansion of the following composition:

$$L_2 \circ (f_1 + \rho_1 \circ Z, \ldots f_{84} + \rho_{84} \circ Z, \ldots q_1 + \rho_{85} \circ Z, \ldots q_{14} + \rho_1 \circ Z) \circ L_1,$$

which is a multivariate quadratic system of 98 equations in 84 unknowns defined over $\mathrm{GF}(2)$.

4 Implementations and Performance Results

This section exposes the benchmarks we conducted on our various implementations of the multivariate cryptosystems presented in the previous section. These benchmarks were done on several computer architectures. We used a modified version of the eSTREAM Testing Framework made by C. de Cannière [6] to evaluate the performance of our implementations when compiled with different compilers and compiling options. We mostly used compilers `gcc-4`, `gcc-3.4`, `gcc-3.3`, and `gcc-2.95`, although we also used Intel's `icc` compiler where supported. The following lists our set of machines together with a description of the processor installed:

name	vendor	processor	frequency	L2 cache
M1	Intel	Pentium 4	2505 MHz	512 kB
M2	Intel	Pentium M	1862 MHz	2048 kB
M3	Intel	Xeon	2784 MHz	512 kB
M4	AMD	Opteron	2197 MHz	1024 kB
M5	AMD	AMD64	1790 MHz	512 kB
M6	AMD	Athlon XP	2162 MHz	512 kB
M7	Power PC	G3	900 MHz	512 kB

All our implementations are written in ANSI C. Of course it is possible to improve the efficiency of these implementations by writing assembly code. However using generic code makes it possible to compare the different architectures and to evaluate cache effects.

For all versions of QUAD and PMI$^+$, speed figures are given in cycles/byte and in Mbits/second, since these are ciphers. For the SFLASH and Rainbow signature schemes, speed is given in cycles/byte and we also give the overall time needed to verify the signature.

4.1 Practical Implementations of QUAD over GF(2)

Our fastest implementation over GF(2) uses two techniques described in the previous sections: we generate only the non-zero monomials and use the differential trick. Notice that since we implement a quadratic system of 320 polynomials in 160 unknowns the total amount of storage required is about 518 K bytes so it explains the penalty on machines with 512 K bytes L2 cache.

Table 1. Speed in cycles/byte

version	M1	M2	M3	M4	M5	M6	M7
32 bit	7057	3746	4600	**2930**	3205	4866	4983
64 bit				**2081**	2636		

Table 2. Speed in Mbits/second

version	M1	M2	M3	M4	M5	M6	M7
32 bit	2.83	3.98	4.84	**6.00**	4.47	3.55	1.44
64 bit				**8.45**	5.43		

4.2 Practical Implementations of QUAD over GF(2^4)

We made five different implementations of QUAD defined over GF(2^4) with 40 unknowns and 80 polynomials. Each of these variants uses the technique of

Table 3. Speed in cycles/byte

monomials	tables	version	M1	M2	M3	M4	M5	M6	M7
naïve	2 luts	32 bit	2526	2364	2604	2134	2149	2576	**1010**
naïve	2 luts	64 bit				**1617**	1732		
naïve	1 lut	32 bit	2390	1395	1704	**1157**	1190	1546	1419
naïve	1 lut	64 bit				**994**	1639		
rotation	2 luts	32 bit	3468	2360	3452	2111	2154	2471	**977**
rotation	2 luts	64 bit				**921**	1335		
rotation	1 lut	32 bit	2858	1357	2014	**1139**	1192	1514	1435
rotation	1 lut	64 bit				**921**	1359		
bitslice	4 luts	32 bit	1906	1204	1849	1003	990	1257	**874**
bitslice	4 luts	64 bit				**745**	885		

Table 4. Speed in Mbits/second

monomials	tables	version	M1	M2	M3	M4	M5	M6	M7
naïve	2 luts	32 bit	7.93	6.30	**8.55**	8.24	6.66	6.71	7.13
naïve	2 luts	64 bit				**10.87**	8.27		
naïve	1 lut	32 bit	8.38	10.68	13.07	**15.19**	12.03	11.19	5.07
naïve	1 lut	64 bit				**17.68**	8.74		
rotation	2 luts	32 bit	5.78	6.31	6.45	**8.33**	6.65	7.00	7.37
rotation	2 luts	64 bit				**19.08**	10.73		
rotation	1 lut	32 bit	7.01	10.98	11.06	**15.43**	12.01	11.42	5.02
rotation	1 lut	64 bit				**19.08**	10.54		
bitslice	4 luts	32 bit	10.51	12.37	12.05	**17.52**	14.46	13.76	18.24
bitslice	4 luts	64 bit				**23.59**	16.18		

precomputing the contribution each monomial to all polynomials described before but with either one, two, or four tables. Implementations also have distinct monomial generation strategies.

The storage required by the coefficients of the system is about 32 K bytes. Using only one lookup table, we need to store $2^4 - 1$ times 32 K bytes, that is 492 K bytes. This value is quite close to the amount of L2 cache on some machines and thus we also implemented a version with two lookup tables. Using two lookup tables requires storing $2^2 - 1$ times 32 K bytes that is about 197 K bytes. Experimental results show that using one table is always better except on the Power PC.

For the first two implementations, we chose to generate the monomials the naïve way, and used either one or two lookup tables. For the third and fourth implementations, we used the rotation technique, and either one or two lookup tables. The last implementation, which is the fastest, uses bitslice multiplication to generate the monomials and four lookup tables of 32 K bytes, which corresponds to the contribution of each of the four bits of any monomial.

4.3 Practical Implementations of QUAD over GF(2^8)

We implemented two variants. Both of them share the monomial/coefficient multiplication precomputation technique. Since the coefficient set can definitely not be stored 255 times (it would require more than 2 M bytes), we store two lookup tables of size 25 K bytes instead.

In the first variant, we use variables rotation to generate the monomials, while in the second variant we use the naïve way to generate the monomials. Since all variables are 8-bit values, we have direct access to them which explains the fact that the naïve technique is competitive.

Table 5. Speed in cycles/byte

version	M1	M2	M3	M4	M5	M6	M7
Naive 32 bit	862	618	883	**530**	560	699	770
Naive 64 bit				**417**	464		
Rotation 32 bit	978	704	983	603	622	775	**493**
Rotation 64 bit				**497**	546		

Table 6. Speed in Mbits/second

version	M1	M2	M3	M4	M5	M6	M7
Naive 32 bit	23.25	24.10	25.22	**33.16**	25.57	24.74	9.35
Naive 64 bit				**42.15**	30.86		
Rotation 32 bit	20.49	21.16	22.66	**29.14**	23.02	22.31	14.60
Rotation 64 bit				**35.36**	26.23		

Thus, on processor M4, our implementation of QUAD over GF(2^8) achieves a throughput of 4.15 M bytes per second on the 32-bit platform and 5.27 M bytes per second on the 64-bit platform.

4.4 Practical Implementations of SFLASH

As described in the previous section, verifying an SFLASH signature can be thought of as evaluating a randomly chosen system of 26 quadratic polynomials in 37 unknowns over the finite field GF(2^7). We implemented two variants: the first one uses the naïve technique to generate the monomials, while the second one uses the rotation technique. Both of them use the precomputation of the contribution of

Table 7. Speed in cycles/byte

version	M1	M2	M3	M4	M5	M6	M7
Naive 32 bit	934	954	1046	807	848	1072	**448**
Naive 64 bit				**253**	**253**		
Rotation 32 bit	1126	863	1124	726	756	981	**452**
Rotation 64 bit				**266**	270		

Table 8. Time to verify a signature in μs

version	M1	M2	M3	M4	M5	M6	M7
Naive 32 bit	13.79	18.94	13.89	**13.58**	17.51	18.35	18.41
Naive 64 bit				**4.25**	5.22		
Rotation 32 bit	16.63	17.14	14.93	**12.22**	15.64	16.78	18.57
Rotation 64 bit				**4.47**	5.58		

the monomials to the polynomials with the help of two lookup tables. The speed measurements are obtained by verifying many signatures for the same public key.

The following table also give the overall time required to verify an SFLASH signature on the different processors:

It may be of interest to compare those figures with the `openssl` implementation of RSA-1024 and RSA-2048 signature verification. On processor M1, those implementation respectively require 2.15 ms and 3.80 ms to verify a signature. Our implementation of SFLASH on the same computer is about 150 times faster than RSA-1024.

4.5 Practical Implementations of Rainbow

Just as with SFLASH, verifying a Rainbow signature can be thought of as evaluating a randomly chosen system of 27 quadratic polynomials in 33 unknowns defined over the ground field GF(2^8). Our implementation follows the same strategy as the fastest implementation of QUAD over GF(2^8). The speed measurements are obtained by verifying many signatures for the same public key. We also give the overall time needed to verify a signature on the different processors.

Table 9. Speed in cycles/byte

version	M1	M2	M3	M4	M5	M6	M7
Naive 32 bit	1016	882	1068	719	750	989	**479**
Naive 64 bit				262	**259**		

Table 10. Time to verify a signature in μs

version	M1	M2	M3	M4	M5	M6	M7
Naive 32 bit	13.37	15.63	12.66	**10.79**	13.82	15.10	17.57
Naive 64 bit				**3.93**	4.77		

4.6 Practical Implementations of PMI$^+$

The multivariate quadratic system underlying PMI$^+$ is made of 98 polynomials in 84 unknowns over GF(2). Our implementation uses the same techniques as QUAD's implementation over GF(2), but since the numbers of variables and of polynomials are much smaller, the implementation is much faster. Additionally, the system only requires 100 K bytes of storage, so that there is no cache effect.

Table 11. Speed in cycles/byte

version	M1	M2	M3	M4	M5	M6	M7
32 bit	1443	1259	1440	909	921	1180	**589**
64 bit				**768**	821		

Table 12. Speed in Mbits/second

version	M1	M2	M3	M4	M5	M6	M7
32 bit	13.89	11.83	15.47	**19.34**	15.55	14.66	12.22
64 bit				**22.89**	17.44		

5 Conclusion

In this paper we presented several methods for efficiently implementing multivariate quadratic systems of equations. We applied these techniques to implement several multivariate cryptosystems: SFLASH, Rainbow, PMI$^+$, and the stream cipher QUAD. Our implementations were run on a large variety of architectures and appear to be quite efficient. A critical parameter when it comes to optimizations is the size of L2 cache available. Consequently, new processors with larger L2 cache leave more room for further improvement.

References

1. Akkar, M.-L., Courtois, N.T., Goubin, L., Duteuil, R.: A Fast and Secure Implementation of SFLASH. In: Desmedt, Y.G. (ed.) PKC 2003. LNCS, vol. 2567, pp. 267–278. Springer, Heidelberg (2002)
2. Berbain, C., Gilbert, H., Patarin, J.: QUAD: a Practical Stream Cipher with Provable Security. In: Vaudenay, S. (ed.) EUROCRYPT 2006. LNCS, vol. 4004, Springer, Heidelberg (2006)
3. Billet, O., Gilbert, H.: Cryptanalysis of Rainbow. In: De Prisco, R., Yung, M. (eds.) SCN 2006. LNCS, vol. 4116, Springer, Heidelberg (2006)
4. Ding, J., Gower, J.E.: Inoculating multivariate schemes against differential attacks. In: Yung, M., Dodis, Y., Kiayias, A., Malkin, T.G. (eds.) PKC 2006. LNCS, vol. 3958, pp. 290–301. Springer, Heidelberg (2006)
5. Ding, J., Schmidt, D.: Rainbow, a New Multivariable Polynomial Signature Scheme. In: Ioannidis, J., Keromytis, A.D., Yung, M. (eds.) ACNS 2005. LNCS, vol. 3531, pp. 164–175. Springer, Heidelberg (2005)
6. ECRYPT. eSTREAM: ECRYPT Stream Cipher Project, IST-2002-507932 Accessed September 29, 2005 (2005), Available at `http://www.ecrypt.eu.org/stream/`
7. Imai, H., Matsumoto, T.: Algebraic Methods for Constructing Asymmetric Cryptosystems. In: Calmet, J. (ed.) Algebraic Algorithms and Error-Correcting Codes. LNCS, vol. 229, pp. 108–119. Springer, Heidelberg (1986)
8. Kipnis, A., Patarin, J., Goubin, L.: Unbalanced Oil and Vinegar Signature Schemes. In: Stern, J. (ed.) EUROCRYPT 1999. LNCS, vol. 1592, pp. 206–222. Springer, Heidelberg (1999)

9. Matsui, M.: How Far Can We Go on the x64 Processors? In: Robshaw, M. (ed.) FSE 2006. LNCS, vol. 4047, Springer, Heidelberg (2006)
10. Matsumoto, T., Imai, H.: A Class of Asymmetric Cryptosystems Based on Polynomials over Finite Rings. In: IEEE International Symposium on Information Theory, pp. 131–132 (1983)
11. Matsumoto, T., Imai, H.: Public Quadratic Polynominal-Tuples for Efficient Signature-Verification and Message-Encryption. In: Günther, C.G. (ed.) EUROCRYPT 1988. LNCS, vol. 330, pp. 419–453. Springer, Heidelberg (1988)
12. Osvik, D.A., Shamir, A., Tromer, E.: Cache attacks and countermeasures: The case of aes. In: Pointcheval, D. (ed.) CT-RSA 2006. LNCS, vol. 3860, pp. 1–20. Springer, Heidelberg (2006)
13. Patarin, J.: Cryptoanalysis of the Matsumoto and Imai Public Key Scheme of Eurocrypt '88. In: Coppersmith, D. (ed.) CRYPTO 1995. LNCS, vol. 963, pp. 248–261. Springer, Heidelberg (1995)
14. Patarin, J.: Asymmetric Cryptography with a Hidden Monomial. In: Koblitz, N. (ed.) CRYPTO 1996. LNCS, vol. 1109, pp. 45–60. Springer, Heidelberg (1996)
15. Patarin, J.: Hidden Fields Equations (HFE) and Isomorphisms of Polynomials (IP): Two New Families of Asymmetric Algorithms. In: Maurer, U.M. (ed.) EUROCRYPT 1996. LNCS, vol. 1070, pp. 33–48. Springer, Heidelberg (1996)
16. Patarin, J., Goubin, L., Courtois, N.T.: C*-+ and HM: Variations Around Two Schemes of T. Matsumoto and H. Imai. In: Ohta, K., Pei, D. (eds.) ASIACRYPT 1998. LNCS, vol. 1514, pp. 35–49. Springer, Heidelberg (1998)
17. Yang, B.-Y., Chen, J.-M., Chen, Y.-H.: TTS: High-Speed Signatures on a Low-Cost Smart Card. In: Joye, M., Quisquater, J.-J. (eds.) CHES 2004. LNCS, vol. 3156, p. 371. Springer, Heidelberg (2004)

Unbridle the Bit-Length of a Crypto-coprocessor with Montgomery Multiplication

Masayuki Yoshino, Katsuyuki Okeya, and Camille Vuillaume

Hitachi, Ltd., Systems Development Laboratory, Kawasaki, Japan
{m-yoshi,ka-okeya,camille}@sdl.hitachi.co.jp

Abstract. We present a novel approach for computing $2n$-bit Montgomery multiplications with n-bit hardware Montgomery multipliers. Smartcards are usually equipped with such hardware Montgomery multipliers; however, due to progresses in factoring algorithms, the recommended bit length of public-key schemes such as RSA is steadily increasing, making the hardware quickly obsolete. Thanks to our double-size technique, one can re-use the existing hardware while keeping pace with the latest security requirements. Unlike the other double-size techniques which rely on *classical* n-bit modular multipliers, our idea is tailored to take advantage of n-bit *Montgomery* multipliers. Thus, our technique increases the perenniality of existing products without compromises in terms of security.

Keywords: Montgomery multiplication, RSA, crypto-coprocessor, smartcard.

1 Introduction

The algorithm proposed by Montgomery to calculate modular multiplications [6], usually referred to as "Montgomery multiplication" technique, is extensively used in practical implementations of public-key cryptosystems such as RSA [10]. In particular, Montgomery multiplications are not affected by delays which are commonly introduced by carries in other strategies for computing modular multiplications. As a consequence, Montgomery's approach is very effective for high-performance hardware implementations of modular multiplications. Low-end devices such as smartcards can benefit from crypto-coprocessors implementing Montgomery multiplications [7], which can drastically reduce the time necessary to encrypt and decrypt data, or sign and verify signatures. But such hardware accelerators suffer from an important restriction: their operand size is limited [8]. Now, because of progresses in integer factorization [11], official security institutions are slowly but surely moving their recommendation from 1024-bit to 2048-bit key sizes for RSA; unfortunately, the latter bit length is not supported by many crypto-coprocessors.

This problem has motivated many studies for developing double-size modular multiplication techniques using single-size hardware multipliers. On the one hand, thanks to the Chinese Remainder Theorem, *private* computations (decryption or signature generation) require only single-size multiplications for computing a double-size decryption or signature generation [9]. On the other hand, in the case of *public* computations, the Chinese Remainder Theorem is of no help, and double-size modular multiplications are needed.

Paillier initiated the work on double-size multiplications [8], and showed how to efficiently compute a kn-bit classical modular multiplication with n-bit classical modular multiplication units. Later, Fischer et al. optimized Paillier's scheme for the $2n$-bit case [2]. Finally, Chevallier-Mames et al., also concentrating on the case of $2n$-bit multiplications, showed further improvements in the general case and when the modulus has a special form [1]. We note a recurring problem in these techniques: they are based on *classical* modular multipliers, and as such, do not concern about taking high-performance of *Montgomery* multipliers which usually equip smartcards. This is a serious limitation of these techniques, which cannot fully take advantage of the available hardware.

In this paper, we propose a technique for computing $2n$-bit *Montgomery* multiplications with n-bit *Montgomery* multiplication units. Firstly, we define the notion of quotients of n-bit Montgomery multiplications; indeed, such quotients are necessary to calculate $2n$-bit Montgomery remainders. We consider two types of settings, and in each case, propose efficient solutions to compute the quotients. In the first settings, we assume that we have to re-use an existing n-bit Montgomery multiplier, and that we cannot modify it. In this case, we show how to emulate the calculation of the quotient in software with two calls to the n-bit Montgomery multiplier. In the second settings, the modification of the hardware Montgomery multiplier is allowed, but still restricted to n-bit operands. We explain how to modify the circuitry with minimal changes in order to calculate the quotients along with their remainders. In addition to Montgomery quotients, our double-size technique also requires a new representation of $2n$-bit integers, tailored for a better use of Montgomery multipliers. Indeed, to satisfy the requirements of the Montgomery multipliers, the moduli must be odd and greater than 2^{n-1}. Our representation does not only achieve this, but also allows an efficient conversion from/to the standard binary representation of integers.

As a result, thanks to our double-size technique, one can compute $2n$-size Montgomery multiplications with available n-bit hardware Montgomery multipliers, allowing the current generation of crypto-coprocessor to survive the shift towards higher security and longer key lengths.

The rest of this paper is organized as follows. In Section 2, we review previous double-size technique based on classical modular multiplications [2,1]. In Section 3, we introduce the Montgomery multiplications and put their limitations in evidence. In Section 4, we explain our idea for computing $2n$-bit Montgomery multiplications with n-bit Montgomery multipliers. Since our technique requires the n-bit quotients of the Montgomery multiplications, in Section 5 we introduce two approaches to calculate these quotients; the first is well-suited for software implementations, and the second is for hardware implementations.

Section 6 shows experimental results and introduces some practical issues. Finally, we conclude with Section 7.

Notation
We let n denote the operand-size of Montgomery/classical modular multiplication units. We also let capital letters: A, B, N and M denote $2n$-bit integers, and small letters denote the others, such as n-bit integers.

2 Known Double Size Techniques

Low-end devices such as smartcards are equipped with crypto-coprocessors to calculate modular multiplications; however, such hardware accelerators have a strict restriction: their operand size is limited. Recently, because of progresses in integer factorization [11], official security institutions are changing their recommended key-length for RSA from 1024-bit to 2048-bit. Now, this problem has motivated many studies for developing double-size modular multiplication using single-size hardware multipliers and only public information [3].

Paillier first initiated the work on double size-multiplications [8], and showed how to compute a kn-bit classical modular multiplication with n-bit classical modular multiplication units and public information. Later, Fischer et al. [2] optimized Paillier et al.'s scheme for the $2n$-bit case. Finally, Chevallier-Mames et al. [1] showed further improvements in the case of $2n$-bit multiplications, too.

This section introduces works about schemes of Fischer et al. and not Chevallier-Mames et al., because in Section 4 we will propose our double size technique which modifies Fischer et al.'s one.

2.1 Fischer et al.'s Schemes

In this subsection, we introduce the work of Fischer et al. [2]: how to compute double-size *classical* modular multiplication with single-size *classical* modular multiplication units. Their double-size technique requires not only remainders but also quotients of n-bit multiplication to build a $2n$-bit remainder.

The equation $xy - q_c w = r_c$ shows the relation between the product of two integers x and y, the modulus w, the quotient q_c and the remainder r_c in the case of classical modular multiplications. The basic idea of classical modular multiplications is to subtract the modulus w from the most significant bit of product xy, q_c times until the product becomes less than modulus w; thus, digits of the product are eliminated from left to right.

Fischer et al. assumed that the instruction $\mathsf{MultModDiv}$ is available, where $\mathsf{MultModDiv}$ computes the remainder and the quotient of n-bit classical modular multiplications. For n-bit positive integers x, y and w, $\mathsf{MultModDiv}(x, y, w) = (q_c,\ r_c)$ with $q_c = \lfloor (xy)/w \rfloor$ and $r_c = xy \pmod{w}$.

The $\mathsf{MultModDiv}$ instruction is a natural extension of the classical modular multiplication. If the classical modular multiplications is implemented in hardware, the $\mathsf{MultModDiv}$ instruction can be emulated with two calls to the multiplier, or by changing the hardware of the multiplier only a little.

Algorithm 1. Fischer et al.'s algorithm

INPUT: $A = a_1 2^n + a_0$, $B = b_1 2^n + b_0$, $N = n_1 2^n + n_0$ with $0 \le A, B < N$, $2^{2n-1} < N < 2^{2n}$;
OUTPUT: $AB \pmod N$;

```
1. $(q_1, r_1) = \mathsf{MultModDiv}(b_1, 2^n, n_1)$
2. $(q_2, r_2) = \mathsf{MultModDiv}(q_1, n_0, 2^n)$
3. $(q_3, r_3) = \mathsf{MultModDiv}(a_1, r_1 - q_2 + b_0, n_1)$
4. $(q_4, r_4) = \mathsf{MultModDiv}(a_0, b_1, n_1)$
5. $(q_5, r_5) = \mathsf{MultModDiv}(q_3 + q_4, n_0, 2^n)$
6. $(q_6, r_6) = \mathsf{MultModDiv}(a_1, r_2, 2^n)$
7. $(q_7, r_7) = \mathsf{MultModDiv}(a_0, b_0, 2^n)$
8. **Return** $(r_3 + r_4 - q_5 - q_6 + q_7)2^n + (r_7 - r_6 - r_5)$
```

We introduce an algorithm proposed by Fischer et al. to compute $2n$-bit classical modular multiplication ($AB \mod N$). Given $2n$-bit integers A, B, N, where $0 \le A$, $B < N$. First, every $2n$-bit integers are divided into n-bit integers that can be handled by the MultModDiv instruction. $A = a_1 2^n + a_0$, $B = b_1 2^n + b_0$, $N = n_1 2^n + n_0$. The equation $n_1 2^n \equiv -n_0 \pmod N$ holds thanks to the above transformation. Their proposed algorithm derives from the equation $n_1 2^n \equiv -n_0 \pmod N$, and is described in Algorithm 1.

3 Montgomery Multiplications

Montgomery multiplications, which are based on a technique to calculate modular multiplication proposed by Montgomery [6], are widely used in practical implementations of public-key cryptosystems, such as RSA. Montgomery multiplications are very suitable for high-performance hardware implementations of modular multiplications, which are typically one of the most expensive operations in public-key cryptosystems. Most low-end devices such as smart cards have crypto-coprocessors implementing Montgomery multiplications to encrypt and decrypt data, or sign and verify signatures.

In this section, we introduce Montgomery multiplications and describe the problems that occur when one tries to extend known double size technique to n-bit Montgomery multipliers.

3.1 Montgomery Multiplication Algorithm

The basic idea of Montgomery multiplications is to replace expensive divisions by cheaper multiplications and additions in computations. For n-bit integers x, y, w, $0 \le x, y < w$, and gcd(w, m)=1, the Montgomery multiplication algorithm outputs the remainder r, $r := xym^{-1} \pmod w$ where m is called *Montgomery constant* and m^{-1} is the inverse of m modulo w. The value $m = 2^n$ is widely

Algorithm 2. Montgomery multiplication algorithm

INPUT: x, y, w with $0 \leq x, y < w, m = 2^n$, $\gcd(w, m) = 1$ and $w' = -w^{-1} \mod 2$;
OUTPUT: r;

```
1. $r \leftarrow 0$
2. For $i$ from 0 to $(n-1)$ do the following:
   (a) $u_i \leftarrow (r_0 + x_i y_0)w' \mod 2$
   (b) $r \leftarrow (r + 2r_i + u_i w)/2$
3. If $r \geq w$ then $r \leftarrow r - w$
4. **Return** $r$
```

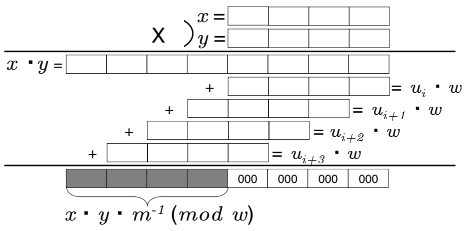


Fig. 1. Basic Idea of Montgomery Multiplication

used in practice because reduction modulo m and division by m are both intrinsically fast operations [5]. Thus the Montgomery algorithm is faster than classical modular multiplication.

Figure 1 illustrates the principle of Montgomery multiplications. Unlike classical modular multiplication techniques, Montgomery multiplications add the modulus w to the product xy from the least significant bit (that is, from right to left), and save the remainder r in the most significant side.

3.2 Problems of Previous Techniques

The schemes proposed by Fischer et al. [2] and Chevallier-Mames et al. [1] show how to compute a $2n$-bit *classical* modular multiplication with n-bit *classical* modular multiplication units. But their schemes cannot take advantage of high-performance *Montgomery* multipliers for the two following reasons:

Problem 1: Quotient
Their double-size techniques require not only n-bit remainders but also n-bit quotients of multiplications to construct $2n$-bit remainders. However, there is no notion of quotient of Montgomery multiplications.

Problem 2: Modulus
The moduli of Montgomery multipliers must be odd because they are restricted to be coprime to Montgomery constants. However, their double-size techniques

allow even moduli. For example, in Algorithm [1], they set upper n-bit value n_1 as modulus, but n_1 can be even.

4 New Double Size Techniques

We propose a new scheme for computing $2n$-bit Montgomery multiplications with the existing coprocessors. On the one hand, Montgomery multiplications are widely implemented on coprocessors for public-key cryptosystems, and produce high-performance hardware modular multiplications for encrypting and decrypting data, or signing and verifying signatures. On the other hand, there is no scheme to compute a double-size Montgomery multiplication efficiently with such coprocessors.

4.1 Instruction for Remainders

First, we define the instruction for computing the remainder of Montgomery multiplications in Definition [1].

Definition 1. *For numbers,* $0 \le x, y < \min\{w, 2^n\}$*,* $2^{n-1} < w < 2^{n+1}$*,* $m = 2^n$*,* *gcd*$(m, w) = 1$*, the* MultMon *instruction is defined as*
$r = \mathsf{MultMon}(x, y, w)$ *with* $r := xym^{-1} \pmod{w}$.

4.2 Instruction for Quotients

As the schemes of Fischer et al. [2] and Chevallier-Mames et al. [1] require quotients of n-bit classic modular multiplications, our scheme also requires quotients of n-bit Montgomery multiplications to construct $2n$-bit remainder. However, there is no definition of quotients of Montgomery multiplications. To solve this problem, we extend the notion of quotients to the case of Montgomery multiplications.

The remainder calculated by Montgomery multiplications is different from the remainder calculated by classical modular multiplications. Indeed, from Definition [1], the following equation holds; $xy \equiv rm \pmod{w}$, where $m = 2^n$. The above equation means that there is some integer q satisfying: $xy - qw = rm$. We call this integer q *the quotient of the Montgomery multiplication.*

Now, we define the instruction to calculate the quotient and the remainder of the Montgomery multiplication in Definition [2].

Definition 2. *For numbers,* $0 \le x, y < \min\{w, 2^n\}$, $2^{n-1} < w < 2^{n+1}$, $m = 2^n$, *gcd*(m, w)=1*, the* MultMonDiv *instruction is defined as* $(q, r) = \mathsf{MultMonDiv}(x, y, w)$ *with* $r := xym^{-1} \pmod{w}$ *and* q *satisfies the equation;* $xy = qw + rm$.

In Section [5], we will show an algorithm to implement the MultMonDiv instruction and establish its correctness.

Algorithm 3. Modified Fischer et al.'s Algorithm

INPUT: $A = a_1 z + a_0 m, B = b_1 z + b_0 m, N = n_1 z + n_0 m,$ with $M = 2^{2n}$;
OUTPUT: $ABM^{-1} \pmod{N}$;

```
1. $(q_1, r_1) = \mathsf{MultMonDiv}(b_1, z, n_1)$
2. $(q_2, r_2) = \mathsf{MultMonDiv}(q_1, n_0, z)$
3. $(q_3, r_3) = \mathsf{MultMonDiv}(a_1, r_1 - q_2 + b_0, n_1)$
4. $(q_4, r_4) = \mathsf{MultMonDiv}(a_0, b_1, n_1)$
5. $(q_5, r_5) = \mathsf{MultMonDiv}(q_3 + q_4, n_0, z)$
6. $(q_6, r_6) = \mathsf{MultMonDiv}(a_1, r_2, z)$
7. $(q_7, r_7) = \mathsf{MultMonDiv}(a_0, b_0, z)$
8. **Return** $(r_3 + r_4 - q_5 - q_6 + q_7)z + (r_7 - r_6 - r_5)m$
```

4.3 Representation of $2n$-Bit Integers

We define a new representation to divide a $2n$-bit integer into two n-bit integers for n-bit Montgomery multipliers.

Definition 3. *For numbers,* $0 \leq A, B < N$, $2^{2n-1} < N < 2^{2n}$, $2^{n-1} < z < 2^n$, *z is odd,* $m = 2^n$ *and* $gcd(m, z) = 1$, *the representation is defined as* $N = n_1 z + n_0 m$, $A = a_1 z + a_0 m$, $B = b_1 z + b_0 m$.

The product $n_0 m$ is always even. Therefore z and n_1 must be odd, whenever N is odd.

4.4 Modified Fischer et al.'s Algorithm

Thanks to Definition 3, we extend schemes of Fischer et al. to the case of Montgomery multiplication in Algorithm 3, which only uses odd moduli n_1 and z. Since n-bit Montgomery multiplications output the n-bit remainder: $xym^{-1} \pmod{w}$ where $0 \leq x, y < w$, $2^{n-1} < w < 2^n$ and $m = 2^n$, our algorithm outputs the $2n$-bit remainder of the Montgomery multiplication: $ABM^{-1} \pmod{N}$ where $0 \leq A, B < N$, $2^{2n-1} < N < 2^{2n}$ and $M = 2^{2n}$.

Theorem 1. *Algorithm 3 computes* $ABM^{(-1)} \pmod{N}$ *calling length* n-$\mathsf{MultMonDiv}$ *instruction, provided that* $0 \leq A, B < N < 2^{2n}$ *and* $M = 2^{2n}$.

We show the proof of Theorem 1 in Appendix A.

5 Implementations for Quotients

In Section 4, we defined the quotient of Montgomery multiplications; in fact, this quotient is necessary to compute $2n$-bit Montgomery multiplications. We consider two types of settings, and in each case, show efficient algorithms to calculate the quotients. In the first settings, we assume that we have to re-use an existing n-bit Montgomery multiplier in software, and cannot modify it. Thus,

Algorithm 4. MultMonDiv instruction calling the MultMon instruction

```
INPUT: $x, y, w$ with $0 \le x, y < w$, $2^{n-1} < w < 2^n$, $m = 2^n$ and $\gcd(w, m) = 1$;
OUTPUT: $q$, $r$;

1. $r \leftarrow \mathsf{MultMon}(x, y, w)$
2. $r' \leftarrow \mathsf{MultMon}(x, y, w + 2^n)$
3. $tmp \leftarrow xy - rw + r'(w + 2^n) \pmod{2^2}$
4. If $tmp > 2$, then $tmp \leftarrow tmp - 2^2$.
5. $q \leftarrow tmp \times (w + 2^n) + r' - r$
6. **Return** $(q, r)$
```

we assume a pure software implementation of our double-size technique. Section 5.1 shows how to emulate the calculation of the quotient with two calls to the n-bit Montgomery multiplier. In the second settings, modifications of the hardware Montgomery multiplier are allowed, but still restricted to n-bit operands. Section 5.2 explains how to modify the circuitry with minimal changes.

5.1 Software Approach: Calling Montgomery Multipliers

In this subsection, we introduce Algorithm 4, which emulates the calculation of quotients with two calls to the n-bit Montgomery multiplier.

Theorem 2. *Algorithm 4 computes* MultMonDiv(x, y, w) *instruction calling length* $(n + 1)$-MultMon *instruction twice, provided that* $0 \le x, y < w$, $2^{n-1} < w < 2^n$, $m = 2^n$ *and* $\gcd(m, w) = 1$.

Appendix B shows the proof of Theorem 2.

5.2 Hardware Approach: Changing Montgomery Multipliers

This subsection shows that the implementation of the MultMon instruction can be changed in order to directly compute the MultMonDiv instruction. In fact, since the MultMon instruction already has information of the quotient, Algorithm 5 has little changes compared to the MultMon instruction, which is the standard technique proposed by Montgomery [6]. We just insert Step 2.(c) to calculate the quotient and output the quotient along with the remainder in Step 4.

6 Experimental Results

6.1 Validation

We implemented 2048-bit Montgomery multiplications and exponentiations on an emulator for smartcards using our proposed technique. Its coprocessor can only handle 1024-bit operands for Montgomery multiplications. Therefore, unlike the assumption in Theorem 2, we make a more strict assumption for getting the quotients: the bit-length of the modulus is exactly twice as much as the operands

Algorithm 5. MultMonDiv instruction with modified MultMon instruction

INPUT: x, y, w with $0 \leq x, y < w, m = 2^n$, $\gcd(w, m) = 1$ and $w' = -w^{(-1)} \mod 2$;
OUTPUT: q, r;

```
1. $q \leftarrow 0$ and $r \leftarrow 0$
2. For $i$ from 0 to $(n-1)$ do the following:
   (a) $u_i \leftarrow (r_0 + x_i y_0)w' \mod 2$
   (b) $q \leftarrow q + u_i 2^i$
   (c) $r \leftarrow (r + 2r_i + u_i w)/2$
3. If $r \geq w$ then $r \leftarrow r - w$
4. Return $(q, r)$
```

size of the coprocessor. We show another algorithm in Appendix C for implementing the MultMonDiv instruction with this more strict condition which we faced.

6.2 Practical Implementation Issues

Representations of $2n$-bit Integers
We implemented our technique with w set as $(2^n - 1)$ for two reasons. One reason is that w should be odd because of the requirements of Montgomery multiplications, and that w is $2^{n-1} < w < 2^n$ because of the assumptions in Theorem 2. The other is that the conversion to such representation is easy. We show how to get the representation of $2n$-bit moduli for Montgomery multiplications. $N = n'_1 2^n + n'_0 = n_1(2^n - 1) + n_0 2^n$. Then, we can calculate n_1 and n_0 easily for Montgomery multiplications. $n_1 := 2^n - n'_0$ and $n_0 := n'_1 - n_1 + 1$.

Condition on the Modulus
Unfortunately, it is not easy to achieve the assumption of Theorem 2, namely that the modulus must be greater than 2^{n-1}. For example, if w is $(2^n - 1)$, $0 < n_1 < 2^n$; n_1 can be smaller than 2^{n-1}. One choice is to modify Algorithm 3 slightly. When the value of n_1 is $0 < n_1 < 2^{n-1}$, $(m - n_1)$ can be applied to Algorithm 3 as modulus instead of n_1 to satisfy the assumption. We can compute the quotient and the remainder of the modulus n_1 from the modulus $(m - n_1)$ with the following equations. $xy = q(m - w) + rm = (-q)w + (q + r)m$.

Handling of Input Value
The input value in Algorithm 3 may break the assumption $(0 < x, y < 2^n)$ of Theorem 2. Since the Montgomery remainder only is affected by this problem, it could be solved by the following fact: if $xy \cdot m^{(-1)} \pmod{w} = r$, then $(x + im)(y + jm) \cdot m^{(-1)} \pmod{w} = r + jx + iy + ijm$ holds, where i and j are small integers.

Reduction of the intermediate output
It often happens that the value of the intermediate output q and r are not reduced. In this case, we have a strategy to compute new integers q' and r'

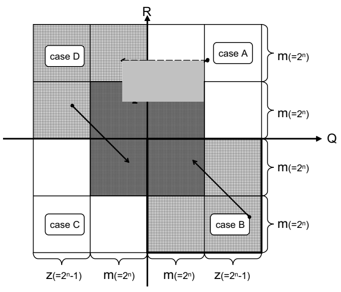


Fig. 2. Reduction for a quotient and a remainder

satisfying the following equation with two vectors; $(z, -m)$ and $(n_0 - z, n_1 + m)$. $(q', r') = (q, r) + i(z, -m) + j(n_0 - z, n_1 + m)$, where i and j are small integers. If one set z as $2^n - 1$ and m as 2^n, the vector $(z, -m)$ is independent of the other $(n_0 - z, n_1 + m)$: The direction of the vector is that $-m/z \approx -1$ and $0 \leq (n_1 + m)/(n_0 - z) \leq 3/4$.

Figure 2 shows works of our strategy to reduce q and r to $|q| < m$ and $|r| < m$ respectively, where $m \leq |q| \leq m + z$, $m \leq |r| \leq m + z$, $m = 2^n$ and $z = 2^n - 1$.

There are four cases to reduce q and r.

Case A: $0 \leq q, r$ and, $m \leq q \leq z + m$ or $m \leq r \leq z + m$.
Compute $(q, r) \leftarrow (q - (n_0 - z), r - (n_1 + m))$, then return (q, r) or go to case A or D.

Case B: $0 \leq q, r$ and, $m \leq q \leq z + m$ or $-(z + m) \leq r \leq -m$.
Compute $(q, r) \leftarrow (q - m, r + z)$, then return (q, r).

Case C: $q, r < 0$ and, $-(z + m) \leq q \leq -m$ or $-(z + m) \leq r \leq -m$.
Compute $(q, r) \leftarrow (q + (n_0 - z), r + (n_1 + m))$, then return (q, r) or go to case B or C.

Case D: $0 \leq q, r$ and, $-(z + m) \leq q \leq -m$ or $m \leq r \leq z + m$.
Compute $(q, r) \leftarrow (q + m, r - z)$, then return (q, r).

In fact, q and r are always in fixed range; $-3 \cdot 2^n < q < 5 \cdot 2^n$ and $-2 \cdot 2^n < r < 2^n$, because of the equation in Algorithm 3; $q = r_3 + r_4 - q_5 - q_6 + q_7$ and $r = -r_5 - r_6 + r_7$ with $-2^n < q_i < 2^n$ and $0 \leq r_i < 2^n$. Therefore, it might happen that q and r are outside the area defined by case A to D. However, we can easily extend our technique to such cases.

7 Conclusion

We proposed a novel technique for $2n$-bit Montgomery multiplications, provided that n-bit Montgomery multiplications are available. We defined the quotient of Montgomery multiplications for $2n$-bit Montgomery multiplications. Since Montgomery multiplications have already been implemented on many platforms, we proposed one technique to emulate the calculation of Montgomery quotients

Table 1. Calculation of a quotient of Montgomery Multiplications

Calls (average)	case 1 : do not change MultMon instruction		case 2 : change MultMon instruction	
modulus	$0 < w < 2^{n-1}$	$2^{n-1} < w < 2^n$	$0 < w < 2^{n-1}$	$2^{n-1} < w < 2^n$
Addition/Subtraction	7.5	5.5	0	2
MultMon instruction	2	2	0	0
MultMonDiv instruction	0	0	1	1

Table 2. Average calls of a $2n$-bit Montgomery Multiplications and an n-bit one

Bitlength	n	$2n$	
Calls(average)	–	case 1	case 2
Addition/Subtraction	0	50.5	12
MultMoninstruction	1	14	0
MultMonDivinstruction	0	0	7

with the available Montgomery multiplications unit. In addition, we proposed another approach where the implementation of the Montgomery multiplications unit is changed in order to directly calculate the quotient. The approach of calling available units takes two instructions, and the approach changing the units has roughly the same cost as one instruction.

As a result, our proposed techniques calculate $2n$-bit Montgomery multiplications by calling the crypto-coprocessor implementing n-bit Montgomery multiplications only, or the instruction for computing remainders and quotients of n-bit Montgomery multiplications.

This paper concentrates on the way to compute double-size Montgomery multiplications with a single-size crypto-coprocessor. Therefore, although the scheme of Chevallier-Mames et al. requires less calls to the multiplier than Fischer et al.'s one, we extended the scheme of Fischer et al., which allows to make our scheme simple: our proposed representation of $2n$-bit integers can avoid using even moduli for Montgomery multipliers. A further direction of this research is to optimize computational costs of $2n$-bit Montgomery multiplications, for example by using Chevallier-Mames et al. technique.

References

1. Chevallier-Mames, B., Joye, M., Paillier, P.: Faster Double-Size Modular Multiplication From Euclidean Multipliers. In: Walter, C.D., Koç, Ç.K., Paar, C. (eds.) CHES 2003. LNCS, vol. 2779, pp. 214–227. Springer, Heidelberg (2003)
2. Fischer, W., Seifert, J.P.: Increasing the bitlength of crypto-coprocessors. In: Kaliski Jr., B.S., Koç, Ç.K., Paar, C. (eds.) CHES 2002. LNCS, vol. 2523, pp. 71–81. Springer, Heidelberg (2003)
3. Handschuh, H., Paillier, P.: Smart card crypto-coprocessors for public-key cryptography. In: Schneier, B., Quisquater, J.-J. (eds.) CARDIS 1998. LNCS, vol. 1820, pp. 372–379. Springer, Heidelberg (2000)

4. Koc, C.: Montgomery reduction with even modulus. IEE Proceedings Computer and Digital Techniques 141(5), 314–316 (1994)
5. Menezes, A.J., van Oorschot, P.C., Vanstone, S.A.: Handbook of Applied Cryptography. CRC Press, Boca Raton (1996)
6. Montgomery, P.L.: Modular multiplication without trial division. Mathematics of Computation 44(170), 519–521 (1985)
7. Naccache, D., M'Raïhi, D.: Arithmetic co-processors for public-key cryptography: The state of the art. In: CARDIS, pp. 18–20 (1996)
8. Paillier, P.: Low-cost double-size modular exponentiation or how to stretch your cryptoprocessor. In: Imai, H., Zheng, Y. (eds.) PKC 1999. LNCS, vol. 1560, pp. 223–234. Springer, Heidelberg (1999)
9. Quisquater, J.J., Couvreur, C.: Fast decipherment algorithm for rsa public-key cryptosystem. Electronics Letters 18(21), 905–907 (1982)
10. Rivest, R.L., Shamir, A., Adelman, L.M.: A method for obtaining digital signatures and public-key cryptosystems. Communications of the ACM 21(2), 120–126 (1978)
11. RSA Laboratories: RSA challenges, `http://www.rsasecurity.com/rsalabs/`

A Modified Fischer et al.'s Algorithm

We will prove theorem 1 in Section 2.1 to be correct, which an output of Algorithm 1 is indeed congruent to $ABM^{(-1)}$ modulo N, where $0 \leq A, B < N$, $2^{2n-1} < N < 2^{2n}$ and $M = 2^{2n}$.

Proof. Firstly, $2n$-bit integers A, B, N are decomposed on the following equation;

$$A = a_1 z + a_0 m,\ B = b_1 z + b_0 m,\ N = n_1 z + n_0 m$$

where z is odd, $2^{n-1} \leq z < 2^n$ and $m = 2^n$. Then, we continue to be the following.

$$\begin{aligned}
AB &= (a_1 z + a_0 m)(b_1 z + b_0 m) \\
&= a_1 b_1 zz + a_1 b_0 zm + a_0 b_1 zm + a_0 b_0 mm \\
&= a_1(q_1 n_1 + r_1 m)z + a_1 b_0 zm + a_0 b_1 zm + a_0 b_0 mm \\
&= a_1 r_1 zm - a_1 q_1 n_0 m + a_1 b_0 zm + a_0 b_1 zm + a_0 b_0 mm \\
&= a_1 r_1 zm - a_1(q_2 z + r_2 m)m + a_1 b_0 zm + a_0 b_1 zm + a_0 b_0 mm \\
&= a_1(r_1 - q_2 + b_1)zm - a_1 r_2 mm + a_0 b_1 zm + a_0 b_0 mm \\
&= (q_3 n_1 + r_3 m)zm - a_1 r_2 mm + a_0 b_1 zm + a_0 b_0 mm \\
&= (q_3 n_1 + r_3 m)zm - a_1 r_2 mm + (q_4 n_1 z + r_4 m)zm + a_0 b_0 mm \\
&= (r_3 + r_4)zmm - a_1 r_2 mm - (q_3 + q_4)mm + a_0 b_0 mm \\
&= (r_3 + r_4)zmm - a_1 r_2 mm - (q_5 z + r_5 m)mm + a_0 b_0 mm \\
&= (r_3 + r_4)zmm - (q_6 z + r_6 m)mm - (q_5 z + r_5 m)mm + a_0 b_0 mm \\
&= (r_3 + r_4)zmm - (q_6 z + r_6 m)mm - (q_5 z + r_5 m)mm + (q_7 z + r_7 m)mm \\
&= (r_3 + r_4 - q_5 - q_6 + q_7)zmm - (r_5 + r_6 - r_7)mmm \\
&= \{(r_3 + r_4 - q_5 - q_6 + q_7)z - (r_5 + r_6 - r_7)m\}m^2 \qquad \square
\end{aligned}$$

B Proof of Theorem 2

To show the proof of Theorem 2 telling correctness of Algorithm 4, we prove the following Lemmas 1, 2 and 3, step by step. Lemma 1 states the ranges of the quotient.

Lemma 1. *If* $0 \leq x, y < \min\{w, 2^n\}$, $2^{n-1} < w < 2^{n+1}$, $m = 2^n$, *r is provided by* $\mathsf{MultMon}(x, y, w)$ *and q satisfies the equation;* $xy = qw + rm$, *then* $-2^n < q < 2^{n+1}$ *holds.*

Proof. If w is in the range of $2^{n-1} < w < 2^{n+1}$, then we have $qw = xy - rm$ and $-w2^n < xy - rm < 2^{2n}$. Since the minimum value of w is greater than 2^{n-1}, Lemma 1 holds. □

Lemma 2 states ranges of difference between two different quotients of Montgomery multiplications. Whenever Lemma 1 holds, Lemma 2 also holds.

Lemma 2. *If* $0 \leq x, y < w$, $2^{n-1} < w < 2^n$, $m = 2^n$, $gcd(w,\ m) = 1$, *r is provided by* $\mathsf{MultMon}(x,y,w)$, *r' is provided by* $\mathsf{MultMon}(x,y,w + 2^n)$, *$q$ satisfies the equation;* $xy = qw + rm$ *and q' satisfies the equation;* $xy = q'(w + 2^n) + r'm$, *then when we set δ as* $\{xy+rw-r'(w+2^n) \pmod{2^2}\}$, *either* $q'-q = \delta 2^n$ *or* $q' - q = (\delta - 2^2)2^n$ *holds.*

Proof. From Lemma 1, $-2^{n+1} \leq q' - q \leq 2^{n+1}$. Moreover, noticing that $(w + 2^n)^{-1} - w^{-1} = 2^n \pmod{2^{n+2}}$, we get:

$$\begin{aligned} q' - q &= xy((w + 2^n)^{-1} - w^{-1}) - r'm(w + 2^n)^{-1} + rmw^{-1} \\ &\equiv xy2^n - r'(w + 2^n)^{-1}m + rw^{-1}m \pmod{2^{n+2}} \\ &= \{xy - r'(w + 2^n)^{-1} + rw^{-1}\} \times 2^n \end{aligned}$$

Furthermore, $(w + 2^n)^{-1} = w^{-1} = w \pmod{2^2}$, so we have:

$$q' - q \equiv \{xy - r'(w + 2^n) + rw \pmod{2^2}\} \times 2^n \pmod{2^{n+2}}$$

If $(xy - r'(w + 2^n) + rw) < 2 \pmod{2^2}$, then

$$q' - q = \{xy - rw + r'(w + 2^n) \pmod{2^2}\} \times 2^n.$$

Otherwise, we have:

$$q'-q = \{\{xy+rw-r'(w+2^n) \pmod{2^2}\}-2^2\}\times 2^n. \qquad \square$$

Finally, based on Lemma 2, Lemma 1 shows the equation for getting the quotient of Montgomery multiplications.

Lemma 3. *If* $0 \leq x, y < w$, $2^{n-1} < w < 2^n$, $m = 2^n$, $gcd(w,\ m) = 1$, *r is provided by* $\mathsf{MultMon}(x,y,w)$, *r' is provided by* $\mathsf{MultMon}(x,y,w + 2^n)$, *$q$ satisfies the equation;* $xy = qw + rm$ *and q' satisfies the equation;* $xy = q'(w + 2^n) + r'm$, *then when we set δ as* $\{xy + rw - r'(w + 2^n) \pmod{2^2}\}$, *either* $q = \delta(w + 2^n) + r' - r$ *or* $q = (\delta - 2^2)(w + 2^n) + r' - r$ *holds.*

Proof. By hypothesis, $qw + rm = q'(w + 2^n) + r'm$. Therefore $q2^n = (q - q')(w + 2^n) + (r - r')m$ holds. From lemma 2, we have:

$$\text{either } q2^n = \delta 2^n(w + 2^n) + (r' - r)m, \text{ or}$$
$$q2^n = (\delta - 2^2)2^n(w + 2^n) + (r' - r)m. \qquad \square$$

The proof of Theorem 2 follows from Lemma 3.

C Approach for Quotients of Montgomery Multiplications Based on Limited Memories of a Coprocessor

Lemma 2 and Lemma 3 assume $r' = \mathsf{MultMon}(x, y, w + 2^n)$ and q' that satisfy $xy = q'(w + 2^n) + r'm$. But it is possible that the MultMon instruction cannot treat the modulus $(w + 2^n)$ when it is implemented in hardware with just n-bit memory, or when there are restrictions for the size of the modulus. For this case, we use $(w \pm 2^{n-2})$ rather than $(w + 2^n)$. In this case, one can prove lemmas and theorems similar to that from Section 5. We show algorithms to calculate the quotient of Montgomery multiplications instead of Algorithm 4,

Algorithm 6. Implementation of MultMonDiv Limited version1

INPUT: x, y, w with $0 \le x, y < w$, $2^{n-1} + 2^{n-2} < w < 2^n$ and $m = 2^n$ with $\gcd(w, m) = 1$;
OUTPUT: q,r;

```
1. $r \leftarrow \mathsf{MultMon}(x, y, w)$
2. $r' \leftarrow \mathsf{MultMon}(x, y, w - 2^{n-2})$
3. $tmp \leftarrow xy + rw - r'(w - 2^{n-2}) \pmod{2^4}$
4. if tmp $\ge 2^3$ then $tmp \leftarrow tmp - 2^4$.
5. $q \leftarrow tmp \times (w - 2^{n-2}) + 4(r' - r)$
6. **Return** $(q, r)$
```

Algorithm 7. Implementation of MultMonDiv Limited version2

INPUT: x, y, w with $0 \le x, y < w$, $2^{n-1} < w < 2^{n-1} + 2^{n-2}$ and $m = 2^n$ with $\gcd(w, m) = 1$;
OUTPUT: q,r;

```
1. $r \leftarrow \mathsf{MultMon}(x, y, w)$
2. $r' \leftarrow \mathsf{MultMon}(x, y, w + 2^{n-2})$
3. $tmp \leftarrow xy - rw + r'(w + 2^{n-2}) \pmod{2^4}$
4. if tmp $\ge 2^3$ then $tmp \leftarrow tmp - 2^4$.
5. $q \leftarrow tmp \times (w + 2^{n-2}) + 4(r - r')$
6. **Return** $(q, r)$
```

only when coprocessors have limited memories, whose bit-lengths are just n bits. Our proposed algorithms are divided into Algorithm 6 and Algorithm 7, because we treat $(2^{n-1} < w < 2^{n-1} + 2^{n-2})$ and $(2^{n-1} + 2^{n-2} < w < 2^n)$ separately.

Firstly, we show Algorithm 6, where w is $2^{n-1} < w < 2^{n-1} + 2^{n-2}$.

Secondly, we show Algorithm 7, where w is $2^{n-1} + 2^{n-2} < w < 2^n$.

Delaying and Merging Operations in Scalar Multiplication: Applications to Curve-Based Cryptosystems

Roberto Maria Avanzi[1,⋆]

Faculty of Mathematics and Horst Görtz Institute for IT Security
Ruhr-University Bochum, Germany
roberto.avanzi@ruhr-uni-bochum.de

Abstract. In this paper we introduce scalar multiplication algorithms for several classes of elliptic and hyperelliptic curves. The methods are variations on Yao's scalar multiplication algorithm where independent group operations are shown in an explicit way. We can thus merge several group operations and reduce the number of field operations by means of Montgomery's trick. The results are that scalar multiplication on elliptic curves in even characteristic based on point halving can be improved by at least 10% and the performance of Koblitz curves by 25% to 32%.

1 Introduction

This paper describes efficient methods for scalar multiplication on elliptic curves (EC) and hyperelliptic curves (HEC). We provide replacements for techniques such as: double–and–add, halve–and–add [18,25], and methods based on the Frobenius operation [19,26,27].

Our scenario is the following: the base point is *variable*, and memory usage is *not* an issue (such as on personal computers). We are *not* concerned with minimizing memory usage: we consider the cases where we do not significantly change it, and where we can use as much memory as we want. The techniques developed here are applicable to a wide range of groups, that satisfy the following assumptions:

1. The group $(G, +, 0)$ is an algebraic variety, with explicit formulae for an addition, which uses field inversion, and for a second operation which is a group automorphism.
2. The speed of field inversion relative to field multiplication is in a quite common range for software implementations, i.e. from 6 to 30.
3. The group automorphism ϕ satisfies the following properties:
   (a) ϕ is identified with an element (also denoted by ϕ) of a number ring R, identified with a subring of the automorphism ring of G.

⋆ Supported by the European Commission through the IST Programme under Contract IST-2002-507932 ECRYPT.

© Springer-Verlag Berlin Heidelberg 2007

(b) We can represent each rational integer n to the the "base of ϕ" using a suitable digit set $\mathcal{D} \subset \mathbb{Z}[\phi] \subset R$ in the form

$$n \equiv \sum_{i=0}^{\ell} n_i \phi^i \bmod \Im \ , \tag{1}$$

where $n_i \in \mathcal{D}$ and $\Im$ is a (possibly trivial) ideal of R of elements acting like the identity on the group. Relation (1) implies that

$$n \cdot P = \sum_{i=0}^{\ell} n_i \phi^i(P) \ . \tag{2}$$

Important examples of group–automorphism pairs (G, ϕ) are

1. ϕ is the doubling of an element of G, i.e. $\phi(P) = 2 \cdot P$ for all $P \in G$. We identify ϕ with the numer 2. Typical examples are EC and HEC [16].
2. Halving in a subgroup G of an EC in even characteristic, i.e. for a given $P \in G$ computing $Q \in G$ such that $2Q = P$. The group G must have odd order N and ϕ is identified with $1/2$ modulo N.
3. The Frobenius operation on a Koblitz curve [19] E over a degree m extension of $\mathbb{F}_2$. The map ϕ is the Frobenius endomorphism (usually denoted by τ), ϕ^m operates trivially on the points of E, and $\Im$ is the ideal generated by $(\phi^m - 1)/(\phi - 1)$. Here ϕ is identified with $\tau := \frac{\mu + \sqrt{-7}}{2}$ where $\mu = \pm 1$ depends on the curve.
4. ϕ is the multiplication by a small constant p, where G is the rational point group of an EC over $\mathbb{F}_{p^m}$. Such curves have been proposed for pairing based systems [6,13,15].

Our starting point is the observation that Yao's scalar multiplication algorithm [28] contains group operations that are independent from each other. If the implementation of these operations makes use of field inversions, these can be merged using a trick by Montgomery. This trick has been already used to merge group operations in the context of scalar multiplication. These improvements have been restricted to: simultaneous scalar products; improving the precomputation stage [12]; devising formulae to compute expressions of the form $3P$, $4P$, $2P + Q$ etc. on elliptic curves using only one inversion [7] and thus reduce the number of inversions required in a slightly modified Horner scheme; and when the base point is fixed [22]. *The present paper is the first application of Montgomery's trick to the main loop of Yao's algorithm for a single scalar multiplication with a variable base point.*

Our techniques can improve the performance of scalar multiplication on EC using point halving, as well as on Koblitz curves. The improvements in these cases can often reach and exceed 10% and 30% respectively. On the other hand, the methods do not seem to bring savings to EC and HEC scalar multiplication based on point doubling.

The paper is organised as follows. In § 2 we describe the classical Φ-and-add and Yao's scalar multiplication methods. Our techniques are introduced in § 3, and analysed in §4. §5 is devoted to further refinements and practical usage of our methods, including operation counts in some settings of cryptographic relevance. Finally, in §6 we conclude.

2 Scalar Multiplication Techniques

As in the introduction let $(G, +, 0)$ be a group, with a composition operation $+$ called addition and neutral element 0, endowed with a group automorphism ϕ enjoying the property that scalars can be written "to the base of ϕ" as in (1). Examples of such groups have already been mentioned. Some examples of recodings for the scalar n are: the binary and radix-r representation of integers; the Non Adjacent Form (NAF) [24]; the GNAF [8]; sliding window expansions [11,2] and their τ-adic counterparts [26,27]; and fractional windows [23].

2.1 Φ-and-Add Scalar Multiplication

Algorithm 1 describes several scalar multiplication techniques commonly known also as Horner schemes.

Let (G, ϕ) satify the requirements set in the introduction. Before proceeding we need some notation.

Notation 1. *The digit set* $\mathcal{D}$ *can be of the form* $\mathcal{D} = \{0\} \cup \mathcal{D}^+$ *with* $0 \notin \mathcal{D}^+$ *or* $\mathcal{D} = \{0\} \cup \mathcal{D}^+ \cup (-\mathcal{D}^+)$ *with* $\mathcal{D}^+ \cap (-\mathcal{D}^+) = \emptyset$*.*

The set $\mathcal{D}^+$ *is called set of* positive *digits, the digits in* $-\mathcal{D}^+$ *are the* negative *digits. For* $d \in \mathcal{D}$ *define the* sign *of a digit as follows:* $\mathrm{Sign}(d) = 0$ *if* $d = 0$*,* $\mathrm{Sign}(d) = 1$ *if* $d \in \mathcal{D}^+$*, and* $\mathrm{Sign}(d) = -1$ *if* $d \in -\mathcal{D}^+$*. If* $d \in \mathcal{D}^+$ *or* $d = 0$ *then put* $|d| = d$*, otherwise (if* $d \in -\mathcal{D}^+$*) put* $|d| = -d$*. This is the* absolute value *of a digit.*

Note that we consider two cases according to the fact whether computing $-P$ from P is a computationally expensive or negligible operation. The situation where $\mathcal{D} = \{0\} \cup \mathcal{D}^+ \cup (-\mathcal{D}^+)$ occurs when in the group G the computation of the inverse takes negligible time. In what follows we shall only consider this situation as it is the most common one in our context. The other case is just a simple adaptation.

In Steps 5 and 6, $n_i \cdot P$ is taken from the table computed in Step 1.

2.2 Yao's Method

Algorithm 2 is a straigthforward modification of Yao's [28] algorithm obtained upon replacing doublings with ϕ and distinguishing between "positive" and "negative" digits.

Observe that we are adding to registers corresponding to the various digits, instead of adding from a table indexed by them, and at the end we have to recombine the result.

Algorithm 2 is less used than Algorithm 1 as it is considered slower: First of all, a careful observation of the steps required, including replacing the first addition to registers initially set to zero with assignments, shows that (for example in

Algorithm 1. ϕ-and-add method for scalar multiplication

INPUT: An element P of a group G endowed with an addition $+$ and an automorphism ϕ, and integer n recoded as $n \approx \sum_{i=0}^{\ell} n_i \phi^i$ according to the notation of (1), and the n_i from a digit set $\mathcal{D}$ as per Notation 1.
OUTPUT: The element $n \cdot P \in G$.

```
1. Precompute $d \cdot P$ for all the digits $d \in \mathcal{D}^+$ and store them in a table
2. $Q \leftarrow 0 \in G$
3. **for** $i = \ell$ **downto** $0$ **do**
4.     $Q \leftarrow \phi(Q)$
5.     **if** $n_i \in \mathcal{D}^+$ **then** $Q \leftarrow Q + n_i \cdot P$
6.     **if** $n_i \in -\mathcal{D}^+$ **then** $Q \leftarrow Q - (-n_i) \cdot P$
7. **return** $y$
```

the case $\Phi = 2$) Algorithms 1 and 2 take the same number of group operations. Furthermore, the choice of the representation for the group elements plays a big role in determining the performance: For EC [12] and genus 2 HEC [20] one can represent the elements using different coordinate systems, and there are formulae that add elements given in different coordinate systems with output in another system: In Algorithm 1 we choose a coordinate system with fast and inversion-free doubling for Q, and keep the elements $d \cdot P$ in affine coordinates to add them to Q faster. A similar strategy has not been investigated for Algorithm 1 yet. Hence, Yao's method can even be *less* efficient than the Horner scheme.

We call the computation of $\sum_{d\in\mathcal{D}^+} dX_d$ the *final accumulation step*. If $\mathcal{D}^+ = \{1, 3, 5, \ldots, t\}$ with t odd, then we can compute $\sum_{d\in\mathcal{D}^+} dX_d$ as two time the sum of the terms X_t, $X_t + X_{t-2}$, $X_t + X_{t-2} + X_{t-4}$, ..., $X_t + X_{t-2} + \cdots + X_3$, to which we add $X_t + X_{t-2} + \cdots + X_3 + X_1$. Each of the partial summands is computed by a successive group addition. The total number of operations is $t-1$ group additions and one doubling. If $t = 2^{w-1} - 1$ this amounts to $2^{w-1} - 2$ group additions and one doubling.

Algorithm 2. Yao's scalar multiplication algorithm

INPUT: An element P of a group G and integer n recoded as $n \approx \sum_{i=0}^{\ell} n_i \phi^i$ according to the notation of (1), with $n_i \in \mathcal{D}$ as per Notation 1.
OUTPUT: The element $n \cdot P \in G$.

```
1. Create a table of *accumulator registers* $X_d \leftarrow 0 \in G$ for all $d \in \mathcal{D}^+$
2. $Q \leftarrow P$
3. **for** $i = 0$ **to** $\ell$ **do**
4.     **if** $n_i \in \mathcal{D}^+$ **then** $X_{n_i} \leftarrow X_{n_i} + Q$
5.     **if** $n_i \in -\mathcal{D}^+$ **then** $X_{-n_i} \leftarrow X_{-n_i} - Q$
6.     $Q \leftarrow \phi(Q)$                                   [**if** $(i \neq \ell)$]
7. **return** $\sum_{d\in\mathcal{D}^+} d \cdot X_d$
```

3 Delaying Group Additions to Collect Field Inversions

3.1 First Approach: Conservative Memory Usage

In Steps 4 and 5 of Yao's algorithm at successive iterations of the main loop, the group additions are in fact independent from each other until a digit (or its opposite) is encountered again. In fact, the value of Q depends only on the previous value of Q and thus only on P, and not on the other computations of the algorithms; hence, as long as we add independently computed points to different registers (i.e. these additions are associated to digits with absolute values different from each other), these additions can be performed in any order. We want to perform these group additions simultaneously using Montgomery's trick of simultaneous inversions. A realization of this idea is given as Algorithm 3.

Instead of performing the addition right away as in Algorithm 2, we first store information which is sufficient to keep track of the operations to be done, i.e. the value of Q to be added and the register that it has to be added to. To do this we keep a table $(A_j, d_j) \in G \times \mathcal{D}^+$, called the *queue*, containing up to q pairs, where A_j is the element (a value taken by the variable Q) to be added to the d_j-th element of the sequence of accumulator registers $\{X_d\}$. We also implement a routine called `FLUSH` that finally performs the delayed, independent, group additions $X_{d_j} \leftarrow X_{d_j} + A_j$ for $j = 1, 2, ..., k$ in parallel, where k is the number of pairs currently in the queue. Such a routine is invoked in the following cases: when a digit equal to a digit already in the queue (or its opposite) is encountered, when the queue itself contains q different elements and cannot accept any further pairs (A_j, d_j), or at the end of the algorithm, before performing the final accumulation step. As a consequence, there is no need for a value of q larger than the size of the digit set, i.e. $q \leq \#\mathcal{D}$.

Note that the first addition to an accumulator register is in fact just an assignment. This is explicitly reflected in the algorithm itself (Step 7).

3.2 Second Approach: Using More Memory

Algorithm 4 first computes all the $\phi^i(P)$ with $n_i \neq 0$, i.e. those that get added to or subtracted from the accumulator registers. Then it tries to collect them in groups as large as possible where each element corresponds to a different digit value. In fact, the absolute values of non zero digit which are *consecutive* in a given integer recoding usually do not form a *complete* set of positive digits. But, if we first compute all the $\phi^i(P)$, we do not need to group the additions in the same order the corresponding digits appear in the recoding. The hope here is to be able to collect groups of as many as $\#\mathcal{D}^+$ additions together. As §§4.2 and 5.3 will show, this will result in a gain over the previous approach. As in Algorithm 3 the first additions to accumulator registers are replaced by assignments.

Remark 1. Counters for the number of digits that still do not have the `used` bit set, as well as $\#\mathcal{D}^+$ many similar counters for the digits equal to each $d \in \mathcal{D}^+$, can be kept to speed up implementation of the tests in Steps 11 and 14. The last values found for the minimal i's in Step 15 can also be kept for each positive

Algorithm 3. Scalar multiplication with delayed group operations

INPUT: An element P of a group G and integer n recoded as $n \approx \sum_{i=0}^{\ell} n_i \phi^i$ according to the notation of (1), with $n_i \in \mathcal{D}$ as per Notation 1, a parameter q.
OUTPUT: The element $n \cdot P \in G$.

```
1.  Create a table of accumulator registers $X_d \leftarrow 0_G$ for all $d \in \mathcal{D}^+$
2.  Create a table (called the queue) $(A_k, d_k)$ for $1 \leq k \leq q$
3.  $Q \leftarrow P$
4.  for $i = 0$ to $\ell$ do
5.      if $n_i \neq 0$ do
6.          if $X_{|n_i|} = 0_G$ then                          [First addition to register]
7.              $X_{|n_i|} \leftarrow \mathrm{Sign}(n_i) Q$
8.          else                        [Otherwise, accumulate additions to be merged]
9.              if $|n_i| = d_j$ for some $j$ with $1 \leq j \leq q$ then FLUSH, $k \leftarrow 0$
10.             if $n_i \in \mathcal{D}^+$ then $k \leftarrow k+1$, $(A_k, d_k) \leftarrow (Q, n_i)$
11.             if $n_i \in -\mathcal{D}^+$ then $k \leftarrow k+1$, $(A_k, d_k) \leftarrow (-Q, -n_i)$
12.             if $k = q$ then FLUSH, $k \leftarrow 0$
13.     $Q \leftarrow \phi(Q)$                                          [if $(i \neq \ell)$]
14. if $k \neq 0$ then FLUSH
15. return $\sum_{d \in \mathcal{D}^+} d \cdot X_d$
```

digit, to avoid scanning the whole recoding each time. These counters permit to keep to total time spent scanning the recoding and building the queues linear in the recoding length, and negligible with respect to the group operations.

4 Performance Analysis

4.1 Analysis of the Conservative Memory Method (Algorithm 3)

Our analysis is based on the analysis of the birthday paradox.

Let $\mathcal{E}s$ be the expected number of elements in the queue of length q before we find a colliding digit, i.e. a digit equal, up to sign, to one already met after the last time the queue was flushed.

We adopt a urn-and-balls model: In an urn there are r balls, each inscribed with one of the positive digits. We repeatedly draw one ball at random from the urn, remember the result of the draw and replace the ball in the urn. As soon as when we draw a ball which we have already drawn, or after we have drawn q different balls, we stop, and record the number s of draws. We want to find the expected value $\mathcal{E}s$ of s.

We assume that in our representations all nonzero digits occur with equal probability. This is in fact the case in sliding or fixed windows methods (non fractional), both binary and τ-adic, or generic radix p representations.

The expected value $\mathcal{E}s$ is equal to $\sum_{j=0}^{q} p_j$ where p_j is the probability there is no match between balls after j draws. It is also clear that $p_1 = 1$, $p_2 = \left(1 - \frac{1}{r}\right)$

Algorithm 4. Scalar multiplication with delayed group operations, second method

INPUT: An element P of a group G and integer n recoded as $n \approx \sum_{i=0}^{\ell} n_i \phi^i$ according to the notation of (1), with $n_i \in \mathcal{D}$ as per Notation 1, a parameter q.
OUTPUT: The element $n \cdot P \in G$.

```
1.  Create a table of accumulator registers $X_d \leftarrow 0 \in G$ for all $d \in \mathcal{D}^+$
2.  Create a table (called the queue) $(A_k, d_k)$ for $1 \le k \le \#\mathcal{D}^+$
3.  Create a usage boolean table used[$i$] for $0 \le i \le \ell$
4.  $Q \leftarrow P$
5.  for $i = 0$ to $\ell$ do
6.      if $n_i \neq 0$ then
7.          store $Q^{(i)} := Q\ (= \phi^i(P)\,)$, used[$i$] $\leftarrow$ false
8.      else
9.          used[$i$] $\leftarrow$ true
10.     $Q \leftarrow \phi(Q)$
11. while $(\exists\, i :$ used[$i$] $=$ false$)$ do
12.     $k \leftarrow 0$
13.     for all $d \in \mathcal{D}^+$ do
14.         if $(\exists\, i : ($used[$i$] $=$ false and $|n_i| = d))$ then
15.             Let $i$ be minimal among those with used[$i$] $=$ false and $|n_i| = d$
16.             if $X_{|n_i|} = 0_G$ then
17.                 $X_{|n_i|} \leftarrow \mathrm{Sign}(n_i)\, Q^{(i)}\ \ (= \mathrm{Sign}(n_i)\phi^i(P)\,)$
18.             else
19.                 if $n_i \in \mathcal{D}^+$ then
20.                     $k \leftarrow k+1,\ (A_k, d_k) \leftarrow (\ Q^{(i)}, d)\ \ (= (\ \phi^i(P), d)\,)$
21.                 if $n_i \in -\mathcal{D}^+$ then
22.                     $k \leftarrow k+1,\ (A_k, d_k) \leftarrow (-Q^{(i)}, d)\ \ (= (-\phi^i(P), d)\,)$
23.             used[$i$] $\leftarrow$ true
24.     FLUSH, $k \leftarrow 0$
25. return $\sum_{d \in \mathcal{D}^+} d \cdot X_d$
```

and in general $p_j = \left(1 - \frac{1}{r}\right)\left(1 - \frac{2}{r}\right)\cdots\left(1 - \frac{j-1}{r}\right)$. Therefore $\mathcal{E}s = s(r,q) = 1 + \sum_{j=2}^{q} \prod_{k=1}^{j-1} \left(1 - \frac{k}{r}\right)$ where $r = 2^{w-2}$ in the sliding window case.

The following table collects the expected values of s for different positive digit set sizes in sliding window methods. The last column contains the maximal value of $\mathcal{E}s$, achieved for $q \ge r$.

$r \setminus q$	2	3	4	5	6	8	asym.
2	1.500						1.500
4	1.750	2.125	2.219				2.219
8	1.875	2.531	2.941	3.146	3.223	3.245	3.245
16	1.938	2.758	3.424	3.924	4.268	4.603	4.704
32	1.969	2.877	3.700	4.420	5.028	5.907	6.774

4.2 Analysis of the Large Memory Method (Algorithm 4)

This case is more complicated than the previous one.

Suppose that a given recording $n \equiv \sum_{i=0}^{\ell} n_i \phi^i$ has k non zero digits, and that there are r nonnegative integers $a_1 \geq a_2 \geq \ldots \geq a_r$, $r = \mathcal{D}^+$, adding up to k, with the following property: we can write $\mathcal{D}^+ = \{d_1, d_2, \ldots, d_r\}$ such that for $1 \leq i \leq r$ the digit d_i appears exactly a_i times in the recoding.

Algorithm 4 will then build a_r queues of r elements (therefore merging r inversions in one, with $3(r-1)$ additional multiplications, a_r times – if the addition formula requires only one inversion), $a_{r-1} - a_r$ queues of $r-1$ elements (therefore merging $r-1$ inversions $a_{r-1} - a_r$ times), …, will perform $a_2 - a_3$ pairs of additions simultaneously (merging each time two inversions in one inversion and 3 multiplications), and at the end perform $a_1 - a_2$ single additions. The final number of blocks will then be $a_r + (a_{r-1} - a_r) + (a_{r-2} - a_{r-1}) + \cdots + (a_2 - a_3) + (a_1 - a_2) = a_1$. If there is just one field inversion per group addition, the final number of inversions and multiplications replacing the $k-1$ inversions coming from group additions in the original scalar multiplication is a_1 inversions and $3(k - a_1)$ multiplications[1]. This quantity depends only on a_1, which is the maximum of the a_i (and on k). We are thus interested in the expected value of a_1. The computation of this value is not easy and there are asymptotic estimates for this kind of "occupancy of boxes" problem, see for example [17, Ch. II, 6]. However, this does not solve our question for small values of the parameters r and k. We now describe approaches more suitable to direct computation.

First, we drop the assumption that $a_1 \geq a_2 \geq \ldots \geq a_r$, but keep that the nonnegative integers a_i add up to k. Write $\mathbf{a} := [a_1, a_2, ..., a_r]$ and $|\mathbf{a}| = a_1 + a_2 + \ldots + a_r$. Instead of a_1 we are now interested in $\max(\mathbf{a}) := \max(a_1, ..., a_r)$. The expected value of $\max(\mathbf{a})$ is

$$\frac{1}{r^k} \sum_{\mathbf{a} \in \mathbb{N}_0^r \,:\, |\mathbf{a}| = k} \binom{k}{\mathbf{a}} \max(\mathbf{a}) \tag{3}$$

where $\binom{k}{\mathbf{a}}$ is the multinomial coefficient $\binom{k}{a_1\ a_2\ \cdots\ a_r} = \frac{k!}{a_1! a_2! \cdots a_r!}$. Expression (3) can be easily evaluated by a simple computer program, but the number of summands increases very quickly and soon gets out of control. The best way to estimate the expected value of $\max(\mathbf{a})$ is then by averaging the value obtained on a few million random recodings, i.e. by a Monte Carlo method. This can be done by a second, short computer program. We first considered "small" values of r and k, that could be handled by our computer using (3), then and we chose the number of experiments for the second computer program so that the two outcomes agreed to at least 4 decimal places; then, we computed other expected values for $\max(\mathbf{a})$ using only the second program.

In the next table we display the results for some values of ℓ and $r = 2^{w-2}$, where k is given as the closest integer to $\frac{\ell}{w+1} - r$ – in other words we are using

[1] If there are t inversions per groups addition, we replace $t(k-1)$ inversions by ta_1 inversions and $3t(k - a_1)$ multiplications. This case may occur in some circumstances if a formula with more inversions is less expensive than a formula with less inversions but a much larger number of multiplications.

width-w signed sliding window methods (binary or τ-adic) and take into account the fact that the first addition to each accumulator register is just an assignment. We provide the expected queue length $\mathcal{E}q = k/\mathcal{E}\max(\mathbf{a})$, too.

ℓ	$w=3$ $(r=2)$			$w=4$ $(r=4)$			$w=5$ $(r=8)$			$w=6$ $(r=16)$			$w=7$ $(r=32)$		
	k	$\mathcal{E}\max(\mathbf{a})$	$\mathcal{E}q$	k	$\mathcal{E}\max(\mathbf{a})$	$\mathcal{E}q$	k	$\mathcal{E}\max(\mathbf{a})$	$\mathcal{E}q$	k	$\mathcal{E}\max(\mathbf{a})$	$\mathcal{E}q$	k	$\mathcal{E}\max(\mathbf{a})$	$\mathcal{E}q$
160	38	21.443	1.772	28	9.806	2.856	19	4.764	3.989	7	1.905	3.675	4	1.181	3.386
192	46	25.691	1.791	35	11.877	2.947	24	5.662	4.239	12	2.575	4.660	8	1.665	4.806
240	58	32.025	1.811	44	14.497	3.035	32	7.047	4.541	18	3.320	5.422	14	2.247	6.231
256	62	34.129	1.817	48	15.649	3.067	35	7.554	4.663	21	3.657	5.742	16	2.392	6.688
320	78	42.512	1.835	60	19.071	3.146	45	9.207	4.888	30	4.625	6.486	24	2.990	8.026
512	126	67.469	1.868	98	29.681	3.302	75	13.944	5.378	57	7.244	7.869	48	4.515	10.632

Note that when using the *minimum* of $\mathbf{a}$ in place of the maximum in all the computations we just described we get the expected number of "full" queues, i.e. those of length r. The computations show that with overwhelming probability there is always at least one such queue.

5 Practical Aspects

5.1 Optimizing the Precomputation and Accumulation Steps

One potential performance problem with scalar multiplication methods derived from Yao's is the final accumulation step, i.e. the computation of $\sum_{d\in\mathcal{D}^+} X_d$, especially for larger window sizes.

5.1.1 Precomputations

For the precomputation phase for ϕ-and-add methods there is a very efficient method by Cohen (see [12]). In general we have a total of $2^{w-2}-1$ group additions and 1 group doubling fused in $w-1$ groups, in other words requiring just $w-1$ inversions. It has been described originally for EC with doubling, but can be used also with halving, and HEC.

Other coordinate systems for the precomputations can be chosen, but only if inversion is extremely expensive; for example using affine coordinates in the precomputation phase with Cohen's approach is the best choice if $\mathtt{I} < 33.9\mathtt{M}$ with a key length of 192 bits.

Another classical precomputation strategy requires just 2 inversions. We first compute $2P$ in affine coordinates and then get $3P = P + 2P, \ldots, (2n+1)P = (2n-1)P+2P$ each as a mixed coordinate addition (Jacobian coordinates for odd characteristic EC and Lopez-Dahab coordinates for even characteristic EC). To get back the precomputations in affine coordinates, all Z-coordinates are inverted together by Montgomery's trick.

For our values of the I/M ratio Cohen's method is almost consistently the best precomputation strategy.

5.1.2 Accumulation

We present a technique that applies to the final accumulation step of Yao's algorithm. Since the digit set for doubling and halving based methods for EC and HEC is the same, we give a unitary treatment. Before proceeding we introduce some notation.

Notation 2. $\mathtt{M}$, *resp.* $\mathtt{I}$, $\mathtt{S}$ *denote field multiplication, resp. inversion, squaring.* $\mathtt{A}$ *and* $\mathtt{D}$ *mean group addition and doubling.* $\mathtt{A}[i]$, *resp.* $\mathtt{D}[i]$, *denote* $i \geq 1$ *simultaneous group additions, resp. doublings.*

For any two integers a, b we can write $\mathbf{X}_{a,b} = X_a + X_b$. By $\mathbf{X}_{[a\,..\,b]}$ we understand the sum $X_a + X_{a+2} + \ldots + X_b$ for $a < b$ and a, b odd. The length of a sum of the form $\mathbf{X}_{[a\,..\,b]}$ is the number of summands X_i it contains.

Let $N_{\mathtt{A}}$, resp $N_{\mathtt{D}}$, N_G be the number of $\mathtt{A}$s, $\mathtt{D}$s and groups of operations in an algorithm. Under the assumption that $\mathtt{A}$ and $\mathtt{D}$ both contain just one inversion, the operation count will be

$$N_{\mathtt{A}}(\mathtt{A}-\mathtt{I}) + N_{\mathtt{D}}(\mathtt{D}-\mathtt{I}) + N_G\mathtt{I} + 3(N_{\mathtt{A}} + N_{\mathtt{D}} - N_G)\mathtt{M}\ . \tag{4}$$

(The case where $\mathtt{A}$ and $\mathtt{D}$ contain more inversions is left to the reader.)

Here and in what follows $t = 2^{w-1}$. The standard approach consists in computing X_{t-1}, $\mathbf{X}_{t-3,t-1} = X_{t-3} + X_{t-1}$, $\mathbf{X}_{[t-5\,..\,t-1]} = X_{t-5} + X_{t-3} + X_{t-1}$,..., $\mathbf{X}_{[1\,..\,t-1]} = X_1 + X_3 + \ldots + X_{t-1}$ by successive additions, add all these sums but the last one together, double the result and add it to the last sum. This requires $2^{w-1} - 1$ group operations (one is a doubling).

If $w = 3$ then there are just two registers, X_1 and X_3 and the computation of $X_1 + 3X_3$ is done by first computing $X_1 + X_3$ and $2X_3$ at the same time, merging one addition and one doubling, followed by another addition. For EC this takes $2\mathtt{I} + 12\mathtt{M} + 3\mathtt{S}$ (and one more $\mathtt{S}$ if the characteristic is odd). Using different coordinate systems we can first compute $X_1 + X_3$ and $2X_3$ in Jacobian coordinates then add the results and convert them into affine coordinates. This takes $1\mathtt{I} + 21\mathtt{M} + 11\mathtt{S}$ in odd characteristic and $1\mathtt{I} + 23\mathtt{M} + 5\mathtt{S}$ in characteristic 2. The first method is faster for EC in odd characteristic if $\mathtt{I} < 11\mathtt{M}$ and $\mathtt{I} < 12\mathtt{M}$ in characteristic 2.

For all $w \geq 4$ we present a simple strategy by which we can compute the final expression with $3 \cdot 2^{w-2} - w$ group operations (one is a doubling), by fusing them in $3w - 5$ groups. On the other hand, the classic approach requires $2^{w-1} - 1$ group additions (one is a doubling).

We first compute $\mathbf{X}_{1,3}$, $\mathbf{X}_{5,7}$, ..., $\mathbf{X}_{t-3,t-1}$ (additions of adjacent elements in the list $X_1, X_3, \ldots, X_{t-1}$) by means of one $\mathtt{A}[2^{w-3}]$, then $\mathbf{X}_{[1\,..\,7]}$, $\mathbf{X}_{[9\,..\,15]}$, ..., $\mathbf{X}_{[(t-7)\,..\,(t-1)]}$ (quadruplets of adjacent elements) by one $\mathtt{A}[2^{w-4}]$, and so on until we get the length 2^{w-2} sum $\mathbf{X}_{[1\,..\,(t-1)]}$ by one $\mathtt{A}$.

The second phase of our strategy consists in calculating the sums $\mathbf{X}_{[i\,..\,(t-1)]}$ for all odd i with $3 \leq i \leq t-3$.

The last sum which we just computed involves 2^{w-2} summands X_i. All the partial sums which we still require and whose length is a multiple of 2^{w-3} (namely, $\mathbf{X}_{[1\,..\,(t-1)]}$ and $\mathbf{X}_{[(t/2+1)\,..\,(t-1)]}$) are already known.

We first get the sums of a number of terms which is multiple of 2^{w-4}: the first one, the last and in general all those whose length is a power of two are already known. One sum is to be computed (cost: one $\mathtt{A}$).

Then we compute the $3 = 2^2 - 1$ sums of length multiple of 2^{w-5} with one $\mathtt{A}[3]$. The next step is to compute the $7 = 2^3 - 1$ sums of length multiple of 2^{w-6} with one $\mathtt{A}[7]$. We proceed this way until we compute the last $2^{w-3} - 1$ sums of an odd number of elements with an $\mathtt{A}[2^{w-3} - 1]$.

As an example, to get $\mathbf{X}_{[3\,..\,(t-1)]}$ we first need to add $\mathbf{X}_{[(t/4+1)\,..\,(t/2-1)]}$ to $\mathbf{X}_{[(t/2+1)\,..\,(t-1)]}$, to get $\mathbf{X}_{[(t/4+1)\,..\,(t-1)]}$, then we add $\mathbf{X}_{[(t/8+1)\,..\,(t/4-1)]}$ to this sum, $\mathbf{X}_{[(t/18+1)\,..\,(t/8-1)]}$, and so on, until we just add X_3 to $\mathbf{X}_{[5\,..\,(t-1)]}$. All partial summands $\mathbf{X}_{[i\,..\,(t-1)]}$ for all odd i with $3 \le i \le t-1$ are obtained this way, with the sums of addends $\mathbf{X}_{[a\,..\,b]}$ of the same length done in parallel.

The last phase consists in adding all the sums $\mathbf{X}_{[i\,..\,(t-1)]}$ for all odd i with $3 \le i \le t-1$ together. This can be done in a binary tree fashion, by $\mathtt{A}[2^{w-3} - 1]$ (note that $\mathbf{X}_{[1\,..\,(t-1)]}$ is not added), $\mathtt{A}[2^{w-4}]$, $\mathtt{A}[2^{w-5}]$, …, $\mathtt{A}[2]$, $\mathtt{A}$, this sum is then doubled and added to $\mathbf{X}_{[1\,..\,(t-1)]}$.

It is easy to see that the total operation count is $3 \cdot 2^{w-2} - w$ additions (of which one is in fact a doubling), fused in $3w - 5$ groups: the latter number, multiplied by the number of $\mathtt{M}$ per group operation (usually 1), is the number of $\mathtt{I}$ effectively computed. This number is usually much lower than $2^{w-1} - 1$ for the classical approach.

For $4 \le w \le 6$ it is easy to devise faster ad-hoc groupings of operations (possibly using mixed coordinates). For $3 \le w \le 8$ the groupings and the operation counts for EC are given in the following table. We also provide the ratio $\mathtt{I}/\mathtt{M}$, over which our method is faster than the classical approach.

w	Classic approach	Our method(s)		Example: Elliptic Curves *Field operation counts*		*Thresholds*	
	Ops.	*Ops.*	*Groups*	Classic approach	Our method(s)	*odd char.*	*even char.*
3	3	3 –	2 –	$3\mathtt{I} + 6\mathtt{M} + 3\{+1\}\mathtt{S}$	$2\mathtt{I} + 9\mathtt{M} + 3\{+1\}\mathtt{S}$ $1\mathtt{I} + 21\{+2\}\mathtt{M} + 11\{+2\}\mathtt{S}$	3 9.50	3 9.50
4	7	9	6	$7\mathtt{I} + 14\mathtt{M} + 7\{+1\}\mathtt{S}$	$6\mathtt{I} + 27\mathtt{M} + 9\{+1\}\mathtt{S}$	13.50	13.40
5	15	20	8	$15\mathtt{I} + 30\mathtt{M} + 15\{+1\}\mathtt{S}$	$8\mathtt{I} + 76\mathtt{M} + 20\{+1\}\mathtt{S}$	6.93	6.71
6	31	40	12	$31\mathtt{I} + 62\mathtt{M} + 31\{+1\}\mathtt{S}$	$12\mathtt{I} + 164\mathtt{M} + 40\{+1\}\mathtt{S}$	5.60	5.46
7	63	89	16	$63\mathtt{I} + 126\mathtt{M} + 63\{+1\}\mathtt{S}$	$16\mathtt{I} + 397\mathtt{M} + 89\{+1\}\mathtt{S}$	6.04	5.88
8	127	184	19	$127\mathtt{I} + 254\mathtt{M} + 127\{+1\}\mathtt{S}$	$19\mathtt{I} + 863\mathtt{M} + 184\{+1\}\mathtt{S}$	5.90	5.74

Note: $\{+1\}$ means "add 1" in odd characteristic.

5.2 Operation Counts

We consider here the cost of different scalar multiplication algorithms on (hyper)elliptic curves. Our references are [10, §§ 13.2, 13.3, 15.1] and [5].

5.2.1 Binary Elliptic Curves with Point Doubling

Let us review the costs for a scalar multiplication (including precomputation) with the double-and-add method. Operation counts are from [10, §§ 13.2, 13.3].

Let $n = \sum_{i=0}^{\ell} n_i 2^i$ be the scalar, and w the window width. Put $\ell_1 = \ell - (w-1)/2$ and $K = 1/2 - 1/(w+1)$. On average $\ell_1 + K$ doublings and $v = (\ell_1 - K)/(w+1)$ additions are used. If only affine coordinates $\mathcal{A}$ are used, then an auxiliary system $\mathcal{A}'$ is introduced, v doublings use a special form ($2\mathcal{A}' = \mathcal{A}'$) that requires only $\mathtt{M} + 2\mathtt{S}$, whereas the remaining ones (of the form $2\mathcal{A}' = \mathcal{A}$) cost $\mathtt{I} + \mathtt{M} + \mathtt{S}$. The additions require the ability of adding two points in $\mathcal{A}$ with a result in $\mathcal{A}'$, and cost $2\mathtt{I} + 3\mathtt{M} + \mathtt{S}$. In the main loop we need about $(\ell_1 + K)(\mathtt{I}+\mathtt{M}+\mathtt{S}) + \frac{w+1}{\ell_1 - K}(3\mathtt{M}+2\mathtt{S})$ to compute $n \cdot P$. If we do not use inversions, we perform the computation using López-Dahab coordinates, and the cost is $(\ell_1 + K)(4\mathtt{M}+4\mathtt{S}+\mathtt{M}_2) + \frac{w+1}{\ell_1 - K}(8\mathtt{M}+5\mathtt{S}+\mathtt{M}_2) + (\mathtt{I}+2\mathtt{M})$ where $\mathtt{M}_2$ denotes multiplication by a fixed value. Note that there is a final conversion to affine coordinates. The precomputation phase is done as described in § 5.1.1.

With our algorithms we perform $\ell_1 + K$ doublings and $\widehat{v} := v - 2^{w-2}$ additions (recall that the first time a digit appears, a group addition is replaced by an assignment). The additions come in $\widehat{v}/\mathcal{E}s$ groups of merged additions, and the concrete values of $\mathcal{E}s$ are taken from the tables in the previous section.

Both in the Conservative Memory algorithm (§ 3.1) and in the Large Memory algorithm (§ 3.2) we perform the doublings in the coordinate system that has fastest doublings, i.e. López-Dahab's ($\mathcal{LD}$) and then convert the v points that get added (or written) to the accumulator registers to $\mathcal{A}$. The conversions are done in parallel (with some obvious modifications to the algorithms): in the Conservative Memory algorithm only $\mathcal{E}s$ points at a time are converted, and in the Large Memory algorithm all the v points are converted simultaneously. The cost of the main loop is about

$$(\ell_1{+}K)(4\mathtt{M}{+}4\mathtt{S}{+}\mathtt{M}_2){+}\Big(\frac{v}{\mathcal{E}s}\,(\mathtt{I} + 3(\mathcal{E}s - 1)\mathtt{M}) + 2v\,\mathtt{M}\Big){+}\Big(\frac{\widehat{v}}{\mathcal{E}s}\,(\mathtt{I} + 3(\mathcal{E}s - 1)\mathtt{M}) + \widehat{v}(2\mathtt{M} + \mathtt{S})\Big)$$

in the Conservative Memory algorithm and

$$(\ell_1 + K)(4\mathtt{M} + 4\mathtt{S} + \mathtt{M}_2) + (\mathtt{I} + 3(v-1)\mathtt{M} + 2v\,\mathtt{M}) + \left(\frac{\widehat{v}}{\mathcal{E}s}\,(\mathtt{I} + 3(\mathcal{E}s - 1)\mathtt{M}) + \widehat{v}(2\mathtt{M} + \mathtt{S})\right)$$

in the Large Memory algorithm. The cost of the final accumulation step is given in §5.1.

5.2.2 Using Point Halving on Elliptic Curves

Let n, ℓ, w, ℓ_1, v as above. We use here point halving, which consists, given a point P, in finding a point Q such that $2Q = P$. It is denoted by $\mathtt{H}$. Operation counts are essentially as before, where halvings replace doublings in the main loops, but the precomputation and the final accumulation stages retain the doubling. Only affine coordinates are used, following [18,25]. According to [14], halving is about twice as fast as a doubling. The classic halve-and-add method costs then $(\ell_1 + K)\mathtt{H} + v\,\mathtt{A}$ in the main loop, the precomputations being as in §5.2.1.

5.2.3 Elliptic Koblitz Curves

The reasoning is the same as in the previous Subsections, where doubling and halving are here replaced by Frobenius operations. As in § 5.2.1 we will determine operation counts when affine coordinates are used as well as with $\mathcal{LD}$

coordinates. The recoding used is a windowed τ-adic recoding using the digit set by Solinas [26,27]. Precomputation and final accumulation steps have to be adapted to the new context of τ-adic digit sets, essentially ad-hoc for each w: the difference in the operations counts are just a few Frobenius operations.

5.2.4 Elliptic Curves in Odd Characteristic

In odd characteristic the situation is entirely similar to that of binary Elliptic curves where point doubling is used. The details are similar and are thus omitted.

The coordinate systems used for the Horner scheme are: affine only, and a mixed coordinate approach involving precomputed points in $\mathcal{A}$ and intermediate computations done using Jacobian ($\mathcal{J}$) and Jacobian modified ($\mathcal{J}^m$) coordinates.

In our algorithms we perform the doublings in $\mathcal{J}^m$, since they have the fastest doubling, and then convert the required results to $\mathcal{A}$.

5.2.5 Higher Genus Curves

We consider here curves of genus 2 and 3 in even characteristic with the formulae of [21] in genus 2 and those of [5] for genus 3. For genus 2 a group addition, resp. doubling, costs $\mathtt{I} + 22\mathtt{M} + 3\mathtt{S}$, resp. $\mathtt{I} + 6\mathtt{M} + 5\mathtt{S}$. For genus 3 these operations cost $\mathtt{I} + 47.7\mathtt{M} + 6\mathtt{S}$, resp. $\mathtt{I} + 9.3\mathtt{M} + 11\mathtt{S}$. Apart from the operation counts for the group operations, the computations are performed as in the EC case with affine coordinates.

5.3 Comparisons and Results

Here we collect the operation counts relative to a field multiplication for several types of groups at different security levels. We compare the classical techniques (Algorithm 1), denoted by the label "Classic" with our algorithms, and for Koblitz curves also with the methods in [4]. CMA stands for *Conservative Memory Algorithm* (Algorithm 3) and LMA for *Large Memory Algorithm* (Algorithm 4). In some cases the "Classic" column contains two subcolumns: The one labeled $\mathcal{A}$ refers to computations entirely done in affine coordinates, whereas $\mathcal{LD}$, resp. $\mathcal{J}^m$ refers to the usage of López-Dahab, resp. Jacobian modified coordinates. The relative costs of the field operations in the even characteristic are taken from the implementations reported [3,5]. For the ratios in the odd characteristic case we refer to [1]: for example for fields of 160, 192 and 256 bits we have $\mathtt{I}/\mathtt{M}$ ratios around 30. The bit size of the groups in the table for HEC is approximative, and the field of definition is given, too.

The table for elliptic Koblitz Curves contains more comparisons to put the results of this paper in a broader perspective. Firstly, some curves whose size is outside the scope of this paper are considered: the sizes are those of the curves K-163, K-233, K-283, K-409 and K-571 as standardised by NIST [29]. Secondly, as already mentioned we compare also with two algorithms from [4] – for the details we refer the reader to the cited paper.

Clearly, we get mixed results. In some circumstances the new algorithms perform significantly better than the previous ones, in other cases the performance is similar, or worse. The Large Memory algorithm performs better than the Conservative Memory approach, but the differences are often quite small.

Elliptic Curves in Even Characteristic							
bits	Elliptic Curve, Doubling				Elliptic Curve, Halving		
	Classic $\mathcal{A}$	Classic $\mathcal{LD}$	CMA	LMA	Classic	CMA	LMA
163	1840.17	1144.70	1254.79	1213.14	853.57	773.61	768.22
199	2532.87	1383.57	1537.93	1474.85	989.05	870.29	859.46
233	3131.90	1583.06	1756.22	1699.34	1174.77	1022.08	1002.11

Elliptic Koblitz Curves						
bits	Classic		This paper		From [4]	
	$\mathcal{A}$	$\mathcal{LD}$	CMA	LMA	Alg. 3	Alg. 4
163	390.77	406.55	305.75	300.35	359.59	376.40
233	601.41	523.13	427.69	407.71	610.61	500.23
283	1040.88	605.06	639.28	568.96	928.49	585.94
409	3122.46	832.23	1266.51	1144.68	2719.01	801.94
571	7633.01	1311.20	2746.00	2428.36	6340.55	1058.10

Elliptic Curves in Odd Char.				
bits	Classic $\mathcal{A}$	Classic $\mathcal{J}^m$	CMA	LMA
160	6378.20	1422.42	1825.50	1528.52
192	7472.86	1670.41	2099.27	1787.72
256	9794.09	2166.41	2623.72	2321.92

Hyperelliptic Curves in Even Characteristic								
bits	Genus 2				Genus 3			
	Field	Classic	CMA	LMA	Field	Classic	CMA	LMA
160	$\mathbb{F}_{83}$	2877.14	2883.70	2882.04	$\mathbb{F}_{53}$	4394.68	4437.62	4435.47
192	$\mathbb{F}_{97}$	3300.71	3291.20	3287.44	$\mathbb{F}_{67}$	5219.82	5252.92	5248.99
256	$\mathbb{F}_{127}$	4146.97	4138.74	4133.97	$\mathbb{F}_{89}$	6531.11	6578.93	6575.54

For EC scalar multiplication with point doubling, a lot of research went into refined coordinate systems (that can dispense with inversions), and this explains why our approach, that is based for the most part on “good old” affine coordinates that require inversions, does not win. For HEC in even characteristic, that also profit from very refined doubling formulae, the situation is similar.

Halving and Frobenius methods profit the most because they use relatively many inversions that can be effectively reduced by our techniques. Savings of 10% to 15% (for halving methods) to 25-32% (for Frobenius methods) are not uncommon. In particular, for generic binary EC, halving brings noteworthy speedups, our methods being as much as 30 to 40% faster than the mixed coordinates implementation of the Horner scheme. We also confirm Schroeppel’s claims that a scalar multiplication based on halving is usually (at least) twice as fast than one based on doubling in affine coordinates.

For Koblitz curves, the methods from [4] are already faster than classical methods for small curves, yet slower than ours for the sizes that are in the scope of the present paper. For larger sizes, the CMA and LMA methods lose ground and become slower than classic methods (starting from the 409 bit level). However, the inversion-free sublinear Algorithm 4 from [4], quickly reveals itself as the fastest scalar multiplication method for Koblitz curves of at least 409 bits, with speedups reaching 24% with respect to the mixed coordinates τ-adic Horner scheme for 571 bit curves.

6 Conclusions

In this paper we showed that Yao's algorithm presents an intrinsic parallelism that permits us to merge some of the group operations it performs. Two variants of this method are given: one adopts a strategy of augmenting memory usage in a conservative way, and is given as Algorithm 3; the second method, Algorithm 4, uses more memory but is also more efficient.

A simple analysis has been done. This is then used to estimate the costs of scalar multiplications for different groups used in cryptographic applications: EC over odd and even characteristic fields (in this case we also chose parameters that are presented in standards and have been considered by several authors for performance improvements), using various scalar multiplication techniques, as well as low genus HEC.

The techniques presented here can significantly improve the performance for EC cryptosystems in even characteristic when point halving is used, as well as Koblitz curve Cryptosystems. The gains in these cases can be estimated in the 10-15%, resp. 25-32% range with common timing ratios of field operations with respect to the field multiplication.

Acknowledgements. Part of this paper was written while visiting the Department of Mathematical Sciences, Stellenbosch University. I wish to express my gratitude to this institution for its hospitality. I am grateful to Johann Großschädl, Clemens Heuberger, Helmut Prodinger, Francesco Sica, Douglas Stabila and Nicolas Thériault for interesting discussions and remarks.

Disclaimer. The information in this document reflects only the author's views, is provided as is and no guarantee or warranty is given that the information is fit for any particular purpose. The user thereof uses the information at its sole risk and liability.

References

1. Avanzi, R.M.: Aspects of hyperelliptic curves over large prime fields in software implementations. In: Joye, M., Quisquater, J.-J. (eds.) CHES 2004. LNCS, vol. 3156, pp. 148–162. Springer, Heidelberg (2004)
2. Avanzi, R.M.: A Note on the Signed Sliding Window Integer Recoding and its Left-to-Right Analogue. In: Handschuh, H., Hasan, M.A. (eds.) SAC 2004. LNCS, vol. 3357, pp. 130–143. Springer, Heidelberg (2004)

3. Avanzi, R.M., Ciet, M., Sica, F.: Faster Scalar Multiplication on Koblitz Curves combining Point Halving with the Frobenius Endomorphism. In: Bao, F., Deng, R., Zhou, J. (eds.) PKC 2004. LNCS, vol. 2947, pp. 28–40. Springer, Heidelberg (2004)
4. Avanzi, R.M., Heuberger, C., Prodinger, H.: On Redundant τ-adic Expansions and Non-Adjacent Digit Sets. LNCS. vol. 4356, pp. 285–301, Springer, Heidelberg (to appear)
5. Avanzi, R.M., Thériault, N., Wang, Z.: Rethinking Low Genus Hyperelliptic Jacobian Arithmetic over Binary Fields: Interplay of Field Arithmetic and Explicit Formulae. CACR report 2006-07, Available at http://www.cacr.math.uwaterloo.ca/tech_reports.html
6. Barreto, P., Kim, H., Lynn, B., Scott, M.: Efficient Algorithms for Pairing-Based Cryptosystems. In: Yung, M. (ed.) CRYPTO 2002. LNCS, vol. 2442, pp. 354–368. Springer, Heidelberg (2002)
7. Ciet, M., Joye, M., Lauter, K., Montgomery, P.L.: Trading Inversions for Multiplications in Elliptic Curve Cryptography. Designs Codes and Cryptography 39(2), 189–206 (2006)
8. Clark, W.E., Liang, J.J.: On arithmetic weight for a general radix representation of integers. IEEE Transactions on Information Theory IT- 19, 823–826 (1973)
9. Cohen, H.: A course in computational algebraic number theory. Graduate Texts in Math. 138, Springer, Heidleberg, 1993, Third corrected printing (1996)
10. Cohen, H., Frey, G. (eds.): The Handbook of Elliptic and Hyperelliptic Curve Cryptography. CRC Press, Boca Raton (2005)
11. Cohen, H., Miyaji, A., Ono, T.: Efficient elliptic curve exponentiation. In: Han, Y., Quing, S. (eds.) ICICS 1997. LNCS, vol. 1334, pp. 282–290. Springer, Heidelberg (1997)
12. Cohen, H., Miyaji, A., Ono, T.: Efficient Elliptic Curve Exponentiation Using Mixed Coordinates. In: Ohta, K., Pei, D. (eds.) ASIACRYPT 1998. LNCS, vol. 1514, pp. 51–65. Springer, Heidelberg (1998)
13. Duursma, I., Lee, H-S.: Tate Pairing Implementation for Hyperelliptic Curves $y^2 = x^p - x + d$. In: Laih, C.-S. (ed.) ASIACRYPT 2003. LNCS, vol. 2894, pp. 111–123. Springer, Heidelberg (2003)
14. Fong, K., Hankerson, D., López, J., Menezes, A.: Field Inversion and Point Halving Revisited. IEEE Trans. Computers 53(8), 1047–1059 (2004)
15. Galbraith, S., Harrison, K., Soldera, D.: Implementing the Tate pairing. In: Fieker, C., Kohel, D.R. (eds.) Algorithmic Number Theory. LNCS, vol. 2369, pp. 324–337. Springer, Heidelberg (2002)
16. Koblitz, N.: Hyperelliptic cryptosystems. J. Cryptology 1, 139–150 (1989)
17. Kolchin, V.F., Sevast'yanov, B.A., Chistyakov, V.P.: Random Allocations. V.H. Winston and Sons, Washington DC (1978)
18. Knudsen, E.W.: Elliptic Scalar Multiplication Using Point Halving. In: Lam, K.-Y., Okamoto, E., Xing, C. (eds.) ASIACRYPT 1999. LNCS, vol. 1716, pp. 135–149. Springer, Heidelberg (1999)
19. Koblitz, N.: CM-curves with good cryptographic properties. In: Feigenbaum, J. (ed.) CRYPTO 1991. LNCS, vol. 576, pp. 279–287. Springer, Heidelberg (1992)
20. Lange, T.: Formulae for arithmetic on genus 2 hyperelliptic curves. Appl. Algebra Engrg. Comm. Comput. 15(5), 295–328 (2005)
21. Lange, T., Stevens, M.: Efficient doubling for genus two curves over binary fields. In: Handschuh, H., Hasan, M.A. (eds.) SAC 2004. LNCS, vol. 3357, pp. 170–181. Springer, Berlin (2004)

22. Mishra, P.K., Sarkar, P.: Application of Montgomery's Trick to Scalar Multiplication for Elliptic and Hyperelliptic Curves Using a Fixed Base Point. In: Bao, F., Deng, R., Zhou, J. (eds.) PKC 2004. LNCS, vol. 2947, pp. 41–54. Springer, Heidelberg (2004)
23. Möller, B.: Algorithms for Multi-exponentiation. In: Vaudenay, S., Youssef, A.M. (eds.) SAC 2001. LNCS, vol. 2259, pp. 165–180. Springer, Heidelberg (2001)
24. Reitwiesner, G.W.: Binary arithmetic. Advances in Computers 1, 231–308 (1960)
25. Schroeppel, R.: Elliptic curve point ambiguity resolution apparatus and method. International Application Number PCT/US00/31014, filed (November 9, 2000)
26. Solinas, J.A.: An improved algorithm for arithmetic on a family of elliptic curves. In: Kaliski Jr., B.S. (ed.) CRYPTO 1997. LNCS, vol. 1294, pp. 357–371. Springer, Heidelberg (1997)
27. Solinas, J.A.: Efficient Arithmetic on Koblitz Curves. Codes and Cryptography 19(2/3), 125–179 (2000)
28. Yao, A.C.: On the evaluation of powers. SIAM J. Comp. 5, 100–103 (1976)
29. National Institute of Standards and Technology. Digital Signature Standard. FIPS Publication 186-2 (February 2000)

On the Problem of Finding Linear Approximations and Cryptanalysis of Pomaranch Version 2[*]

Martin Hell and Thomas Johansson

Dept. of Information Technology, Lund University,
P.O. Box 118, 221 00 Lund, Sweden
{martin,thomas}@it.lth.se

Abstract. We give a simple algorithm that can find biased linear approximations of nonlinear building blocks. The algorithm is useful if the building block is relatively small and exhaustive search is possible. Instead of searching all possible linear relations individually, we show how the most biased relation can be found in just a few steps. As an example we show how we can find a biased relation in the output bits of the stream cipher Pomaranch Version 2, a tweaked variant of Pomaranch, resulting in both distinguishing and key recovery attacks. These attacks will break both the 128-bit variant and the 80-bit variant of the cipher with complexity faster than exhaustive key search.

Keywords: cryptanalysis, linear approximation, Pomaranch, stream ciphers.

1 Introduction

Finding a biased linear approximation of some output bits of a stream cipher can be devastating for the security of the cipher. If the bias of the approximation is sufficiently strong a distinguishing attack can be mounted. In some cases this distinguishing attack can be modified into a key recovery attack which will succeed with computational complexity much lower than exhaustive key search. The process of finding the best approximation usually requires considerable effort from the cryptanalyst. It can be the case that the cipher design does not allow for a comprehensive theoretic analysis and thus a possible biased linear approximation is not found, though it exists. In this paper we suggest a method that finds a linear approximation without considering theoretical aspects of the building blocks of a cipher. The algorithm is suitable if the cipher consists of small building blocks and if the blocks are difficult to analyze theoretically. As

[*] The work described in this paper has been supported in part by the European Commission through the IST Programme under Contract IST-2002-507932 ECRYPT. The information in this document reflects only the author's views, is provided as is and no guarantee or warranty is given that the information is fit for any particular purpose. The user thereof uses the information at its sole risk and liability.

© Springer-Verlag Berlin Heidelberg 2007

an example we show how to find a linear approximation in the output bits of the cipher Pomaranch Version 2.

Pomaranch [4] is one of the candidates in the eSTREAM [3] project. It is designed to be efficient both in hardware and software. It is one of the ciphers that has been moved to the second phase in the hardware category in the eSTREAM project. The cipher is an interesting construction since it introduces a new approach to the design of LFSR based stream ciphers. One common way to introduce nonlinearity into the linearly generated LFSR sequence is to clock the LFSR in an irregular way. Doing this will result in a decimated sequence and the number of bits discarded between consecutive output bits is unknown to an attacker. However, this will introduce some problems. First, an output buffer is needed since the keystream bits will not be produced regularly. Second, for the same reason, the construction is likely to be vulnerable to timing and power attacks. In Pomaranch, the LFSRs are based on a new idea, called jump registers. In each update of the LFSRs, the next state is one of two possible states. Which one it jumps to is key dependent and thus unknown to the attacker. The main advantage is that the update of the ciphers behaves like irregular clocking but the problems involved in irregular clocking are avoided.

Two attacks have been proposed on the first version of Pomaranch. The first attack is a chosen IV attack [1] which succeeds because not all IV bits are diffused into the whole state in the initialization phase. The second attack [8] is a correlation attack on the keystream generation of Pomaranch. This attack uses a biased linear relation in the output bits to mount a key recovery attack. Consequently, because of these two attacks, Pomaranch had to be tweaked. This was done and the tweaked version, referred to as Pomaranch Version 2, was proposed in [5] and [7]. Pomaranch has a key size of 128 bits but since hardware dedicated ciphers are required to support 80 bit keys in the eSTREAM project, a reduced variant was also proposed, supporting 80 bit security level. By using the proposed algorithm we show that the changes made in the tweaked version were not enough. It is still possible to find a linear relation in the output bits that is biased enough to allow distinguishing and key recovery attacks. These attacks are the first on Pomaranch Version 2 and they will break both the 128-bit variant and the 80-bit variant of the cipher with complexity faster than exhaustive key search.

This paper is outlined as follows. Section 2 gives some background theory concerning the problem of binary hypothesis testing. In Section 3 we consider the case of letting a binary linear approximation be written as a binary vector. Section 4 presents an algorithm that finds a linear approximation in certain cases. A practical example is given in Section 5, in which the attack on Pomaranch Version 2 is described. In Section 6 some simulation results are given and in Section 7 we look at a new version of Pomaranch which was designed with our attacks in mind. Finally, the paper is concluded in Section 8.

1.1 Notation

Throughout the paper, random variables $X, Y, \ldots$ are denoted by capital letters and their domains are denoted by $\mathcal{X}, \mathcal{Y}, \ldots$. The realizations of random variables

$x \in \mathcal{X}, y \in \mathcal{Y}, \ldots$ are denoted by small letters. A distribution is denoted by D. The probability function of a random variable X following the distribution D is denoted by $Pr_D[x]$ and sometimes as just $D[x]$.

2 Hypothesis Testing and Distinguishers

In this section we give some background theory and consider the general problem of binary hypothesis testing and distinguishers. For a more thorough treatment of hypothesis testing we refer to [2]. This is a very important concept in cryptology. A distinguisher is a mathematical tool, based on a binary hypothesis tests, which is used very frequently in cryptanalysis. The distinguisher is used not only in distinguishing attacks but it is also often used in key recovery attacks.

In a binary hypothesis test we observe a collection of independent and identically distributed data. Denote the distribution of the observed data by D_{obs}. Further, we have two known distributions, D_0 and D_1. We want to decide whether the observed distribution is actually the distribution D_0 (the null hypothesis H_0) or the distribution D_1 (the alternate hypothesis H_1), knowing that one of them is the true distribution. More formally, we define a *decision rule* which is a function $\delta : \mathcal{X} \to \{0, 1\}$ such that

$$\delta = \begin{cases} 0, & D_{obs} = D_0, \\ 1, & D_{obs} = D_1. \end{cases}$$

The function δ makes a decision for each $x \in \mathcal{X}$. The decision rule divides the domain $\mathcal{X}$ into two regions denoted A and A^c. A is called the *acceptance region* of δ and corresponds to the decision to accept the null hypothesis.

There are two types of errors associated with a binary hypothesis test. We can reject the null hypothesis when it is in fact true, i.e., $\delta : \mathcal{X} \to 1$ when $D_{obs} = D_0$. This is called a type I error, or false alarm and the probability of this error is denoted α. The other alternative is that we accept the null hypothesis when the alternate hypothesis is true, i.e., $\delta : \mathcal{X} \to 0$ when $D_{obs} = D_1$. This is called a type II error, or miss. The probability of this error is denoted by β.

In a binary hypothesis test there are several things that have to be considered. Two important things are how to perform the test in an optimal way and how many samples we need in order to have certain error probabilities, α and β. How to perform the optimal test is given by the Neyman-Pearson lemma.

Lemma 1 (Neyman-Pearson). *Let $X_1, X_2, \ldots, X_m$ be drawn i.i.d. according to mass function D_{obs}. Consider the decision problem corresponding to the hypotheses $D_{obs} = D_0$ vs. $D_{obs} = D_1$. For $T \geq 0$ define a region*

$$A_m(T) = \left\{ \frac{P_0(x_1, x_2, \ldots, x_m)}{P_1(x_1, x_2, \ldots, x_m)} > T \right\}.$$

Let $\alpha_m = D_0^m(\mathcal{A}_m^c(T))$ and $\beta_m = D_1^m(\mathcal{A}_m(T))$ be the error probabilities corresponding to the decision region $\mathcal{A}_m$. Let $\mathcal{B}_m$ be any other decision region with associated error probabilities α^* and β^*. If $\alpha^* \leq \alpha$, then $\beta^* \geq \beta$.

In order to show how the error probabilities are related to the distributions and the number of samples we first define the relative entropy between two distributions.

Definition 1. *The relative entropy between two probability mass functions* $Pr_{D_0}[x]$ *and* $Pr_{D_1}[x]$ *over the same domain* $\mathcal{X}$ *is defined as*

$$D\left(Pr_{D_0}[x] \| Pr_{D_1}[x]\right) = \sum_{x \in \mathcal{X}} Pr_{D_0}[x] \log \frac{Pr_{D_0}[x]}{Pr_{D_1}[x]}.$$

In literature the relative entropy is sometimes also referred to as information divergence or Kullback-Leibler distance. For convenience and ease of reading, in the following we write the relative entropy as $D(D_0 \| D_1)$ or $D(D_0[x] \| D_1[x])$. Note that in general $D(D_0 \| D_1) \neq D(D_1 \| D_0)$.

No general expression for the error probabilities α and β exists. Hence, we know how to perform the optimal test, but we do not know the performance of the test. However, there exist asymptotic expressions for the error probabilities, i.e., expressions that hold when the number of samples is large. The relative entropy is related to the asymptotic error probabilities through Stein's lemma. Stein's lemma states that if we fix the error probability $\alpha < \epsilon$ then

$$\beta = 2^{-nD(D_0 \| D_1)}.$$

The value of α does not affect the exponential rate at which β decreases and according to Stein's lemma, this situation always occurs. Thus, the number of samples needed in a binary hypothesis test is a constant times the inverse of the relative entropy between the two distributions.

In order to have an asymptotic expression that minimizes the overall probability of error $P_e = \pi_1 \alpha + \pi_2 \beta$, where π_i are the *a priori* probabilities of the distributions, the Chernoff information, $C(D_0, D_1)$, can be used. The error probability can then be written as

$$P_e = 2^{-nC(D_0, D_1)},$$

where

$$C(D_0, D_1) = -\min_{0 \leq \lambda \leq 1} \log \left(\sum_x Pr_{D_0}^{\lambda}[x] Pr_{D_1}^{1-\lambda}[x] \right).$$

The Chernoff information between two distributions is often hard to find since we have to minimize over λ. A common way of avoiding this is to pick a value of λ, e.g., $\lambda = 0.5$. This will give a lower bound for the Chernoff information and thus an upper bound for the error probability. In this paper we will only consider the relative entropy between the distributions and the number of samples needed will be computed as

$$\text{\# Samples needed} = \frac{1}{D(D_0 \| D_1)}.$$

A larger relative entropy means fewer samples for a successful distinguisher.

3 Vectorial Representation of a Linear Approximation

In this section we consider the advantage of representing a biased linear relation as a vector instead of as a binary relation. We give two propositions which will lead us to a simple algorithm that helps us to find biased linear relations. Both propositions follow from basic information theory.

Proposition 1. *Let the vector* $(z_{t_1}, z_{t_2}, \ldots, z_{t_m})$ *follow the size* 2^m *distribution* D_0*. Let* $z = z_{t_1} \oplus z_{t_2} \oplus \ldots \oplus z_{t_m}$ *follow the marginal distribution* D'_0*. Then*

$$D\left(D_0[x] \| D_1[x]\right) \geq D\left(D'_0[x] \| D'_1[x]\right)$$

for some size 2^m *distribution* D_1 *with corresponding marginal distribution* D'_1*.*

Proof. Denote by $a_i^{(e)}$ $(0 \leq i < 2^{m-1})$ the probabilities of the vectors in D_0 with even Hamming weight and correspondingly by $a_i^{(o)}$ $(0 \leq i < 2^{m-1})$ the probabilities of the vectors with odd Hamming weight. Similarly, the probabilities of the vectors in distribution D_1 will be denoted $b_i^{(e)}$ and $b_i^{(o)}$ $(0 \leq i < 2^{m-1})$ respectively. Hence, we want to show that

$$\sum_i a_i^{(e)} \log \frac{a_i^{(e)}}{b_i^{(e)}} + \sum_i a_i^{(o)} \log \frac{a_i^{(o)}}{b_i^{(o)}} \geq \left(\sum_i a_i^{(e)}\right) \log \frac{\sum_i a_i^{(e)}}{\sum_i b_i^{(e)}} + \left(\sum_i a_i^{(o)}\right) \log \frac{\sum_i a_i^{(o)}}{\sum_i b_i^{(o)}}. \quad (1)$$

Let $E\{\cdot\}$ denote the expected value. Jensen's inequality states that for a strictly convex function f and random variable X it holds that

$$E\{f(X)\} \geq f(E\{X\}),$$

with equality if and only if X is a constant. We use the fact that $t \log t$ is a strictly convex function and introduce $\alpha_i = Pr(t \log t = t_i)$. If we consider only the first term on the left hand side and right hand side of (1) and putting $t_i = a_i^{(e)}/b_i^{(e)}$ we can write

$$\sum_i \alpha_i \frac{a_i^{(e)}}{b_i^{(e)}} \log \frac{a_i^{(e)}}{b_i^{(e)}} \geq \left(\sum_i \alpha_i \frac{a_i^{(e)}}{b_i^{(e)}}\right) \log \sum_i \alpha_i \frac{a_i^{(e)}}{b_i^{(e)}}.$$

This will hold for any choice of α_i as long as $\alpha_i \geq 0$ and $\sum_i \alpha_i = 1$. Hence we can put $\alpha_i = b_i^{(e)} / \sum_i b_i^{(e)}$ and it follows that

$$\sum_i a_i^{(e)} \log \frac{a_i^{(e)}}{b_i^{(e)}} \geq \left(\sum_i a_i^{(e)}\right) \log \frac{\sum_i a_i^{(e)}}{\sum_i b_i^{(e)}}.$$

Doing the same thing for the second terms in (1) will end the proof. □

Proposition 1 implies that we can never lose anything by considering a linear approximation as a binary vector. Moreover, if the distribution of the vectors is such that for all vectors with even Hamming weight the probability is the same and for all vectors with odd Hamming weight the probability is the same, then there is no additional gain in using vectorial representation. We continue the analysis of vectorial representation with the following proposition.

Proposition 2. *Assume that we have a binary vectorial distribution of size 2^m denoted D_0 and the size 2^m uniform distribution D_1. Adding a variable to the length m vector that is statistically independent with all other variables will not affect the relative entropy between two distributions. Furthermore, adding a variable that is correlated with other variables will increase the relative entropy.*

Proof. Let $\mathbf{x} = (x_0, x_1, \ldots, x_{m-1})$. Using the chain rule for relative entropy we write

$$D\left(D_0[\mathbf{x}, x_m] \parallel D_1[\mathbf{x}, x_m]\right) =$$
$$D\left(D_0[\mathbf{x}] \parallel D_1[\mathbf{x}]\right) + D\left(D_0[x_m|\mathbf{x}] \parallel D_1[x_m|\mathbf{x}]\right)$$

where the last term is zero if and only if $Pr_{D_0}[x_m|\mathbf{x}] = 1/2,\ x_m \in \{0,1\}$. □

In the next section we show how these results can be used to find biased linear approximations of nonlinear blocks.

4 Finding a Biased Linear Approximation

In this section the theory developed in Section 3 is used to find biased linear approximations. If we have a linear approximation $x_{t_1} \oplus x_{t_2} \oplus \ldots \oplus x_{t_\mu}$ and add a variable, $x_{t_{\mu+1}}$, that is uniformly distributed and statistically independent with the other variables, the distribution of the resulting approximation is uniform and the approximation is useless. It can not be used in a distinguisher. This is the idea behind a common design principle in stream ciphers, i.e., masking the output with some variable that is assumed to be uniformly distributed. Then the output is guaranteed to be uniform. If we instead write the approximation as a vector, $(x_{t_1}, x_{t_2}, \ldots, x_{t_\mu})$, then, according to Proposition 2, it does not matter how many uniformly distributed and independent variables we add to the vector. As long as all variables from the approximation are present, the relative entropy will never decrease and the vector can be used in a distinguisher.

Consider a cipher containing a relatively small building block B. If the distribution of the output bits, or the distribution of a linear equation of some output bits, can be found, then it is easy to search through all linear relations in order to determine which is most biased. However, as the amount of output bits to be considered increases, the number of possible equations increases exponentially. Considering m consecutive output bits there are 2^m possible equations. If no biased equation is found among these equations, one more output bit has to be considered and an additional 2^m equations has to be checked. If checking one equation requires 2^k computation steps, checking all equations involving m bits will require a computational complexity of 2^{k+m}. Instead, by considering the output bits as a vector, the computational complexity will be limited to 2^k. On the other hand, the memory complexity will be 2^m since the distribution of a length m vector needs to be kept in memory.

Assume that the building block B has a biased linear relation involving some of the output bits $x_0, x_1, \ldots, x_{m-1}$ with a significantly larger bias than any linear relation involving less than m consecutive output bits. Let $\mathbf{x_m} = (x_0, x_1, \ldots, x_{m-1})$. Then

$$D\left(D_0[\mathbf{x_m}] \parallel D_1[\mathbf{x_m}]\right) \gg D\left(D_0[\mathbf{x_{m-1}}] \parallel D_1[\mathbf{x_{m-1}}]\right). \tag{2}$$

Hence, we can start with the vector $\mathbf{x_1} = (x_0)$ and add one extra variable at a time. If, at step m, considering the vector $\mathbf{x_m} = (x_0, x_1, \ldots, x_{m-1})$, the relative entropy between the distribution of $\mathbf{x_m}$ and the uniform distribution increases significantly, we know that there is a biased linear relation involving at least the bits x_0 and x_{m-1}. In order to find the other bits in the biased linear relation we consider the vector $\mathbf{x_m^{(i)}}$ which we define as the vector $\mathbf{x_m}$ with the variable X_i removed. If

$$D\left(D_0[\mathbf{x_m^{(i)}}] \parallel D_1[\mathbf{x_m^{(i)}}]\right) \approx D\left(D_0[\mathbf{x_m}] \parallel D_1[\mathbf{x_m}]\right) \tag{3}$$

then the variable X_i is *not* present in the linear approximation. Note that we do not use equality in (3) since the variable i can be present in some biased linear relation but still not in the most biased relation. We are only interested in the most biased relation and if the variable X_i is present in this relation the decrease in relative entropy will be significant, not just approximate. Of course we can continue increasing the length of the vector, hoping to find even better linear approximations.

5 Application to the Stream Cipher Pomaranch Version 2

In this section we show how we can use the algorithm described in Section 4 to find a heavily biased linear relation in the stream cipher Pomaranch Version 2. We also show that the existence of this linear relation makes it possible to mount both distinguishing and key recovery attacks on the cipher.

5.1 Description of Pomaranch

The attack described in this paper is independent of the initialization procedure. Because of this, only the keystream generation of Pomaranch will be described here. For a more detailed description of Pomaranch we refer to [5]. The 128-bit variant of Pomaranch is based on 9 jump registers, denoted R_i $(1 \leq i \leq 9)$, see Fig 1.

All registers are identical and of length 14. Half of the register cells are normal delay shift cells and half are feedback cells. The current operation of each cell (shift or feedback) is determined by a jump control bit, JC. Each clock, every jump register is updated in one of two ways. The state is multiplied by the transition matrix A $(JC = 0)$, or it is multiplied by $A + I$ $(JC = 1)$. The state of the registers R_1 to R_8 are filtered through a key dependent function, f_{K_i}, producing outputs c_1 to c_8. The key is divided into 8 parts of 16 bits each and part i is used in f_{K_i}. The jump control for register i, denoted JC_i, is then calculated as

$$JC_i = c_1 \oplus \ldots \oplus c_{i-1}, \quad (i = 2, \ldots, 9).$$

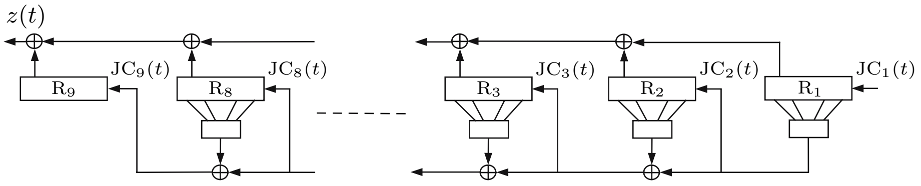


Fig. 1. Overview of Pomaranch (128-bit variant)

The jump control for the first register, JC_1 is always set to 0. The keystream at time t, denoted $z(t)$, is taken as the binary xor of all 9 bits at position 13, denoted $r_i(t)$, in the registers. The 80-bit version is identical to the 128 bit version except that the number of registers is 6 instead of 9.

The design of the second version of Pomaranch is similar to the first version. The differences are the transition matrix A and the choice of taps taken to the function f_{K_i}. Also the IV setup has been changed in order to resist the chosen IV attack in [1].

5.2 Previous Attack on Pomaranch Version 1

The size of the registers is only 14. This suggests that it might be possible to mount a divide-and-conquer kind of attack on the cipher. Since the registers are updated linearly each new output bit will be linearly dependent on the initial state bits. Hence, for any given JC-sequence of length 14 there will be a linear relation in 15 output bits that will always hold, i.e., there is an array $\ell = (\ell_0, \ell_1, \ldots, \ell_{14})$, $\ell_i \in \{0, 1\}$ such that $\sum_{i=0}^{14} \ell_i r(t+i) = 0$. It was shown in [8] that these relations are not evenly distributed among all possible 2^{14} JC-sequences of length 14. Hence, assuming that the JC-sequence is purely random, there will be linear relations in the output bits that are biased. The short register length allows for exhaustive search among all relations and the most biased relation could easily be found.

5.3 New Attack on Pomaranch Version 2

In this section we show that it is still possible, using our proposed algorithm, to find linear relations in the output bits that can be used in an attack. In [7] a theoretical analysis was done based on the attack in [8]. Let $C(x)$ be the characteristic polynomial of A. Then for a given jump control sequence we have the equality

$$\sum_{i=0}^{L} \ell_i x^{i-k_i}(x+1)^{k_i} = C(x),$$

```
for all $2^{14} - 1$ initial states
    for all $2^{i-1}$ possible $JC$-sequences
        clock the jump register $i - 1$ times
        Dist$[(z_0, \ldots, z_{i-1})]$++
    end for
end for
```

Fig. 2. Algorithm to find the distribution for i consequtive output bits. The actual implementation can be recursive to make it faster.

where k_i is the binary weight of the vector $(JC(t), \ldots, JC(t+i-1))$. Using this equality a straightforward $O(L)$ approach to find the values of ℓ_i was described, giving the linear relation for this particular jump control sequence. Hence the most common linear relation among the 2^L sequences was found in $O(L2^L)$. The problem with this analysis is that it only considers relations of length $L+1$. The same analysis is not applicable to relations involving bits further apart since the characteristic polynomial of A is of degree L. Consequently, the design parameters for Pomaranch Version 2 are optimized for the cipher to resist correlation attacks based on relations of length $L + 1$ and it successfully does so.

Unfortunately it is not enough to only consider these relations. It is possible that there are relations involving bits further apart that are more biased than any relation involving only bits $L + 1$ positions apart. The algorithm proposed in Section 4 seems very suitable for Pomaranch. The shift registers are of length 14 which is easy to search exhaustively. Moreover, all registers are identical. Register R_1 will have $JC_1(t) = 0$, $\forall t$ and will thus behave like a regularly clocked shift register with primitive feedback polynomial. Hence, the register R_1 will not be considered using our algorithm. Instead, it will be exhaustively searched. Registers R_i $(2 \leq i \leq 9)$ will have JC_i determined by a key dependent function. Since these registers are identical, finding a biased linear approximation in the output bits of one register means that all other registers (except R_1) will have this biased approximation. In order to find a good approximation, we look at vectors of consecutive output bits. When calculating the bias of these vectors, the following three assumptions will be used:

1. All states of the registers will have the same probability, except the all-zero state which has probability 0.
2. All JC-sequences will have the same probability.
3. All jump sequences, JC_i $(2 \leq i \leq 9)$, are independent.

Since the shift registers are of length 14, the first vector length we check is 15. The algorithm used to find the distribution for i consequtive output bits are given in Figure 2. The output of Pomaranch is given as the XOR of the output bits of the registers. Hence, we need to find the bias of the xor of all 8 distributions. For k

Table 1. Relative entropy between output vectors and the random distribution

Vector Length	$D(D_0 \| D_1)$
15	$2^{-111.914}$
16	$2^{-108.603}$
17	$2^{-107.671}$
18	$2^{-107.108}$
19	$2^{-75.849}$
20	$2^{-74.849}$
21	$2^{-74.264}$
22	$2^{-73.849}$
23	$2^{-73.527}$

Table 2. Relative entropy when bit i is excluded from the vector $\mathbf{x_{19}}$

i	$x_{19}^{(i)}$	i	$x_{19}^{(i)}$
1	$2^{-75.851}$	10	$2^{-75.849}$
2	$2^{-105.383}$	11	$2^{-75.849}$
3	$2^{-75.849}$	12	$2^{-75.849}$
4	$2^{-75.849}$	13	$2^{-75.849}$
5	$2^{-76.077}$	14	$2^{-75.849}$
6	$2^{-105.264}$	15	$2^{-75.849}$
7	$2^{-75.849}$	16	$2^{-75.849}$
8	$2^{-75.849}$	17	$2^{-76.849}$
9	$2^{-75.849}$		

distributions, this can easily be done in $O(k2^{2n})$ time, where n is the size in bits for each random variable. This can be a bottleneck if the vectors are very large. A much more efficient algorithm for finding the distribution of a sum of random variables was given in [9]. They show that the distribution for $Pr(X_1 \oplus X_2 \oplus \ldots \oplus X_k)$ can be found in $O(kn2^n)$ time where all X_i are n-bit random variables. This algorithm was adopted in our implementation. The relative entropies between the vectors of length 15 to 23 and the random distribution have been given in Table 1.

We see that $D(D_0 \| D_1)$ increases significantly when the vector reaches length 19. The fact that $D(D_0 \| D_1)$ is much higher for $\mathbf{x_i}$ ($i \geq 19$) tells us that there might exist a heavily biased linear approximation involving the bits x_0, x_{18} and zero or more bits x_i ($1 \leq i \leq 17$). To find the particular linear approximation that allows us to attack the cipher we look at $D\left(D_0[\mathbf{x_{19}^{(i)}}] \parallel D_1[\mathbf{x_{19}^{(i)}}]\right)$, as suggested by our proposed algorithm. The result is given in Table 2.

We see that when X_2 or X_6 are removed, the relative entropy is almost as when $\mathbf{x_{18}}$ was considered. This implies that the heavily biased linear relation is

$$z(t) \oplus z(t+2) \oplus z(t+6) \oplus z(t+18) = 0. \tag{4}$$

It is possible that the figures in Table 2 stems from the fact that the vector $(z(t), z(t+2), z(t+6), z(t+18))$ is heavily biased but the xor of the variables has a much smaller bias or even no bias at all. However, checking the relative entropy between (4) and random, which is $2^{-77.080}$, confirms that the figures in Table 2 stems from the biased linear relation. In any cipher, constructed using jump registers together with some other function, this relation would have been an important part of linear cryptanalysis. In the specific case of Pomaranch, which uses the output bits from the jump registers immediately in the output function it is even better to consider the total vector of as many bits as possible, since this will give us more information. Though, most of this information stems immediately from this relation. The discovery of the biased linear relation given by our algorithm may help the designers to get additional theoretical knowledge about the design principle of the Pomaranch family of stream ciphers.

5.4 Distinguishing and Key Recovery Attacks

In this section we give the details and complexities of the attacks on Pomaranch Version 2. Using output vectors of length 23, which is the largest vectors we were able to find the distribution for, the relative entropy $D(D_0[\mathbf{x_{23}}] \| D_1[\mathbf{x_{23}}]) = 2^{-73.527}$. The amount of keystream needed in order to distinguish these distributions is about $1/D(D_0[\mathbf{x_{23}}] \| D_1[\mathbf{x_{23}}]) = 2^{73.527}$. Note that the output of register R_1, which always has its jump control sequence $JC_1(t) = 0 \ \forall t$, is not considered in these distributions. Instead the starting state of this register is guessed in the attack. This will add a factor of 2^{14} to the computational complexity. Hence, a distinguishing attack on Pomaranch Version 2 can be mounted using $2^{73.527}$ keystream bits and a computational complexity of $2^{73.527}2^{14} = 2^{87.527}$.

This proposed distinguisher can be turned into a key recovery attack by guessing 16 bits of the key at a time. By mounting the distinguishing attack we will obtain the state of register R_1. In the next step we guess the 16 key bits used in the function f_{K_1} and the 14 bit state of register R_2. The stream generated by the other 7 registers now have relative entropy $2^{-63.897}$ compared to the random distribution. The computational complexity for finding the 16 bits of the key will be $2^{63.897}2^{30} = 2^{93.897}$. Finding the remaining bits will have lower complexity since the bias will increase as we know more register states while we still have to guess 30 bits each time. Hence the key recovery attack will succeed with $2^{73.527}$ keystream bits and computational complexity $2^{93.897}$. The 80-bit hardware oriented variant of Pomaranch Version 2 has the same structure as the 128-bit variant with the exception that only 6 registers are used. The corresponding distinguishing attack on the 80-bit version will require $2^{44.587}$ bits of keystream and a computational complexity of $2^{58.587}$. The key recovery attack will require

Table 3. Complexities of the attacks proposed in this paper

Attack Complexities			
Type of Attack	Variant	Amount of keystream needed	Computational Complexity
Distinguishing Attack	80 bits	$2^{44.587}$	$2^{58.587}$
	128 bits	$2^{73.527}$	$2^{87.527}$
Key recovery Attack	80 bits	$2^{44.587}$	$2^{64.882}$
	128 bits	$2^{73.527}$	$2^{93.897}$

$2^{44.587}$ keystream bits and has a computational complexity of $2^{64.882}$. Table 3 summarizes all proposed attacks on Pomaranch Version 2 and the corresponding complexities.

6 Simulation Results

When the distribution for the output vectors was theoretically calculated, we assumed that the initial states had the same probability, that all jump control sequences had the same probability and that all jump control sequences JC_i $(2 \leq i \leq 9)$ were independent. To verify these assumptions, the real distribution was simulated for scaled down variants of the cipher. An interesting question here is the amount of samples that is needed in the simulation. Let the simulated distribution of the output from the jump registers be denoted D_0^*. As before, the *theoretical* distribution using the assumptions above is denoted D_0 and the uniform distribution is denoted D_1. The size of the distributions is denoted $|\mathcal{X}|$. We use the following theorem taken from [2].

Theorem 1. *Let $X_1, X_2, \ldots, X_n$ be independent and identically distributed according to D_0. Then*

$$Pr\{D(D_0^* \| D_0) > \epsilon\} \leq 2^{-n\left(\epsilon - |\mathcal{X}|\frac{\log(n+1)}{n}\right)}. \tag{5}$$

If $D(D_0\|D_1) = \mu$, then $\epsilon \leq \mu/2$. To reach the amount of samples where the error probability in (5) is less than or equal to 1, n should satisfy

$$n\left(\mu/2 - |\mathcal{X}|\frac{\log(n+1)}{n}\right) \geq 0 \Rightarrow \frac{n}{\log(n+1)} \geq \frac{2|\mathcal{X}|}{\mu}. \tag{6}$$

It is clear that the amount of samples needed increases exponentially with the vector length and in order to be able to simulate more than one register, we chose to simulate the distribution for the linear relation (4) instead of vectors. (Note that in the hypothesis test used in the cryptanalysis, this is not the case. The amount of samples needed then is still in the order of $1/D(D_0\|D_1)$. Stein's lemma is still applicable.) We found the distribution for 1, 2 and 3 registers.

Table 4. Simulated values for the distributions

Jump Registers Used	$D(D_0 \| D_1)$	
	Theoretical	Simulated
1	$2^{-10.05}$	$2^{-9.82}$
2	$2^{-19.62}$	$2^{-19.36}$
3	$2^{-29.20}$	$2^{-29.27}$

Using a random key, register R_1 was fed with the all zero JC-sequence. The output was taken as the xor of the output of the other registers, i.e. R_2, $R_2 \oplus R_3$ and $R_2 \oplus R_3 \oplus R_4$ respectively. In all cases, the amount of samples used was 2^{39} which, according to (6), should be enough. The simulated values for the relative entropy can be found in Table 4. From the simulated values, we conclude that the assumptions made in the calculation of the theoretical distributions are valid at least up to 3 registers. From this it should be safe to assume that the theoretical distributions are valid also up to 8 registers. Thus, the theoretical distributions can be used in the hypothesis test.

7 Pomaranch Version 3

Following the results given in a preliminary version of this paper, the designers of Pomaranch proposed a new, improved version of Pomaranch, denoted Pomaranch Version 3 [6]. Before we conclude, we briefly look at the design of this version and see how our results apply to it.

The overall design is kept almost the same but there are a few important changes. First, the length of the registers is increased to 18 instead of 14 as in Version 2. Second, there are two different register types used. Register type 1 is used in the odd numbered jump registers and type 2 is used in the even numbered registers. The two types differ in both feedback polynomials and selection of S-cells and F-cells. A third change is that, in the 80-bit variant, the xor of the outputs from registers 1 to 5 is replaced with a nonlinear function G. The output of G is then xored with the output of register 6 to form the keystream.

Considering binary linear relations in the output bits of the registers, even if a highly biased relation is found it is very likely that this relation will be different for the two types of registers. Combining these two relations in order to get a biased sum of keystream bits will result in an extremely small bias. A second option is to find the same biased linear relation in both registers. However, what is highly biased in one register might have a small bias in the other. By instead considering binary vectors as have been done in this paper, the fact that there are two different types of registers is not a problem. The same variables will be used in vectors from both types of registers and the information from the most biased linear relation will be in each vector. Though, for the attack to be successful, it is still required that the vectors have a high bias. We have computed the bias of output vectors of different lengths for 8 registers, 4 of each type, and it is clear that the resulting bias is too low. The approach which proved to be very successful on Pomaranch Version 2 will not be successful on Version 3 and we can conclude that Pomaranch Version 3 is immune to the attack described in this paper.

8 Conclusion

We proposed a new simple algorithm that can be helpful in finding linear approximations of a nonlinear block. The algorithm is suitable for small blocks that can be exhaustively searched. As an example we demonstrate how to find a heavily biased linear approximation for the stream cipher Pomaranch Version 2. The linear approximation gives us information that can be used in a distinguishing or a key recovery attack. In the specific case of Pomaranch, we could get even more information by considering the full vector of output bits instead of just the linear relation provided by our algorithm. In a different setting, the linear relation might be used as a part of the cryptanalysis, possibly involving more building blocks. The linear relation provided by our algorithm might help getting a better theoretical understanding of the design principles of the Pomaranch family of stream ciphers. This further theoretical analysis is left as future work.

References

1. Cid, C., Gilbert, H., Johansson, T.: Cryptanalysis of Pomaranch. IEE Proceedings - Information Security 153(2), 51–53 (2006)
2. Cover, T., Thomas, J.A.: Elements of Information Theory. Wiley series in Telecommunication. Wiley (1991)
3. ECRYPT. eSTREAM: ECRYPT Stream Cipher Project, IST-2002-507932. Available at http://www.ecrypt.eu.org/stream/
4. Jansen, C.J.A, Helleseth, T., Kholosha, A.: Cascade jump controlled sequence generator (CJCSG). eSTREAM, ECRYPT Stream Cipher Project, Report 2005/022
5. Jansen, C.J.A., Helleseth, T., Kholosha, A.: Cascade jump controlled sequence generator and Pomaranch stream cipher (version 2). eSTREAM, ECRYPT Stream Cipher Project, Report 2006/006 (2006), http://www.ecrypt.eu.org/stream
6. Jansen, C.J.A, Helleseth, T., Kholosha, A.: Cascade jump controlled sequence generator and Pomaranch stream cipher (version 3). eSTREAM, ECRYPT Stream Cipher Project (2006), http://www.ecrypt.eu.org/stream
7. Jansen, C.J.A., Helleseth, T., Kholosha, A.: Pomaranch - design and analysis of a family of stream ciphers. In: The State of the Art of Stream Ciphers, Workshop Record, SASC 2006, Leuven, Belgium (February 2006)
8. Khazaei, S.: Cryptanalysis of pomaranch (CJCSG). eSTREAM, ECRYPT Stream Cipher Project, Report 2005/065 (2005), http://www.ecrypt.eu.org/stream
9. Maximov, A., Johansson, T.: Fast computation of large distributions and its cryptographic applications. In: Roy, B. (ed.) ASIACRYPT 2005. LNCS, vol. 3788, pp. 313–332. Springer, Heidelberg (2005)

Multi-pass Fast Correlation Attack on Stream Ciphers*

Bin Zhang and Dengguo Feng

State Key Laboratory of Information Security,
Institute of Software, Chinese Academy of Sciences,
Beijing 100080, P.R. China
martin_zhangbin@yahoo.com.cn

Abstract. Fast correlation attacks are one of the most important attacks against stream ciphers. Previous results on this topic mainly regard the initial state of the involved linear feedback shift register as a whole and only use one sort of parity-checks to decode the corresponding linear code. In this paper we propose a new kind of attack, called multi-pass fast correlation attack, on stream ciphers. This kind of attack can make good use of different kinds of parity-checks without increasing the asymptotic complexity and restore the initial state part-by-part. It has no restriction on the weight of the underlying linear feedback shift register and both theoretical analysis and simulation results show that it is more efficient than all the previously known fast correlation attacks.

Keywords: Stream cipher, Fast correlation attack, Linear feedback shift register (LFSR), Parity-check.

1 Introduction

Stream ciphers are an important class of encryption algorithms. They are widely used in many applications and a deliberately designed stream cipher is often more efficient than a block cipher in software and/or in hardware. However, the security of stream ciphers has not been widely and deeply studied as what have been done for block ciphers. It is helpful to launch new attacks on stream ciphers since the security of a cipher can only be measured by attacks.

So far, several kinds of attacks have been developed against stream ciphers, e.g. (fast) correlation attacks [2,3,4,10,11,14,16,17,19,21,23,24,26], (fast) algebraic attacks [1,6,7]. A popular and powerful approach to attack stream ciphers is to exploit different kinds of correlations between the keystream and some subset of the involved LFSRs. In a known plaintext scenario, these correlations can be used to restore the secret keys, usually the initial states of some LFSRs, from the keystream. The original idea was proposed by Siegenthaler [26] in 1984. Since then, many improvements have been made, producing more and

* Supported by the National Natural Science Foundation of China (Grant No. 90604036, 60603018, 60373047) and the National Grand Fundamental Research 973 program of China (Grant No. 2004CB318004).

© Springer-Verlag Berlin Heidelberg 2007

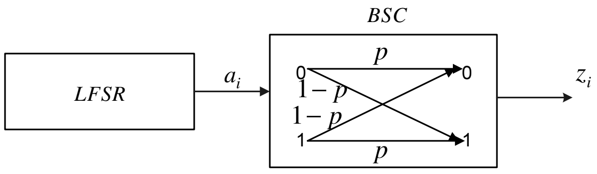


Fig. 1. Model for a fast correlation attack

more powerful fast correlation attacks [2,3,4,14,16,17,21,23,24] on LFSR-based stream ciphers. Not only the nonlinear combination generators but also the nonlinear filter generators, irregularly clocked generators [29], and many variations [5,18] are vulnerable to such a attack under proper conditions. In most articles, the corresponding cryptanalysis problem is viewed as a decoding problem as shown in Figure 1. In such a model, the output keystream $\{z_i\}$ is regarded as a noisy version of the LFSR sequence $\{a_i\}$ and the nonlinearity introduced either by nonlinear boolean function or by other methods is represented as a binary symmetric channel (BSC) with crossover probability $1 - p = 0.5 - \varepsilon$ ($\varepsilon > 0$). In theory, any decoding algorithm can be applied to recover the initial state of the target LFSR, or to restore the original LFSR sequence.

Common fast correlation attacks are classified into iterative algorithms and one-pass algorithms. In an iterative attack, each z_i is associated with a priori probability $p = 0.5 + \varepsilon$, which is updated using parity-checks. These soft-values are used to modify z_i until the BSC noise is removed. In a one-pass attack, the parity-checks are used to directly select some correct bits of $\{a_i\}$ from $\{z_i\}$. If the number of these selected bits is larger than the length of the LFSR, then the initial state can be reconstructed by simple linear algebra. As a summary, the above two approaches all aim at recovering the initial state at one time, though they follow different rules. In fact, all the known fast correlation attacks except that in [3] regard the initial state of the underlying LFSR as a whole, while the attack in [3] is just a preliminary attempt which is less efficient than some attacks targeting the whole initial state [2,4,23,24].

In this paper, we propose new techniques to mount a fast correlation attack on stream ciphers, resulting in a new kind of attack called multi-pass fast correlation attack. Oppositely to the above two approaches, we combine the divide-and-conquer idea with fast correlation attack. More precisely, we first determine *some* bits of the initial state in one pass, then find out some other bits in a second pass conditioned on both the keystream and the known bits. Following this routine, we hope to recover the whole initial state step-by-step. In addition, we propose the concept of *parity-check schedule* to cooperate with the multi-pass attack, i.e. we deploy different kinds of parity-check equations originating from the same idea in different passes. The overall asymptotic complexity of the *parity-check schedule* is not increased by constructing more than one kind of parity-checks. The theoretical analysis shows that our attack can deal with large LFSRs with correlations very close to 0.5. Comparisons with other known fast correlation attacks reveal that our algorithm achieves the best trade-off between keystream length, success probability, and complexities (time, memory and preprocessing). Experimental results confirm the high efficiency of the new attack.

The rest of this paper is organized as follows. A detailed description of the multi-pass fast correlation attack is given in Section 2 together with theoretical analysis. In Section 3, simulation results and comparisons with the best two [24,4] fast correlation attacks are provided. Finally, some conclusions are given in Section 4.

2 Multi-pass Fast Correlation Attack

We first present a outline of the proposed multi-pass fast correlation attack, then a description of the attack is given in detail with theoretical analysis. A potential application of our algorithm beyond the model shown in Figure 1 is also presented at the end of this section.

2.1 Outline of Multi-pass Fast Correlation Attack

The basic idea of the attack presented here is to launch a divide-and-conquer attack to restore the initial state. We divide the initial state $(a_0, a_1, \cdots, a_{L-1})$ of the target LFSR into several parts and plan to recover them one-by-one, as shown in (1).

$$(\underbrace{a_0, \cdots, a_{k^{(1)}-1}}_{k^{(1)}}, \underbrace{a_{k^{(1)}}, \cdots, a_{k^{(1)}+k^{(2)}-1}}_{k^{(2)}}, \underbrace{a_{k^{(1)}+k^{(2)}}, \cdots,}_{\cdots} \cdots, a_{L-1}) \tag{1}$$

We first recover the first part ($k^{(1)}$-bit length) of the initial state, then proceed to determine the second part ($k^{(2)}$-bit length) conditioned on both the first part and the keystream. In this way, we can get an efficient attack without the heavy preprocessing as required by previous attacks for efficient decoding.

Preprocessing Stage. As other fast correlation attacks, we need to pre-compute the parity-checks required by the real time processing stage. In the preprocessing stage of a multi-pass fast correlation attack, we construct various parity-check equations, as shown below.

$$a_{i_1} \oplus a_{i_2} \oplus \cdots \oplus a_{i_{t_1}} = \sum_{i=0}^{k^{(1)}-1} x_i^{(1)} a_i \rightharpoonup \text{first pass}$$

$$a_{j_1} \oplus a_{j_2} \oplus \cdots \oplus a_{j_{t_2}} \oplus \sum_{i=0}^{k^{(1)}-1} x_i'^{(1)} a_i = \sum_{i=k^{(1)}}^{k^{(1)}+k^{(2)}-1} x_i^{(2)} a_i \rightharpoonup \text{second pass}$$

$$\vdots$$

$$a_{l_1} \oplus a_{l_2} \oplus \cdots \oplus a_{l_{t_m}} \oplus \sum_{i=0}^{\delta-1} y_i a_i = \sum_{i=\delta}^{\delta+k^{(m)}-1} x_i^{(m)} a_i, \rightharpoonup mth \text{ pass}$$

where $\delta = \sum_{i=1}^{m-1} k^{(i)}$ $(m \geq 2)$, $t_i \leq t_1$ (for $2 \leq i \leq m$) and $\sum_{i=0}^{\delta-1} y_i a_i$ is a known linear combination of the recovered bits. In the first pass of the processing stage, we use parity-checks of the form $a_{i_1} \oplus a_{i_2} \oplus \cdots \oplus a_{i_{t_1}} \oplus \sum_{i=0}^{k^{(1)}-1} x_i^{(1)} a_i = 0$, while in the second pass, $a_{j_1} \oplus a_{j_2} \oplus \cdots \oplus a_{j_{t_2}} \oplus \sum_{i=0}^{k^{(1)}-1} x_i'^{(1)} a_i \oplus \sum_{i=k^{(1)}}^{k^{(1)}+k^{(2)}-1} x_i^{(2)} a_i = 0$ is applied, and so on.

Definition 1. *We call such a application schedule of different parity-check equations in different passes of the attack a parity-check schedule.*

For $p < 0.55$ in Figure 1, we only use parity-check equations with $t_i \leq 4$, otherwise the folded noise will drop so close to 0.5 that an efficient decoding is impossible. For N keystream bits, if $t_i \leq 3$, we use the traditional square-root time-memory tradeoff to construct the above equations, the time and memory complexity are $O(N^{\lceil (t_i-1)/2 \rceil} \log_2 N)$ and $O(N^{\lfloor (t_i-1)/2 \rfloor})$, respectively. If $t_i = 4$, we adopt the general match-and-sort algorithm in [4] to get these equations (though the usage and meaning of these parity-checks are different in [4]). The time complexity is $O(N^2 \log_2 N)$ and the memory complexity is $O(N)$. To prepare all the parity-checks involved in the *parity-check schedule*, we have to add the complexity of constructing parity-checks for each pass together. Compared with the complexity of preparing one sort of parity-checks, the overall complexity of a *parity-check schedule* is increased only by a small factor m. In most cases of our algorithm, the number of parity-checks involved in pass $i+1$ is much less than that involved in pass i, thus the asymptotic complexity of the *parity-check schedule* is not increased.

Processing Stage. This stage consists of several passes. In each pass, we exhaustively search over the current segment of the initial state and evaluate the parity-checks to record those possible values passing the test as candidates of the corresponding bits. If the current pass is not the last one, we should pass these candidates to the next pass. The bits remain unknown after all the passes can be recovered by an exhaustive search at a small scale and a correlation check procedure.

2.2 Attack Details and Theoretical Analysis

Let $f(x)$ of degree L be the feedback polynomial of the LFSR modelled in Figure 1 and $P(a_i = z_i) = p = \frac{1}{2} + \varepsilon$, $\varepsilon > 0$. From the initial state $(a_0, a_1, \cdots, a_{L-1})$, we have $(a_0, \cdots, a_{L-1}, a_L, \cdots, a_{N-1}) = (a_0, a_1, \cdots, a_{L-1}) \cdot G$, where N is the length of sequence $\{a_i\}$ under consideration and G is a $L \times N$ matrix over $GF(2)$:

$$G = \begin{pmatrix} g_0^1 & g_1^1 & \cdots & g_{N-1}^1 \\ g_0^2 & g_1^2 & \cdots & g_{N-1}^2 \\ \vdots & \vdots & \cdots & \vdots \\ g_0^L & g_1^L & \cdots & g_{N-1}^L \end{pmatrix}.$$

Thus each a_i is a linear combination of $(a_0, a_1, \cdots, a_{L-1})$. We regard the column vector $g_i = (g_i^1, g_i^2, \cdots, g_i^L)^T$ as a random vector, then there are $\Omega^{(1)} =$

$\binom{N}{t_1}/2^{L-k^{(1)}}$ t_1-tuple column vectors $(g_{i_1}, g_{i_2}, \ldots, g_{i_{t_1}})$ satisfying $g_{i_1} \oplus g_{i_2} \oplus \cdots \oplus g_{i_{t_1}} = (x_0^{(1)}, x_1^{(1)}, \cdots, x_{k^{(1)}-1}^{(1)}, 0, \cdots, 0)^T$, where $\oplus$ is bitwise xor and $(x_0^{(1)}, x_1^{(1)}, \cdots, x_{k^{(1)}-1}^{(1)})$ is a $k^{(1)}$-dimension vector with $k^{(1)} < L$. For each such t_1-tuple, we have $a_{i_1} \oplus a_{i_2} \oplus \cdots \oplus a_{i_{t_1}} = \sum_{j=0}^{k^{(1)}-1} x_j^{(1)} a_j$. We rewrite it as:

$$z_{i_1} \oplus z_{i_2} \oplus \cdots \oplus z_{i_{t_1}} = \sum_{j=0}^{k^{(1)}-1} x_j^{(1)} a_j \oplus \sum_{j=1}^{t_1} e_{i_j}, \tag{2}$$

where $e_j = a_j \oplus z_j (j = i_1, \cdots, i_{t_1})$ is the corresponding random noise variable with distribution $P(e_j = 0) = \frac{1}{2} + \varepsilon$. The basic distinguisher in pass one is that if we exhaustively search all the possible values of $(a_0, a_1, \cdots, a_{k^{(1)}-1})$, then from (2), we have

$$z_{i_1} \oplus z_{i_2} \oplus \cdots \oplus z_{i_{t_1}} \oplus \sum_{j=0}^{k^{(1)}-1} x_j^{(1)} a'_j = \sum_{j=0}^{k^{(1)}-1} x_j^{(1)} \cdot (a_j \oplus a'_j) \oplus \sum_{j=1}^{t_1} e_{i_j}, \tag{3}$$

where $(a'_0, a'_1, \cdots, a'_{k^{(1)}-1})$ is the guessed value. Let $\Delta(i_1, \cdots, i_{t_1}) = \sum_{j=0}^{k^{(1)}-1} x_j^{(1)} \cdot (a_j \oplus a'_j) \oplus \sum_{j=1}^{t_1} e_{i_j}$, it is obvious that if $(a_0, a_1, \cdots, a_{k^{(1)}-1})$ is correctly guessed, we have $\Delta(i_1, \cdots, i_{t_1}) = \sum_{j=1}^{t_1} e_{i_j}$. All the e_j's are independent random variables, thus from the piling-up lemma [20],

$$q^{(1)} = P(\sum_{j=1}^{t_1} e_{i_j} = 0) = \frac{1}{2} + 2^{t_1-1}\varepsilon^{t_1} \tag{4}$$

holds. If $(a_0, a_1, \cdots, a_{k^{(1)}-1})$ is wrongly guessed, $\Delta(i_1, \cdots, i_{t_1}) = \sum_{j: a_j \oplus a'_j = 1} x_j^{(1)} \oplus \sum_{j=1}^{t_1} e_{i_j}$. Since $x_j^{(1)}$ is the xor of t_1 independent uniform distributed variables, we have $P(x_j = 0) = P(x_j = 1) = 0.5$. Hence, when $(a_0, a_1, \cdots, a_{k^{(1)}-1})$ is wrongly guessed, $\Delta(i_1, \cdots, i_{t_1})$ have the distribution $P(\Delta = 0) = 0.5$, which is quite different from that in (4). For $\Omega^{(1)}$ such equations as (3) built from all the t_1-tuples $(g_{i_1}, g_{i_2}, \ldots, g_{i_{t_1}})$, if $(a_0, a_1, \cdots, a_{k^{(1)}-1})$ is correctly guessed, $\sum_{i=1}^{\Omega^{(1)}} (\Delta(i_1, \cdots, i_{t_1}) \oplus 1)$ should follow the binomial distribution $(\Omega^{(1)}, q^{(1)})$. Otherwise, this sum should have the binomial distribution $(\Omega^{(1)}, \frac{1}{2})$, which can be used to filter out the wrong guesses of $(a_0, a_1, \cdots, a_{k^{(1)}-1})$.

To fulfill the above observations, we need to substitute z_i into the parity-check equations and evaluate them to count the number of the vanishing Δs. The straightforward method has a time complexity of $O(2^{k^{(1)}} k^{(1)} \Omega^{(1)})$, which causes an inefficient attack. Instead we proceed as follows. First regroup $\Omega^{(1)}$ parity-check equations according to the pattern of $(x_0^{(1)}, x_1^{(1)}, \cdots, x_{k^{(1)}-1}^{(1)})$ and define

$$h(x_0^{(1)}, x_1^{(1)}, \cdots, x_{k^{(1)}-1}^{(1)}) = \sum_{(x_0^{(1)}, x_1^{(1)}, \cdots, x_{k^{(1)}-1}^{(1)})} (-1)^{z_{i_1} \oplus z_{i_2} \oplus \cdots \oplus z_{i_{t_1}}}$$

for all the $(x_0^{(1)}, x_1^{(1)}, \cdots, x_{k^{(1)}-1}^{(1)})$ patterns appearing in the $\Omega^{(1)}$ parity-check equations. If a pattern of $(x_0^{(1)}, x_1^{(1)}, \cdots, x_{k^{(1)}-1}^{(1)})$ does not appear in all the $\Omega^{(1)}$ equations, we define $h(x_0^{(1)}, x_1^{(1)}, \cdots, x_{k^{(1)}-1}^{(1)}) = 0$ at that point. Thus we have a well-defined function $h : GF(2)^k \to \mathbb{R}$. Consider the Walsh transform of $h(x_0^{(1)}, x_1^{(1)}, \cdots, x_{k^{(1)}-1}^{(1)})$, i.e.

$$\begin{aligned} H(\omega) &= \sum_{x \in GF(2)^{k^{(1)}}} h(x)(-1)^{\omega \cdot x} \qquad (5) \\ &= \sum_{\Omega^{(1)}} (-1)^{z_{i_1} \oplus z_{i_2} \oplus \cdots \oplus z_{i_{t_1}} \oplus \sum_{j=0}^{k^{(1)}-1} \omega_j x_j^{(1)}} = \Omega_0^{(1)} - \Omega_1^{(1)}, \end{aligned}$$

where $\omega = (\omega_0, \omega_1, \cdots, \omega_{k^{(1)}-1})$, $x = (x_0^{(1)}, x_1^{(1)}, \cdots, x_{k^{(1)}-1}^{(1)})$, $\Omega_0^{(1)}$ and $\Omega_1^{(1)}$ are the number of 0 and 1, respectively. Note that if $\omega = (a_0, a_1, \cdots, a_{k^{(1)}-1})$, we have

$$\sum_{j=1}^{\Omega^{(1)}} (\Delta(i_1, \cdots, i_{t_1}) \oplus 1) = \frac{H(\omega) + \Omega^{(1)}}{2}. \qquad (6)$$

Hence, for each guessed value $(a'_0, a'_1, \cdots, a'_{k^{(1)}-1})$, we only need to compute one value of h's Walsh transform to get the number of vanishing Δs. There are $2^{k^{(1)}}$ such guesses, which means that we need to compute all the $2^{k^{(1)}}$ values of h's Walsh transform. Thanks to the fast Walsh transform (FWT) [13,28], this can be done efficiently (at one time) in $O(2^{k^{(1)}} k^{(1)})$ time with $O(2^{k^{(1)}})$ memory. The preparation of h takes $O(\Omega^{(1)})$ time, thus the total time complexity of this step is $O(\Omega^{(1)} + 2^{k^{(1)}} k^{(1)})$. Compared with the time complexity of the former method, $O(2^{k^{(1)}} k^{(1)} \Omega^{(1)})$, this is a large improvement.

Instead of taking the guess with the largest $H(\omega)$, we put a threshold value $T^{(1)}$ of $H(\omega)$ when we make a decision, i.e. we accept all the guesses that satisfy $(H(\omega) + \Omega^{(1)})/2 \geq T^{(1)}$ at the first pass. The probability that the right guess could pass this test is (here we use the normal distribution approximation):

$$P_1^{(1)} = \sum_{i=T^{(1)}}^{\Omega^{(1)}} \binom{\Omega^{(1)}}{i} (q^{(1)})^i (1 - q^{(1)})^{\Omega^{(1)} - i} \to \int_{T^{(1)}}^{\Omega^{(1)}+0.5} \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}} dx, \qquad (7)$$

and the probability that a wrong guess would pass the test is

$$P_2^{(1)} = \sum_{i=T^{(1)}}^{\Omega^{(1)}} \binom{\Omega^{(1)}}{i} (\frac{1}{2})^{\Omega^{(1)}} \to \int_{T^{(1)}}^{\Omega^{(1)}+0.5} \frac{1}{\sqrt{2\pi}\sigma'} e^{-\frac{(x-\mu')^2}{2\sigma'^2}} dx, \qquad (8)$$

where $T^{(1)}$ is the threshold to be determined according to various attack requirements specified below. In (7), $\mu = \Omega^{(1)} \cdot q^{(1)}$ and $\sigma = \sqrt{\Omega^{(1)} q^{(1)} (1 - q^{(1)})}$ are the mean and the standard deviation, respectively, while in (8), $\mu' = \Omega^{(1)} \cdot \frac{1}{2}$ and $\sigma' = \frac{1}{2}\sqrt{\Omega^{(1)}}$ are the mean and the standard deviation in the random wrong guess case. If the guess $(a'_0, a'_1, \cdots, a'_{k^{(1)}-1})$ leads to $H(a'_0, a'_1, \cdots, a'_{k^{(1)}-1}) \geq T^{(1)}$, we accept it into the next pass, otherwise we filter it out right now. It is naturally expected that $P_1^{(1)}$ is very close to 1 so that the right guess could pass the test with high probability and $P_2^{(1)}$ is very small so that all the wrong guesses could be frustrated, or at least reduced to a large extent.

The introduction of the threshold $T^{(1)}$ provides large flexibility when we actually construct a fast correlation attack. Here we list four useful cases.

Case 1. $P_1^{(1)} > 0.99$ and $P_2^{(1)} < 2^{-k^{(1)}}$. The right guess will almost certainly be accepted and none of the wrong guesses could pass the test, i.e. we already restored $k^{(1)}$ bits of the initial state.
Case 2. $P_1^{(1)} > 0.99$ and $P_2^{(1)} \approx 2^{-k_1^{(1)}}$ with $k_1^{(1)} < k^{(1)}$. The right guess will pass the test with high probability together with some wrong guesses, i.e. we have a small list of candidates of the $k^{(1)}$ considered bits.
Case 3. $P_1^{(1)} < 0.99$ and $P_2^{(1)} < 2^{-k^{(1)}}$. None of the wrong guesses could pass the test, while the right guess will go through the test with some probability, i.e. we will get no candidate in some tests. In this case, we have to repeat the whole attack several times to get a high success rate.
Case 4. $P_1^{(1)} < 0.99$ and $P_2^{(1)} \approx 2^{-k_1^{(1)}}$ with $k_1^{(1)} < k^{(1)}$. We will always get some candidates in this case, though some may be wrong. In this case, we will finally get a list of candidates of the initial state, which are to be checked by correlation match.

These four cases can be applied according to different attack conditions and requirements. If Case 1 can work, then we just choose $T^{(1)}$ such that we have already recovered $k^{(1)}$ bits. If $P_1^{(1)} > 0.99$ and $P_2^{(1)} < 2^{-k^{(1)}}$ cannot be satisfied at the same time, then other cases are also helpful, e.g. if Case 2 can work, then we have already reduced the possible values of the $k^{(1)}$ bits to some extent, while we have large flexibility to construct parity-check equations used in the next pass. In this case, we can reduce the keystream length used in the next pass without a compromise of the magnitude of parity-check equations.

Once we finished the first pass, we enter the second pass to determine the next $k^{(2)}$ bits of the initial state conditioned on both the keystream and the recovered information. From the *parity-check schedule* specified in Section 2.1, the distinguisher becomes

$$\Delta(j_1, \cdots, j_{i_{t_2}}) = z_{j_1} \oplus z_{j_2} \oplus \cdots \oplus z_{j_{t_2}} \oplus \sum_{i=0}^{k^{(1)}-1} x_i'^{(1)} a_i \oplus \sum_{j=k^{(1)}}^{k^{(1)}+k^{(2)}-1} x_j^{(2)} a'_j \quad (9)$$
$$= \sum_{j=k^{(1)}}^{k^{(1)}+k^{(2)}-1} x_j^{(2)} (a_j \oplus a'_j) \oplus \sum_{j=1}^{t_2} e_{i_j},$$

where $\sum_{i=0}^{k^{(1)}-1} x_i'^{(1)} a_i$ is a known parameter and $(a'_{k^{(1)}}, \cdots, a'_{k^{(1)}+k^{(2)}-1})$ is the guessed value of the next $k^{(2)}$ bits. There are $\Omega^{(2)} = \binom{N_2}{t_2}/2^{L-k^{(1)}-k^{(2)}}$ parity-check equations used in the second pass, where $N_2 \leq N$ is the keystream length involved in the second pass. The distinguisher works in the same way as that in the first pass, i.e. we exhaustively substitute each $(a'_{k^{(1)}}, \cdots, a'_{k^{(1)}+k^{(2)}-1})$ into $\Omega^{(2)}$ parity-check equations and evaluate them to count the number of vanishing Δs. If the guess is correct, $\sum_{i=1}^{\Omega^{(2)}} (\Delta(j_1, \cdots, j_{t_2}) \oplus 1)$ should follow the binomial distribution $(\Omega^{(2)}, q^{(2)} = 0.5 + 2^{t_2-1}\varepsilon^{t_2})$, otherwise it has binomial distribution $(\Omega^{(2)}, 0.5)$. In this way, we can filter out the wrong guesses. As in the first pass, we define

$$h^{(2)}(x^{(2)}_{k^{(1)}}, \cdots, x^{(2)}_{k^{(1)}+k^{(2)}-1}) = \sum_{(x^{(2)}_{k^{(1)}}, \cdots, x^{(2)}_{k^{(1)}+k^{(2)}-1})} (-1)^{z_{j_1} \oplus \cdots \oplus z_{j_{t_2}} \oplus \sum_{i=0}^{k^{(1)}-1} x_i'^{(1)} a_i}$$

according to the pattern of $(x^{(2)}_{k^{(1)}}, \cdots, x^{(2)}_{k^{(1)}+k^{(2)}-1})$. If a pattern of $(x^{(2)}_{k^{(1)}}, \cdots, x^{(2)}_{k^{(1)}+k^{(2)}-1})$ does not appear in the $\Omega^{(2)}$ parity-check equations, let $h^{(2)} = 0$ at that point. As before, we use the fast Walsh transform of $h^{(2)}$ to compute all the $\sum_{i=1}^{\Omega^{(2)}} (\Delta(j_1, \cdots, j_{t_2}) \oplus 1) = (H^{(2)}(\omega) + \Omega^{(2)})/2$ at one time, where $H^{(2)}$ is the Walsh transform of $h^{(2)}$ and $\omega = (a'_{k^{(1)}}, \cdots, a'_{k^{(1)}+k^{(2)}-1})$.

The time and memory complexity of the second pass are $O(2^{k^{(2)}} k^{(2)} + \Omega^{(2)})$ and $O(2^{k^{(2)}})$, respectively. As in pass one, we put another threshold value $T^{(2)}$ to make a decision, i.e. if one guessed value $a'_{k^{(1)}}, \cdots, a'_{k^{(1)}+k^{(2)}-1}$ leads to $(H^{(2)}(a'_{k^{(1)}}, \cdots, a'_{k^{(1)}+k^{(2)}-1}) + \Omega^{(2)})/2 \geq T^{(2)}$, we accept it into the third pass. Otherwise discard it. In order to get an efficient attack, in the second pass as well as in the consequent passes, we only let Case 1 ($P_1^{(i)} > 0.99$ and $P_2^{(i)} < 2^{-k^{(i)}}$ for $2 \leq i \leq m$) occur, i.e. we ignore all the wrong guesses in the passes from pass two.

So far, we have recovered $k^{(1)} + k^{(2)}$ bits of the initial state. If the number of the unknown bits is still so large that the complexity of exhaustively searching over these unknown bits and the correlation check step dominates/doubles the overall complexity, we should arrange more passes to reduce the complexity. In our experiments, we usually have three or four passes to get a satisfactory low-complexity attack.

The entire description of the precessing stage of a multi-pass fast correlation attack is given below.

- **Parameter:** m, $t_1, \cdots, t_m$, $k^{(1)}, \cdots, k^{(m)}$, p
- **Input:** keystream $\{z_i\}_{i=0}^{N-1}$, feedback polynomial $f(x)$
- **Processing:**
  1. **for** $i = 1, \cdots, m$ **do**
     Define $h^{(i)}(x^{(i)}_{k^{(i-1)}}, \cdots, x^{(i)}_{k^{(i-1)}+k^{(i)}-1}) =$
     $\sum_{(x^{(i)}_{k^{(i-1)}}, \cdots, x^{(i)}_{k^{(i-1)}+k^{(i)}-1})} (-1)^{z_{j_1} \oplus \cdots \oplus z_{j_{t_i}} \oplus \sum_{i=0}^{k^{(i-1)}-1} y'_i a_i}$, where $h^{(1)} = h$ and for $i = 1$, $y'_i = 0$ for $0 \leq i \leq k^{(0)}$

```
    apply FWT to compute $\sum_{j=1}^{\Omega^{(i)}}(\Delta\oplus 1)$ for all the $2^{k^{(i)}}$ possible guesses of
    the current $k^{(i)}$-bit division, where $\Omega^{(i)}=\binom{N_i}{t_i}/2^{L-\sum_{j=1}^{i}k^{(j)}}$ and $N_i \leq N$
    is the keystream length used in pass $i$
    **if** $\sum_{j=1}^{\Omega^{(i)}}(\Delta\oplus 1)\geq T^{(i)}$ **then**
    accept the corresponding guess into pass $i+1$
    **else** discard it
    **end if**
    **if** no guess is selected **then**
    break the loop and restart the algorithm
    **end if**
    **end for**
 2. **if** $\sum_{i=1}^{m}k^{(i)}<L$ **then**
    exhaustively search over the $L-\sum_{i=1}^{m}k^{(i)}$ bits and check each value by
    running the LFSR and computing the correlation between $\{z_i\}$ and the
    generated sequence
    **end if**
```

– **Output:** the initial state $(a_0, a_1, \cdots, a_{L-1})$ or a small list of candidates

The time complexity of the processing stage consists of the complexity of each pass and the correlation check procedure if some bits were left unrecovered after pass m. If all the candidates cannot pass the correlation test, we should run the whole algorithm several times (usually three or four times in practice) to get the correct initial state. The memory complexity of the processing stage is $O(\max_{1\leq i\leq m}\max(\Omega^{(i)}\ ,2^{k^{(i)}}))$, while the time complexity (with success rate higher than 99%) varies according to the four different cases listed before.

Case 1. The time complexity is $O(\sum_{i=1}^{m}(\Omega^{(i)}+2^{k^{(i)}}k^{(i)})+2^{L-\sum_{i=1}^{m}k^{(i)}}(\frac{1}{\varepsilon^2}))$.

Case 2. The time complexity is $O(\Omega^{(1)}+2^{k^{(1)}}k^{(1)}+2^{k^{(1)}-k_1^{(1)}}\cdot(\sum_{i=2}^{m}(\Omega^{(i)}+2^{k^{(i)}}k^{(i)})+2^{L-\sum_{i=1}^{m}k^{(i)}}(\frac{1}{\varepsilon^2})))$.

Case 3. The time complexity is $O(\alpha(\sum_{i=1}^{m}(\Omega^{(i)}+2^{k^{(i)}}k^{(i)})+2^{L-\sum_{i=1}^{m}k^{(i)}}(\frac{1}{\varepsilon^2})))$, where α is the smallest integer satisfying $(1-P_1^{(1)})^{\alpha}<0.01$.

Case 4. The time complexity is $O(\alpha(\Omega^{(1)}+2^{k^{(1)}}k^{(1)}+2^{k^{(1)}-k_1^{(1)}}\cdot(\sum_{i=2}^{m}(\Omega^{(i)}+2^{k^{(i)}}k^{(i)})+2^{L-\sum_{i=1}^{m}k^{(i)}}(\frac{1}{\varepsilon^2}))))$, where α is the smallest integer satisfying $(1-P_1^{(1)})^{\alpha}<0.01$.

Remarks. It is interesting to see that our algorithm has the same time complexity, $O(\Omega^{(i)}+2^{k^{(i)}}k^{(i)})$, in each pass as that of decoding a linear $[\Omega^{(i)}, k^{(i)}]$ code by the method proposed in [19]. However, It is much worth noticing that our approach is quite different from that in [19]. In [19], this decoding was done by a minimum distance rule, whereas we follow the distinguishers built according to the parity-check schedule and the basic observations of our attack have nothing to do with the codeword distance. As can be seen above, our method is more advanced which offers large flexibility. This can be seen from two aspects. First, we can achieve arbitrary success probability from 0 to 1, which is a useful property to reduce the required keystream length to a realistic range. Second, we can freely apply the four cases listed above to deal with various attack conditions. For example, if Case 2 occurs, we still can use the parity-check equations in the second pass, though we do not know the exact value of the first $k^{(1)}$ bits (instead we only reduced the possible values of these bits in the past pass).

2.3 A Potential Application Beyond Figure 1

All the above results are based on the model shown in Figure 1. As can be seen below, this is not always necessary. We can loose the model as follows, i.e. we have a sequence of variables $s_0, s_1, \cdots, s_{n-1} \in GF(2)$ such that the following equations hold.

$$\begin{cases} s_0 &= u_0^{(0)} a_0 \oplus u_1^{(0)} a_1 \oplus \cdots \oplus u_{m-1}^{(0)} a_{m-1} \oplus e_0 \\ s_1 &= u_0^{(1)} a_0 \oplus u_1^{(1)} a_1 \oplus \cdots \oplus u_{m-1}^{(1)} a_{m-1} \oplus e_1 \\ & \quad \cdots\cdots \qquad\qquad \cdots\cdots \\ s_{n-1} &= u_0^{(n-1)} a_0 \oplus u_1^{(n-1)} a_1 \oplus \cdots \oplus u_{m-1}^{(n-1)} a_{m-1} \oplus e_{n-1} \end{cases},$$

where the coefficients $u_j^{(i)}$ $(0 \leq i \leq n-1, 0 \leq j \leq m-1)$ are known parameters and e_i $(0 \leq i \leq n-1)$ are independent random variables with distribution $P(e_i = 0) = 0.5 + \varepsilon_i$ $(\varepsilon_i > 0)$. We do not care how we derive the above linear system, our aim is to recover the m variables $a_0, a_1, \cdots, a_{m-1}$ from the above linear system. Note that the model shown in Figure 1 is just a special case of the above problem, where the coefficients are derived from the generator matrix G.

To solve the above problem, we can follow a similar multi-pass routine. More precisely, we first pre-compute a system of parity-checks from the coefficient matrix $(u_j^{(i)})_{n \times m}$ as that in Section 2.1. Once we get the actual values of $s_0, s_1, \cdots, s_{n-1}$, we proceed as a multi-pass fast correlation attack to determine the m variables $a_0, a_1, \cdots, a_{m-1}$ part-by-part. It is worth noting that the above linear system exists for a variety of stream ciphers, e.g. summation generator [9], one-level bluetooth E0 [12]. In these cases, each a_i $(0 \leq i \leq m-1)$ corresponds to an unknown bit of the initial states of the underlying LFSRs and each s_i $(0 \leq i \leq n-1)$ is some linear combination of the keystream bits. It is our future work to investigate the possibility of applying our method to these stream ciphers.

3 Experimental Results and Comparisons

To check the actual performance of our algorithm, we made experiments on a Pentium 4 processor. To facilitate the comparisons with the best known results

Table 1. Comparisons with the two best attacks previously known with success rate higher than 99%. The LFSR polynomial is $1 + x + x^3 + x^5 + x^9 + x^{11} + x^{12} + x^{17} + x^{19} + x^{21} + x^{25} + x^{27} + x^{29} + x^{32} + x^{33} + x^{38} + x^{40}$.

Attack	Noise	Keystream	Time	Memory	Pre-computation
[24]	0.469	$2^{18.61}$	$O(2^{42})$	$O(2^{16})$	$O(2^{29})$
[24]	0.490	$2^{18.46}$	$O(2^{55})$	$O(2^{25})$	$O(2^{29})$
[4]	0.469	$2^{16.29}$	$O(2^{31})$	$O(2^{25})$	$O(2^{37})$
[4]	0.490	$2^{16.29}$	$O(2^{40})$	$O(2^{35})$	$O(2^{37})$
Ours	0.469	2^{22}	$\mathbf{O(2^{24})}$	$O(2^{23})$	$\mathbf{O(2^{27})}$
Ours	0.490	2^{24}	$\mathbf{O(2^{29})}$	$O(2^{29})$	$\mathbf{O(2^{29})}$

Table 2. Comparisons with the two best attacks previously known with success rate close to 1 against a random chosen LFSR of length 89

Attack	Noise	Keystream	Time	Memory	Pre-computation
[24]	0.469	2^{38}	$O(2^{52})$	$O(2^{18})$	$O(2^{45})$
[4]	0.469	2^{28}	$O(2^{44})$	$O(2^{25})$	$O(2^{61})$
Ours	0.469	2^{32}	$\mathbf{O(2^{32})}$	$O(2^{31})$	$\mathbf{O(2^{37})}$

in other articles, we use the standard feedback polynomial of the LFSR involved in many articles.

Due to the fact that some important attack parameters are not specified in [24], the memory/pre-computation complexities of the attack in [24] listed in Table 1 and 2 are only roughly derived only according to the formulae in [24]. Table 1 shows that multi-pass fast correlation attack provides a better tradeoff between keystream length, success probability and attack complexity. We implemented the attack on the standard 40-bit LFSR with noise 0.469 in C language on a Pentium 4 processor. The parameters are chosen as follows: $m = 2$, $t_1 = t_2 = 2$, $N_1 = N = 2^{22}, N_2 = 2^{15}$ and $k^{(1)} = 20, k^{(2)} = 13$. 7 bits were left to be restored by an exhaustive search of complexity $O(2^{17})$. The preprocessing stage (it has time complexity $O(2^{27})$ and memory complexity $O(2^{22})$) lasts for a few hours, while the decoding stage takes several minutes to output the result. Compared with the preprocessing time complexity $O(2^{37})$ in [4], which takes a few days to finish the pre-computation, the gain is obvious. Besides, the memory usage in the decoding part is only $O(2^{23})$ and the success probability is higher than 99%, which make our attack more realistic. Table 1 also lists the theoretical result for the same LFSR with noise 0.49, the corresponding attack parameters are $m = 2$, $t_1 = t_2 = 2$, $N_1 = N = 2^{24}$, $N_2 = 2^{18}$ and $k^{(1)} = 22, k^{(2)} = 11$. We leave 7 bits to be recovered by an exhaustive search of complexity $O(2^{20})$. The time and memory complexity for the preprocessing stage are $O(2^{29})$ and $O(2^{24})$, respectively. Compared with the preprocessing complexity $O(2^{37})$ in [4], the gain is rather large. The memory complexity of the processing stage is $O(2^{29})$ and the success rate is higher than 99%.

Table 3. Theoretical result of a multi-pass fast correlation attack with success rate close to 1 against a random chosen LFSR of length 103

Attack	Noise	Keystream	Time	Memory	Pre-computation
Ours	0.469	2^{36}	$\mathbf{O(2^{34})}$	$\mathrm{O}(2^{31})$	$\mathbf{O(2^{41})}$

Table 4. Theoretical result of a multi-pass fast correlation attack with success rate close to 1 against a random chosen LFSR of length 61 with noise 0.499

Attack	Noise	Keystream	Time	Memory	Pre-computation
Ours	**0.499**	2^{31}	$\mathbf{O(2^{34})}$	$O(2^{29})$	$\mathbf{O(2^{36})}$

Table 2 yields the theoretical result for a 89-bit LFSR with noise 0.469. The time complexity of our attack is very impressive and multi-pass fast correlation attack makes good use of the keystream. The attack parameters are $m = 3$, $t_1 = 3, t_2 = t_3 = 2$, $N_1 = N = 2^{32}, N_2 = N_1, N_3 = 2^{20}$ and $k^{(1)} = k^{(2)} = 26, k^{(3)} = 24$. We need another 13-bit exhaustive search of complexity $O(2^{23})$ to recover the left 13 bits. The time and memory complexity for the preprocessing stage are $O(2^{37})$ and $O(2^{32})$, respectively. Compared with the preprocessing complexity $O(2^{61})$ in [4], our attack is more realistic. The memory usage of the decoding stage is $O(2^{31})$ and the success probability is higher than 99%.

Table 3 gives the theoretical result of a multi-pass fast correlation attack against an arbitrary weight LFSR of length 103. The parameters for our algorithm are $m = 4$, $t_1 = t_2 = 3, t_3 = t_4 = 2$, $N_1 = N = 2^{36}, N_2 = N_3 = 2^{29}, N_4 = 2^{17}$ and $k^{(1)} = 29, k^{(2)} = 21, k^{(3)} = 26, k^{(4)} = 25$. We leave 2 bits to be recovered by an exhaustive search of complexity $O(2^{12})$. The time and memory complexity for the preprocessing stage are $O(2^{41})$ and $O(2^{36})$, respectively. The memory complexity of the decoding stage is $O(2^{31})$ and the attack has a success rate higher than 99%.

Table 4 gives the theoretical result of a multi-pass fast correlation attack against an arbitrary weight LFSR of length 61 with noise 0.499. The parameters are $m = 3$, $t_1 = t_2 = t_3 = 2$, $N_1 = N = 2^{31}, N_2 = 2^{25}, N_3 = 2^{19}$ and $k^{(1)} = 29, k^{(2)} = 12, k^{(3)} = 11$. We leave 9 bits to be recovered by an exhaustive search of complexity $O(2^{29})$. The time and memory complexity for the preprocessing stage are $O(2^{36})$ and $O(2^{31})$, respectively. The memory complexity of the decoding stage is $O(2^{29})$ and the success rate is higher than 99%. This example shows that our algorithm can deal with moderately large LFSRs with smaller correlations than all the previously reported results. This extends the application scope of fast correlation attacks.

From the above examples, we can see that multi-pass fast correlation attack has at least the following advantages over the past relevant attacks:

– significantly smaller processing time complexity with similar memory complexity, i.e. it has some *uniform* property in the complexity aspect.

– significantly smaller preprocessing time complexity without a compromise of the real attack complexity.
– theoretical analyzibility and flexibility.

These features guarantee that multi-pass fast correlation attack can provide a better tradeoff between keystream length, success probability and attack complexities. The impressively low complexity of this kind of attack mainly comes from the divide-and-conquer idea and the valid application of the keystream of realistic length. Besides, the attack proposed here also provides a general framework for mounting a fast correlation attack on LFSR-based stream ciphers. If we can determine the initial state segments by some method faster than that presented here, then we can get an improved multi-pass attack.

4 Conclusions

In this paper, we presented new approaches to launch a fast correlation attack on stream ciphers. The new attack enables us to analyze larger LFSRs with smaller correlations and has better trade-off between keystream length, success probability and attack complexity. Besides, we show that the new method has some potential application of efficiently solving linear systems with noisy output derived from certain stream ciphers.

References

1. Armknecht, F., Krause, M.: Algebraic Attacks on Combiners with Memory. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 162–175. Springer, Heidelberg (2003)
2. Canteaut, A., Trabbia, M.: Improved Fast Correlation Attacks using parity-check equations of weight 4 and 5. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 573–588. Springer, Heidelberg (2000)
3. Chepyzhov, V.V., Johansson, T., Smeets, B.: A Simple Algorithm for Fast Correlation Attacks on Stream Ciphers. In: Schneier, B. (ed.) FSE 2000. LNCS, vol. 1978, pp. 181–195. Springer, Heidelberg (2001)
4. Chose, P., Joux, A., Mitton, M.: Fast Correlation Attacks: An Algorithmic Point of View. In: Knudsen, L.R. (ed.) EUROCRYPT 2002. LNCS, vol. 2332, pp. 209–221. Springer, Heidelberg (2002)
5. Clark, A., Dawson, E., Fuller, J., Golić, J., et al.: The LILI-128 Keystream Generator. In: Stinson, D.R., Tavares, S. (eds.) SAC 2000. LNCS, vol. 2012, pp. 22–39. Springer, Heidelberg (2001)
6. Courtois, N.T., Meier, W.: Algebraic Attacks on Stream Ciphers with Linear Feedback. In: Biham, E. (ed.) Advances in Cryptology – EUROCRPYT 2003. LNCS, vol. 2656, pp. 345–359. Springer, Heidelberg (2003)
7. Courtois, N.T.: Fast Algebraic Attacks on Stream Ciphers with Linear Feedback. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 176–194. Springer, Heidelberg (2003)

8. Golić, J.D.: Computation of low-weight parity-check polynomials. Electronic Letters 32(21), 1981–1982 (1996)
9. Golić, J.D., Salmasizadeh, M. (ed.): Dawson, Fast correlation attacks on the summation generator, Journal of Cryptology, Springer-Verlag, vol. 13, pp. 245–262 (2000)
10. Golić, J.D.: Iterative optimum symbol-by-symbol decoding and fast correlation attack. IEEE Trans. Inform. Theory 47, 3040–3049 (2001)
11. Golić, J.D., Hawkes, P.: Vetorial appraoch to fast correlation attacks. Designs, Codes and Cryptography 35, 5–19 (2005)
12. Golić, J.D.: Linear cryptanalysis of bluetooth stream cipher. In: Knudsen, L.R. (ed.) EUROCRYPT 2002. LNCS, vol. 2332, pp. 238-255, pp. 51–74. Springer, Heidelberg (2002)
13. Karpovsky, M.: Finite Orthogonal Series in the Design of Diginal Devices. John Wiley and Sons, New York (1976)
14. Johansson, T., Jösson, F.: Fast Correlation Attacks based on turbo code techniques. In: Wiener, M.J. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 181–197. Springer, Heidelberg (1999)
15. Johansson, T.: Reduced complexity correlation attacks on two clock-controlled generators. In: Ohta, K., Pei, D. (eds.) ASIACRYPT 1998. LNCS, vol. 1514, pp. 342–357. Springer, Heidelberg (1998)
16. Johansson, T., Jösson, F.: Improved Fast Correlation Attacks on Stream Ciphers via Convolutional Codes. In: Stern, J. (ed.) EUROCRYPT 1999. LNCS, vol. 1592, pp. 347–362. Springer, Heidelberg (1999)
17. Johansson, T., Jösson, F.: Fast Correlation Attacks through reconstruction of linear polynomals. In: Bellare, M. (ed.) CRYPTO 2000. LNCS, vol. 1880, pp. 300–315. Springer, Heidelberg (2000)
18. Johansson, T., Jösson, F.: A Fast Correlation Attack on LILI-128. Information Processing Letters 81, 127–132 (2002)
19. Lu, Y., Vaudenay, S.: Faster Correlation Attack on Bluetooth Keystream Generator E0. In: Franklin, M. (ed.) CRYPTO 2004. LNCS, vol. 3152, pp. 407–425. Springer, Heidelberg (2004)
20. Matsui, M.: Linear Cryptanalysis Method for DES Cipher. In: Helleseth, T. (ed.) EUROCRYPT 1993. LNCS, vol. 765, pp. 386–397. Springer, Heidelberg (1994)
21. Meier, W., Staffelbach, O.: Fast Correlation Attacks on certain stream ciphers. Journal of Cryptology 159–176 (1989)
22. Menezes, A.J., Van Orschot, P.C., Vanstone, S.A.: Handbook of Applied Cryptography. CRC, Boca Raton (1996)
23. Mihaljević, M., Fossorier, M.P.C., Imai, H.: A Low-complexity and High-performance Algorithm for Fast Correlation Attack. In: Schneier, B. (ed.) FSE 2000. LNCS, vol. 1978, pp. 196–212. Springer, Heidelberg (2001)
24. Mihaljević, M., Fossorier, M.P.C., Imai, H.: Fast Correlation Attack Algorithm with listing decoding and an application. In: Matsui, M. (ed.) FSE 2001. LNCS, vol. 2355, pp. 208–222. Springer, Heidelberg (2002)
25. Molland, H., Helleseth, T.: An Improved Correlation Attack Against Irregular Clocked and Filtered Keystream Generators. In: Franklin, M. (ed.) CRYPTO 2004. LNCS, vol. 3152, pp. 373–389. Springer, Heidelberg (2004)
26. Siegenthaler, T.: Decrypting a Class of Stream Ciphers using ciphertext only. IEEE Transactions on Computer C-34, 81–85 (1985)

27. Wagner, D.: A Generalized Birthday Problem. In: Yung, M. (ed.) CRYPTO 2002. LNCS, vol. 2442, pp. 288–304. Springer, Heidelberg (2002)
28. Yarlagadda, R.K., Hershey, J.E.: Hadamard Matrix Analysis and Synthesis with Applications to Communications and Signal/Image Processing, pp. 17–22. Kluwer Academic, Dordrecht (1997)
29. Zhang, B., Wu, H., Feng, D., Bao, F.: A Fast Correlation attack on the shrinking generator. In: Menezes, A.J. (ed.) CT-RSA 2005. LNCS, vol. 3376, pp. 72–86. Springer, Heidelberg (2005)

Crossword Puzzle Attack on NLS

Joo Yeon Cho and Josef Pieprzyk

Centre for Advanced Computing – Algorithms and Cryptography,
Department of Computing, Macquarie University,
NSW, Australia, 2109
{jcho,josef}@ics.mq.edu.au

Abstract. NLS is one of the stream ciphers submitted to the eSTREAM project. We present a distinguishing attack on NLS by Crossword Puzzle (CP) attack method which is introduced in this paper. We build the distinguisher by using linear approximations of both the non-linear feedback shift register (NFSR) and the nonlinear filter function (NLF). Since the bias of the distinguisher depends on the *Konst* value, which is a key-dependent word, we present the graph showing how the bias of distinguisher vary with *Konst*. In result, we estimate the bias of the distinguisher to be around $O(2^{-30})$. Therefore, we claim that NLS is distinguishable from truly random cipher after observing $O(2^{60})$ keystream words. The experiments also show that our distinguishing attack is successful on 90.3% of *Konst* among 2^{32} possible values. We extend the CP attack to NLSv2 which is a tweaked version of NLS. In result, we build a distinguisher which has the bias of around 2^{-48}. Even though this attack is below the eSTREAM criteria (2^{-40}), the security margin of NLSv2 seems to be too low.

Keywords: Distinguishing Attacks, Crossword Puzzle Attack, Stream Ciphers, Linear Approximations, eSTREAM, Modular Addition, NLS, NLSv2.

1 Introduction

The European Network of Excellence in Cryptology (ECRYPT) launched a stream cipher project called eSTREAM [1] whose aim is to come up with a collection of stream ciphers that can be recommended to industry and government institutions as secure and efficient cryptographic primitives. It is also likely that some or perhaps all recommended stream ciphers may be considered as de facto industry standards. It is interesting to see a variety of different approaches used by the designers of the stream ciphers submitted to the eSTREAM call. A traditional approach for building stream ciphers is to use a linear feedback shift register (LFSR) as the main engine of the cipher. The outputs of the registers are taken and put into a nonlinear filter that produces the output stream that is added to the stream of plaintext.

One of the new trends in the design of stream ciphers is to replace LFSR by a nonlinear feedback shift register (NFSR). From the ciphers submitted to the

© Springer-Verlag Berlin Heidelberg 2007

eSTREAM call, there are several ciphers that use the structure based on NFSR. Amongst them the NLS cipher follows this design approach [4]. The designers of the NLS cipher are Philip Hawkes, Gregory Rose, Michael Paddon and Miriam Wiggers de Vries from Qualcomm Australia.

The paper studies the NLS cipher and its resistance against distinguishing attacks using linear approximation. [1] Typically, distinguishing attacks do not allow to recover any secret element of the cipher such as the cryptographic key or the secret initial state of the NFSR but instead they permit to tell apart the cipher from the truly random cipher. In this sense these attacks are relatively weak. However, the existence of a distinguishing attack is considered as an early warning sign of possible major security flaws.

In our analysis, we derive linear approximations for both NFSR and the non-linear filter (NLF). The main challenge has been to combine the obtained linear approximations in a such way that the internal state bits of NFSR have been eliminated leaving the observable output bits only. Our attack can be seen as a variant of the linear distinguishing attack, and we call it **"Crossword Puzzle" attack** (or shortly CP attack). The name is aligned with the intuition behind the attack in which the state bits of approximations vanish by combining them twice, horizontally and vertically.

Our approach is an extension of the linear distinguishing attack with linear masking (shortly, linear masking method) that was introduced by Coppersmith, Halevi, and Jutla in [3]. Note that the linear masking method was applied for the traditional stream ciphers based on LFSR so it is not directly applicable for the ciphers with NFSR.

The work is structured as follows. Section 2 presents a framework of CP attack. Section 3 briefly describes the NLS cipher. In Section 4, we study best linear approximations for both NFSR and NLF. A simplified NLS cipher is defined in Section 5 and we show how to design a distinguisher for it. Our distinguisher for the original NLS cipher is examined in Section 6 and an improvement using multiple distinguishers is in Section 7. In Section 8, CP attack is applied to NLSv2 which is a tweaked version of NLS. Section 9 concludes our work.

2 Framework of Crossword Puzzle (CP) Attack

In the CP attack, we construct a distinguisher based on linear approximations of both the non-linear feedback shift register (NFSR) and the non-linear filter (NLF). The attack is general and is applicable to the class of stream ciphers that combine a NFSR with nonlinear filters as long as there are "good enough" linear approximations. The roles of the two non-linear components are as follows.

– NFSR transforms the current state $\boldsymbol{s_i}$ into the next state $\boldsymbol{s_{i+1}}$ in a non-linear way using the appropriate function $NF1$, i.e. $\boldsymbol{s_{i+1}} := NF1(\boldsymbol{s_i})$ where $\boldsymbol{s_0}$ is the initial state and $i = 0, 1, 2, \ldots$.
– NLF produces an output $\boldsymbol{z_i}$ from the current state $\boldsymbol{s_i}$ through a non-linear function $NF2$, i.e. $\boldsymbol{z_i} := NF2(\boldsymbol{s_i})$.

[1] This is an extended version of [2].

$$
\begin{array}{ccccccccc}
l_1(\boldsymbol{s_{i_1}}) & = & u_1(\boldsymbol{s_{i_1}}) & + & u_2(\boldsymbol{s_{i_1}}) & + \cdots + & u_n(\boldsymbol{s_{i_1}}) & = & \boldsymbol{s_{i_1+1}} \\
l_1(\boldsymbol{s_{i_2}}) & = & u_1(\boldsymbol{s_{i_2}}) & + & u_2(\boldsymbol{s_{i_2}}) & + \cdots + & u_n(\boldsymbol{s_{i_2}}) & = & \boldsymbol{s_{i_2+1}} \\
\cdots & & \cdots & & & & & & \\
l_1(\boldsymbol{s_{i_m}}) & = & u_1(\boldsymbol{s_{i_m}}) & + & u_2(\boldsymbol{s_{i_m}}) & + \cdots + & u_n(\boldsymbol{s_{i_m}}) & = & \boldsymbol{s_{i_m+1}} \\
 & & \| & & \| & & \| & & \| \\
 & & l_3(\boldsymbol{z_{j_1}}) & & l_3(\boldsymbol{z_{j_2}}) & & l_3(\boldsymbol{z_{j_n}}) & & \boldsymbol{z_{j_{n+1}}}
\end{array}
$$

Fig. 1. An example of crossword puzzling

Let us define a bias ϵ of an approximation as $p = \frac{1}{2}(1+\epsilon), |\epsilon| > 0$ where p is the probability of the approximation.[2] The CP attack consists of the following steps (note that the operation $+$ is a binary (XOR) addition).

1. Find a linear approximation of the non-linear state transition function $NF1$ used by NFSR : $l_1(\boldsymbol{s_i}) = \boldsymbol{s_{i+1}}$ with bias of ϵ_1.
2. Find a linear approximation of the non-linear function $NF2$ applied by NLF : $l_2(\boldsymbol{s_j}) + l_3(\boldsymbol{z_j}) = 0$ with bias of ϵ_2.
3. Obtain two sets of clocks I and J such that $\sum_{i \in I}(l_1(\boldsymbol{s_i}) + \boldsymbol{s_{i+1}}) = \sum_{j \in J} l_2(\boldsymbol{s_j})$.
4. Build a distinguisher by computing
$$\sum_{i \in I}(l_1(\boldsymbol{s_i}) + \boldsymbol{s_{i+1}}) + \sum_{j \in J}(l_2(\boldsymbol{s_j}) + l_3(\boldsymbol{z_j})) = \sum_{j \in J} l_3(\boldsymbol{z_j}) = 0$$
which has bias of $\epsilon_1^{|I|} \cdot \epsilon_2^{|J|}$.

For the CP attack, it is an important task to find the approximations in Step 1 and Step 2 which have the relation required in Step 3. We describe a basic framework for achieving this task.

Given $l_1(\boldsymbol{s_i}) = \boldsymbol{s_{i+1}}$ from NFSR, we divide l_1 into n linear sub-functions $u_1, \ldots, u_n$. Then,

$$l_1(\boldsymbol{s_i}) = u_1(\boldsymbol{s_i}) + u_2(\boldsymbol{s_i}) + \cdots + u_n(\boldsymbol{s_i}) = \boldsymbol{s_{i+1}} \tag{1}$$

Suppose we set up a system of m approximations of l_1 on the clocks $i = i_1, \ldots, i_m$ as in Figure 1.

Now, we seek a linear approximation of NLF which has a form of $l_2(\boldsymbol{s_j}) = l_3(\boldsymbol{z_j})$ such that $l_2(\boldsymbol{s_j})$ corresponds to each column of m approximations of l_1. If there exist a set of such approximations which covers all columns of m approximations as Figure 1, then, those are

$$l_2(\boldsymbol{s_{j_t}}) = \sum_{k=1}^{m} u_t(\boldsymbol{s_{i_k}}) = l_3(\boldsymbol{z_{j_t}}), \quad t = 1, \ldots, n \tag{2}$$

and $\sum_{k=1}^{m} \boldsymbol{s_{i_k+1}} = \boldsymbol{z_{j_{n+1}}}$. Note that Approximation (2) corresponds each column of Approximation in Figure 1.

[2] This definition is useful for computing the bias of multiple approximations when the piling-up lemma [6] is applied. If we have n independent approximations, the probability of n approximations becomes $\frac{1}{2}(1+\epsilon^n)$.

By composing (or "Cross Puzzling") Approximations (1) and (2) in such a way that all the states vanish (as each state occurs twice), we compute a linear approximation

$$\sum_{i=1}^{n} l_3(\boldsymbol{z_{j_i}}) = \boldsymbol{z_{j_{n+1}}} \tag{3}$$

that is true with a non-zero bias. Clearly, Approximation (3) defines our distinguisher.

Discussion. There are more issues in regard to the bias of the distinguisher. Firstly, we assume that all approximations are independent. However, this may not be true since terms in the approximations could be related. The precise value of the bias can be computed by analysis of conditional probabilities of random variables of states involved in the approximations.

Secondly, when we set up a system of m approximations, we may choose different forms of approximations instead of a single approximation $l_1(\boldsymbol{s_i})$ that is used m times. In general, it is of interest to find approximations for both NFSR and NLF in order to maximize the bias of the distinguisher.

Note that the CP attack is reducible to the linear masking method [3] when the NFSR is replaced by a linear feedback shift register (LFSR) with $\epsilon_1 = 1$.

3 Brief Description of NLS Stream Cipher

The NLS keystream generator uses NFSR whose outputs are given to the non-linear filter NLF that produces output keystream bits. Note that we concentrate on the cipher itself and ignore its message integrity function as irrelevant to our analysis. For details of the cipher, the reader is referred to [4].

NLS has two components: NFSR and NLF that are synchronised by a clock. The state of NFSR at time t is denoted by $\sigma_t = (r_t[0], \ldots, r_t[16])$ where $r_t[i]$ is a 32-bit word. The state is determined by 17 words (or equivalently 544 bits). The transition from the state σ_t to the state σ_{t+1} is defined as follows:

1. $r_{t+1}[i] = r_t[i+1]$ for $i = 0, \ldots, 15$;
2. $r_{t+1}[16] = f((r_t[0] \lll 19) \boxplus (r_t[15] \lll 9) \boxplus Konst) \oplus r_t[4]$;
3. if $t = 0$ (modulo f16), $r_{t+1}[2] = r_{t+1}[2] \boxplus t$;

where $f16$ is 65537 and $\boxplus$ is the addition modulo 2^{32}. The $Konst$ value is a 32-bit key-dependent constant. The function $f : \{0,1\}^{32} \to \{0,1\}^{32}$ is constructed using an S-box with 8-bit input and 32-bit output and defined as $f(\omega) = \text{S-box}(\omega_{(H)}) \oplus \omega$ where $\omega_{(H)}$ is the most significant 8 bits of 32-bit word ω. Refer to Figure 2. Each output keystream word ν_t of NLF is obtained as

$$\nu_t = NLF(\sigma_t) = (r_t[0] \boxplus r_t[16]) \oplus (r_t[1] \boxplus r_t[13]) \oplus (r_t[6] \boxplus Konst). \tag{4}$$

The cipher uses 32-bit words to ensure a fast keystream generation.

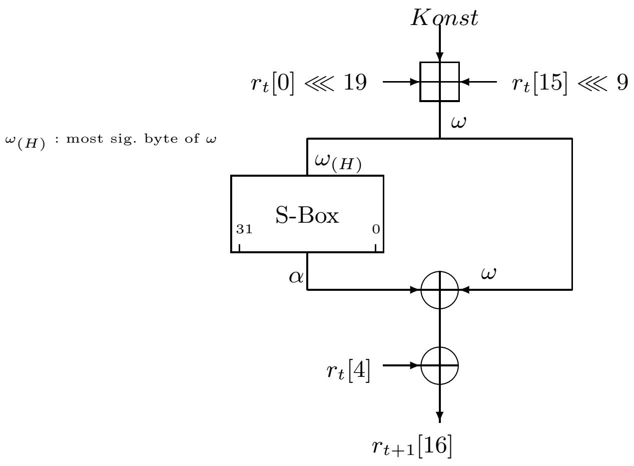


Fig. 2. The f function

4 Analysis of NFSR and NLF

Unlike a LFSR that applies a connection polynomial, the NFSR uses a much more complex nonlinear transition function f that mixes the XOR addition (linear) with the addition modulo 2^{32} (nonlinear). According to the structure of the non-linear shift register, the following equation holds for the least significant bit. Let us denote α_t to be a 32-bit output of the S-box that defines the transition function f. Then, we observe that the following equation holds for the least significant bit.

$$\alpha_{t,(0)} \oplus r_t[0]_{(13)} \oplus r_t[15]_{(23)} \oplus Konst_{(0)} \oplus r_t[4]_{(0)} \oplus r_{t+1}[16]_{(0)} = 0 \qquad (5)$$

where $\alpha_{t,(0)}$ and $x_{(i)}$ stand for the i-th bits of the 32-bit words α_t and x, respectively. (This notation will be used throughout the paper.)

To make our analysis simpler we assume initially that *Konst* is zero. This assumption is later dropped (i.e. *Konst* is non-zero) when we discuss our distinguishing attack on the NLS stream cipher.

4.1 Linear Approximations of $\alpha_{t,(0)}$

Recall that α_t is the 32-bit output taken from the S-box and $\alpha_{t,(0)}$ is its least significant bit. The input to the S-box comes from the eight most significant bits of the addition $((r_t[0] \lll 19) \boxplus (r_t[15] \lll 9) \boxplus Konst)$. Assuming that $Konst$ is zero, the input to S-box is $(r_t[0]' \boxplus r_t[15]')$, where $r_t[0]' = r_t[0] \lll 19$ and $r_t[15]' = r_t[15] \lll 9$. Thus, $\alpha_{t,(0)}$ is completely determined by the contents of two registers $r_t[0]'$ and $r_t[15]'$. Observe that the input of the S-box is affected by the eight most significant bits of the two registers $r_t[0]'$ (we denote the 8 most significant bits of the register by $r_t[0]'_{(H)}$) and $r_t[15]'$ (the 8 most significant bits

Table 1. Linear approximations for $\alpha_{t,(0)}$ when $Konst = 0$

linear approximations of $\alpha_{t,(0)}$	bias
$r_t[0]_{(10)} \oplus r_t[0]_{(6)} \oplus r_t[15]_{(20)} \oplus r_t[15]_{(16)} \oplus r_t[15]_{(15)}$	$1/2(1 + 0.048828)$
$r_t[0]_{(10)} \oplus r_t[0]_{(6)} \oplus r_t[0]_{(5)} \oplus r_t[15]_{(20)} \oplus r_t[15]_{(16)}$	$1/2(1 + 0.048828)$
$r_t[0]_{(12)} \oplus r_t[15]_{(22)}$	$1/2(1 - 0.045410)$
$r_t[0]_{(12)} \oplus r_t[0]_{(11)} \oplus r_t[0]_{(10)} \oplus r_t[15]_{(22)} \oplus r_t[15]_{(21)} \oplus r_t[15]_{(20)}$	$1/2(1 - 0.020020)$

of the register are denoted by $r_t[15]'_{(H)}$) and by the carry bit c generated by the addition of two 24 least significant bits of $r_t[0]'$ and $r_t[15]'$. Therefore

$$\text{the input of the S-box} = r_t[0]'_{(H)} \boxplus r_t[15]'_{(H)} \boxplus c.$$

Now we would like to find the best linear approximation for $\alpha_{t,(0)}$. We build the truth table with 2^{17} rows and 2^{16} columns. Each row corresponds to the unique collection of input variables (8 bits of $r_t[0]'_{(H)}$, 8 bits of $r_t[15]'_{(H)}$, and a single bit for c). Each column relates to the unique linear combination of bits from $r_t[0]'_{(H)}$ and $r_t[15]'_{(H)}$. Table 1 displays a collection of best linear approximations that are going to be used in our distinguishing attack. In particular, we see that the third approximation of Table 1 has high bias with only two terms. This seems to be caused by the fact that $r_t[0]_{(12)} \oplus r_t[15]_{(22)}$ is the only input to the MSB of input of the S-box that is not diffused to other order bits. Note that $r_t[0]'_{(H)} = (r_t[0] \lll 19)_{(H)} = (r_t[0]_{(12)}, \ldots, r_t[0]_{(5)})$ and $r_t[15]'_{(H)} = (r_t[15] \lll 9)_{(H)} = (r_t[15]_{(22)}, \ldots, r_t[15]_{(15)})$. Note also that none of the approximations contains the carry bit c, in other words, the approximations do not depend on c.

4.2 Linear Approximations for NFSR

Having a linear approximation of $\alpha_{t,(0)}$, it is easy to obtain a linear approximation for NFSR. For example, let us choose the first approximation from Table 1. Then, we have the following linear equation:

$$\alpha_{t,(0)} = r_t[0]_{(10)} \oplus r_t[0]_{(6)} \oplus r_t[15]_{(20)} \oplus r_t[15]_{(16)} \oplus r_t[15]_{(15)} \tag{6}$$

with the bias $0.048828 = 2^{-4.36}$. Now we combine Equations (5) and (6) and as the result we have the following approximation for NFSR

$$\begin{aligned} &r_t[0]_{(10)} \oplus r_t[0]_{(6)} \oplus r_t[15]_{(20)} \oplus r_t[15]_{(16)} \oplus r_t[15]_{(15)} \\ &\oplus r_t[0]_{(13)} \oplus r_t[15]_{(23)} \oplus Konst_{(0)} \oplus r_t[4]_{(0)} \oplus r_{t+1}[16]_{(0)} = 0 \end{aligned} \tag{7}$$

with the bias of $2^{-4.36}$.

4.3 Linear Approximations of Modular Addition

Let us take a look at the modular addition $\boxplus$. We know that the least significant bits are linear so the following equation holds

$$(r[x] \boxplus r[y])_{(0)} = r[x]_{(0)} \oplus r[y]_{(0)}. \tag{8}$$

All consecutive bits $i > 0$ of $\boxplus$ are nonlinear. Consider the function $(r[x] \boxplus r[y])_{(i)} \oplus (r[x] \boxplus r[y])_{(i-1)}$. We observe that the function has a linear approximation as follows

$$(r[x] \boxplus r[y])_{(i)} \oplus (r[x] \boxplus r[y])_{(i-1)} = r[x]_{(i)} \oplus r[y]_{(i)} \oplus r[x]_{(i-1)} \oplus r[y]_{(i-1)} \quad (9)$$

that has the bias of 2^{-1}.

In a similar way, we also observe that the function $(r[x] \boxplus r[y])_{(i)} \oplus (r[x] \boxplus r[y])_{(i-1)} \oplus (r[x] \boxplus r[y])_{(i-2)} \oplus (r[x] \boxplus r[y])_{(i-3)}$ has the following approximation. For $i > 2$,

$$(r[x] \boxplus r[y])_{(i)} \oplus (r[x] \boxplus r[y])_{(i-1)} \oplus (r[x] \boxplus r[y])_{(i-2)} \oplus (r[x] \boxplus r[y])_{(i-3)} = \\ r[x]_{(i)} \oplus r[y]_{(i)} \oplus r[x]_{(i-1)} \oplus r[y]_{(i-1)} \oplus r[x]_{(i-2)} \oplus r[y]_{(i-2)} \oplus r[x]_{(i-3)} \oplus r[y]_{(i-3)} \quad (10)$$

that has the bias of 2^{-2}.

4.4 Linear Approximation for NLF

Recall that Equation (4) defines the output keystream generated by NLF. By Equation (8), we obtain the relation for the least significant bits of NLF that takes the following form

$$\nu_{t,(0)} = (r_t[0]_{(0)} \oplus r_t[16]_{(0)}) \oplus (r_t[1]_{(0)} \oplus r_t[13]_{(0)}) \oplus (r_t[6]_{(0)} \oplus Konst_{(0)}). \quad (11)$$

This relation holds with probability one.

For $2 \leq i \leq 31$ and using Equation (9), we can argue that NLF function has linear approximations of the following form:

$$\begin{aligned} \nu_{t,(i)} \oplus \nu_{t,(i-1)} = & (r_t[0]_{(i)} \oplus r_t[16]_{(i)} \oplus r_t[0]_{(i-1)} \oplus r_t[16]_{(i-1)}) \\ & \oplus (r_t[1]_{(i)} \oplus r_t[13]_{(i)} \oplus r_t[1]_{(i-1)} \oplus r_t[13]_{(i-1)}) \\ & \oplus (r_t[6]_{(i)} \oplus Konst_{(i)} \oplus r_t[6]_{(i-1)} \oplus Konst_{(i-1)}) \end{aligned} \quad (12)$$

with the bias of $(2^{-1})^2 = 2^{-2}$ under the condition that $Konst = 0$.

Also applying Approximation (10), for $i > 2$, we get the following expression

$$\begin{aligned} \nu_{t,(i)} & \oplus \nu_{t,(i-1)} \oplus \nu_{t,(i-2)} \oplus \nu_{t,(i-3)} = \\ & (r_t[0]_{(i)} \oplus r_t[0]_{(i-1)} \oplus r_t[0]_{(i-2)} \oplus r_t[0]_{(i-3)} \oplus r_t[16]_{(i)} \oplus r_t[16]_{(i-1)} \\ & \oplus r_t[16]_{(i-2)} \oplus r_t[16]_{(i-3)}) \oplus (r_t[1]_{(i)} \oplus r_t[1]_{(i-1)} \oplus r_t[1]_{(i-2)} \oplus r_t[1]_{(i-3)} \\ & \oplus r_t[13]_{(i)} \oplus r_t[13]_{(i-1)} \oplus r_t[13]_{(i-2)} \oplus r_t[13]_{(i-3)}) \oplus (r_t[6]_{(i)} \oplus r_t[6]_{(i-1)} \\ & \oplus r_t[6]_{(i-2)} \oplus r_t[6]_{(i-3)} \oplus Konst_{(i)} \oplus Konst_{(i-1)} \oplus Konst_{(i-2)} \oplus Konst_{(i-3)}) \end{aligned} \quad (13)$$

that has the bias of $(2^{-2})^2 = 2^{-4}$ when $Konst = 0$.

For non-zero $Konst$, the bias of Approximations (12) and (13) will be studied in Section 6.2.

5 CP Attack on a Simplified NLS

In this Section we present the CP attack on a simplified NLS. This is a preliminary stage of our attack in which we apply the initial idea of crossword puzzle attack that will be later developed and generalized. We assume that the structure of NFSR is unchanged but the structure of NLF is modified by replacing the addition $\boxplus$ by $\oplus$. Thus, Equation (4) that describes the keystream becomes

$$\mu_t = (r_t[0] \oplus r_t[16]) \oplus (r_t[1] \oplus r_t[13]) \oplus (r_t[6] \oplus Konst). \tag{14}$$

This linear function is valid for 32-bit words so it can be equivalently re-written as a system of 32 equations each equation valid for the particular ith bit. Hence, for $0 \leq i \leq 31$, we can write

$$\mu_{t,(i)} = (r_t[0]_{(i)} \oplus r_t[16]_{(i)}) \oplus (r_t[1]_{(i)} \oplus r_t[13]_{(i)}) \oplus (r_t[6]_{(i)} \oplus Konst_{(i)}). \tag{15}$$

To build a distinguisher we combine approximations of NFSR given by Equation (7) with linear equations defined by Equation (15). For the clocks t, $t+1$, $t+6$, $t+13$, and $t+16$, consider the following approximations of NFSR

$$\begin{array}{lllll} r_t[0]_{(10)} & \oplus\, r_t[0]_{(6)} & \oplus\, r_t[15]_{(20)} & \oplus \cdots \oplus r_{t+1}[16]_{(0)} & = 0 \\ r_{t+1}[0]_{(10)} & \oplus\, r_{t+1}[0]_{(6)} & \oplus\, r_{t+1}[15]_{(20)} & \oplus \cdots \oplus r_{t+2}[16]_{(0)} & = 0 \\ r_{t+6}[0]_{(10)} & \oplus\, r_{t+6}[0]_{(6)} & \oplus\, r_{t+6}[15]_{(20)} & \oplus \cdots \oplus r_{t+7}[16]_{(0)} & = 0 \\ r_{t+13}[0]_{(10)} & \oplus\, r_{t+13}[0]_{(6)} & \oplus\, r_{t+13}[15]_{(20)} & \oplus \cdots \oplus r_{t+14}[16]_{(0)} & = 0 \\ r_{t+16}[0]_{(10)} & \oplus\, r_{t+16}[0]_{(6)} & \oplus\, r_{t+16}[15]_{(20)} & \oplus \cdots \oplus r_{t+17}[16]_{(0)} & = 0 \end{array} \tag{16}$$

Since $r_{t+p}[0] = r_t[p]$, we can rewrite the above system of equations (16) equivalently as follows:

$$\begin{array}{lllll} r_t[0]_{(10)} & \oplus\, r_t[0]_{(6)} & \oplus\, r_{t+15}[0]_{(20)} & \oplus \cdots \oplus r_{t+17}[0]_{(0)} & = 0 \\ r_t[1]_{(10)} & \oplus\, r_t[1]_{(6)} & \oplus\, r_{t+15}[1]_{(20)} & \oplus \cdots \oplus r_{t+17}[1]_{(0)} & = 0 \\ r_t[6]_{(10)} & \oplus\, r_t[6]_{(6)} & \oplus\, r_{t+15}[6]_{(20)} & \oplus \cdots \oplus r_{t+17}[6]_{(0)} & = 0 \\ r_t[13]_{(10)} & \oplus\, r_t[13]_{(6)} & \oplus\, r_{t+15}[13]_{(20)} & \oplus \cdots \oplus r_{t+17}[13]_{(0)} & = 0 \\ r_t[16]_{(10)} & \oplus\, r_t[16]_{(6)} & \oplus\, r_{t+15}[16]_{(20)} & \oplus \cdots \oplus r_{t+17}[16]_{(0)} & = 0 \end{array} \tag{17}$$

Consider the columns of the above system of equations. Each column describes a single bit output of the filter (see Equation (15)), therefore the system (17) gives the following approximation:

$$\begin{array}{l} \mu_{t,(10)} \oplus \mu_{t,(6)} \oplus \mu_{t+15,(20)} \oplus \mu_{t+15,(16)} \oplus \mu_{t+15,(15)} \oplus \mu_{t,(13)} \\ \oplus \mu_{t+15,(23)} \oplus \mu_{t+4,(0)} \oplus \mu_{t+17,(0)} = K \end{array} \tag{18}$$

where $K = Konst_{(10)} \oplus Konst_{(6)} \oplus Konst_{(20)} \oplus Konst_{(16)} \oplus Konst_{(15)} \oplus Konst_{(13)} \oplus Konst_{(23)}$. Note that the bit K is constant (zero or one) during the session. Therefore, the bias of Approximation (18) is $(\mathbf{2^{-4.36}})^{\mathbf{5}} = \mathbf{2^{-21.8}}$.

6 The CP Attack on NLS

In this section, we describe the CP attack on the real NLS. The main idea is to find the best combination of approximations for both NFSR and NLF, while the state bits of the shift register vanish and the bias of the resulting approximation is as big as possible. We study the case for $Konst = 0$ at first and then, extend our attack to the case for $Konst \neq 0$. Since NLS allows only a non-zero most significant byte of $Konst$, the second case corresponds to the real NLS.

6.1 Case for $Konst = 0$

The linear approximations of $\alpha_{t,(0)}$ are given in Table 1. For the most effective distinguisher, we choose this time the third approximation from the table which is

$$\alpha_{t,(0)} = r_t[0]_{(12)} \oplus r_t[15]_{(22)} \tag{19}$$

and the bias of this approximation is $0.045410 = 2^{-4.46}$. By combining Equations (5) and (19), we have the following approximation

$$r_t[0]_{(12)} \oplus r_t[15]_{(22)} \oplus r_t[0]_{(13)} \oplus r_t[15]_{(23)} \oplus r_t[4]_{(0)} \oplus r_{t+1}[16]_{(0)} = 0 \tag{20}$$

that has the same bias.

Let us now divide (20) into two parts : the least significant bits and the other bits, so we get

$$\begin{aligned} l_1(r_t) &= r_t[4]_{(0)} \oplus r_{t+1}[16]_{(0)} \\ l_2(r_t) &= r_t[0]_{(12)} \oplus r_t[0]_{(13)} \oplus r_t[15]_{(22)} \oplus r_t[15]_{(23)} \end{aligned} \tag{21}$$

Clearly, $l_1(r_t) \oplus l_2(r_t) = 0$ with the bias $2^{-4.46}$. Since $l_1(r_t)$ has only the least significant bit variables, we apply (11) which is true with the probability one. Then, we obtain

$$\begin{aligned} l_1(r_t) &= r_t[4]_{(0)} \oplus r_{t+1}[16]_{(0)} \\ l_1(r_{t+1}) &= r_{t+1}[4]_{(0)} \oplus r_{t+2}[16]_{(0)} \\ l_1(r_{t+6}) &= r_{t+6}[4]_{(0)} \oplus r_{t+7}[16]_{(0)} \\ l_1(r_{t+13}) &= r_{t+13}[4]_{(0)} \oplus r_{t+14}[16]_{(0)} \\ l_1(r_{t+16}) &= r_{t+16}[4]_{(0)} \oplus r_{t+17}[16]_{(0)} \end{aligned} \tag{22}$$

If we add up all approximations of (22), then, by applying Equation (11), we can write

$$l_1(r_t) \oplus l_1(r_{t+1}) \oplus l_1(r_{t+6}) \oplus l_1(r_{t+13}) \oplus l_1(r_{t+16}) = \nu_{t+4,(0)} \oplus \nu_{t+17,(0)} \tag{23}$$

Now, we focus on $l_2(r_t)$ where the bit positions are 12, 13, 22, and 23, then,

$$\begin{aligned} l_2(r_t) &= r_t[0]_{(12)} \oplus r_t[0]_{(13)} \oplus r_t[15]_{(22)} \oplus r_t[15]_{(23)} \\ l_2(r_{t+1}) &= r_{t+1}[0]_{(12)} \oplus r_{t+1}[0]_{(13)} \oplus r_{t+1}[15]_{(22)} \oplus r_{t+1}[15]_{(23)} \\ l_2(r_{t+6}) &= r_{t+6}[0]_{(12)} \oplus r_{t+6}[0]_{(13)} \oplus r_{t+6}[15]_{(22)} \oplus r_{t+6}[15]_{(23)} \\ l_2(r_{t+13}) &= r_{t+13}[0]_{(12)} \oplus r_{t+13}[0]_{(13)} \oplus r_{t+13}[15]_{(22)} \oplus r_{t+13}[15]_{(23)} \\ l_2(r_{t+16}) &= r_{t+16}[0]_{(12)} \oplus r_{t+16}[0]_{(13)} \oplus r_{t+16}[15]_{(22)} \oplus r_{t+16}[15]_{(23)} \end{aligned} \tag{24}$$

Since $r_{t+p}[0] = r_t[p]$, the above approximations can be presented as follows.

$$\begin{aligned} l_2(r_t) &= r_t[0]_{(12)} \oplus r_t[0]_{(13)} \oplus r_{t+15}[0]_{(22)} \oplus r_{t+15}[0]_{(23)} \\ l_2(r_{t+1}) &= r_t[1]_{(12)} \oplus r_t[1]_{(13)} \oplus r_{t+15}[1]_{(22)} \oplus r_{t+15}[1]_{(23)} \\ l_2(r_{t+6}) &= r_t[6]_{(12)} \oplus r_t[6]_{(13)} \oplus r_{t+15}[6]_{(22)} \oplus r_{t+15}[6]_{(23)} \\ l_2(r_{t+13}) &= r_t[13]_{(12)} \oplus r_t[13]_{(13)} \oplus r_{t+15}[13]_{(22)} \oplus r_{t+15}[13]_{(23)} \\ l_2(r_{t+16}) &= r_t[16]_{(12)} \oplus r_t[16]_{(13)} \oplus r_{t+15}[16]_{(22)} \oplus r_{t+15}[16]_{(23)} \end{aligned} \tag{25}$$

Recall the approximation (12) of NLF. If we combine (25) with (12), then we have the following approximation.

$$\begin{aligned} &l_2(r_t) \oplus l_2(r_{t+1}) \oplus l_2(r_{t+6}) \oplus l_2(r_{t+13}) \oplus l_2(r_{t+16}) = \\ &\nu_{t,(12)} \oplus \nu_{t,(13)} \oplus \nu_{t+15,(22)} \oplus \nu_{t+15,(23)} \end{aligned} \tag{26}$$

By combining the approximations (23) and (26), we obtain the final approximation that defines our distinguisher, i.e.

$$\begin{aligned} &l_1(r_t) \oplus l_1(r_{t+1}) \oplus l_1(r_{t+6}) \oplus l_1(r_{t+13}) \oplus l_1(r_{t+16}) \\ &\oplus l_2(r_t) \oplus l_2(r_{t+1}) \oplus l_2(r_{t+6}) \oplus l_2(r_{t+13}) \oplus l_2(r_{t+16}) \\ &= \nu_{t,(12)} \oplus \nu_{t,(13)} \oplus \nu_{t+15,(22)} \oplus \nu_{t+15,(23)} \oplus \nu_{t+4,(0)} \oplus \nu_{t+17,(0)} \\ &= 0 \end{aligned} \tag{27}$$

The second part of the approximation can be computed from the output keystream that is observable to the adversary. As we use Approximation (20) five times and Approximation (12) twice, the bias of the approximation (27) is $\mathbf{(2^{-4.46})^5 \cdot (2^{-2})^2 = 2^{-26.3}}$.

6.2 Case for $Konst \neq 0$

Since the word $Konst$ occurs in NFSR and NLF as a parameter, the biases of linear approximations of both $\alpha_{t,(0)}$ and NLF vary with $Konst$. If we divide $Konst$ into two parts as $Konst = (Konst_{(H)}, Konst_{(L)})$ where $Konst_{(H)} = (Konst_{(31)}, \ldots, Konst_{(24)})$, and $Konst_{(L)} = (Konst_{(23)}, \ldots, Konst_{(0)})$, then, linear approximations of $\alpha_{t,(0)}$ mainly depend on $Konst_{(H)}$ and those of NLF depend on $Konst_{(L)}$.

Biases of $\boldsymbol{\alpha_{t,(0)}}$ with non-zero $\boldsymbol{Konst_{(H)}}$. Since the most significant 8 bits of $Konst$ mainly contribute to the form of the bit $\alpha_{t,(0)}$, the bias of Approximation (19) fluctuates according to the 8-bit $Konst_{(H)}$. This relation is illustrated in Figure 3.

From this figure, we can see that the bias of Approximation (19) becomes the smallest when $Konst_{(H)}$ is around 51 and 179 and the biggest when $Konst_{(H)}$ is around 127 and 255. The average bias of (19) with $Konst_{(H)}$ is $2^{-5.19}$.

Biases of NLF with $\boldsymbol{Konst_{(L)}}$. Figure 4 displays the bias variation of Approximation (12) according to $Konst_{(L)}$ at $i = 13$. Note that the graph shows the bias distribution from 14 LSBs of $Konst_{(L)}$ (that is, 2^{14}) since the bits $Konst_{(23)}, \ldots, Konst_{(14)}$ do not effect the bias for $i = 13$. We do not display the graph of Approximation (12) at $i = 23$ because the graph is similar to Figure 4 with the slope changed as we consider 24 bits of $Konst_{(L)}$ only. On the average, the bias of (12) is 2^{-3} for any $i > 0$.

6.3 Bias of the Distinguisher

Let us denote the bias of Approximation (19) for NFSR by ϵ_1, the bias of Approximation (12) for NLF at $i = 13$ and $i = 23$ by $\epsilon_{2,13}$ and $\epsilon_{2,23}$ respectively.

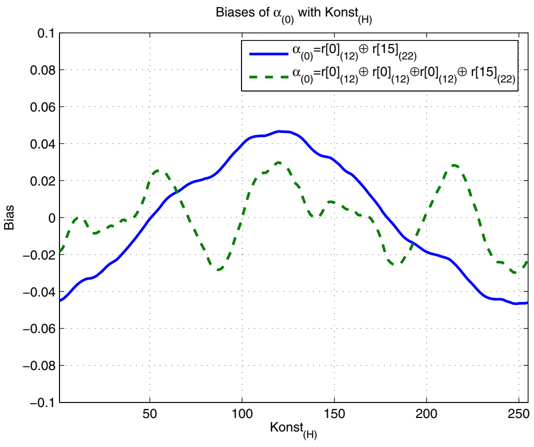


Fig. 3. Variation of biases of two $\alpha_{t,(0)}$ approximations with $Konst_{(H)}$

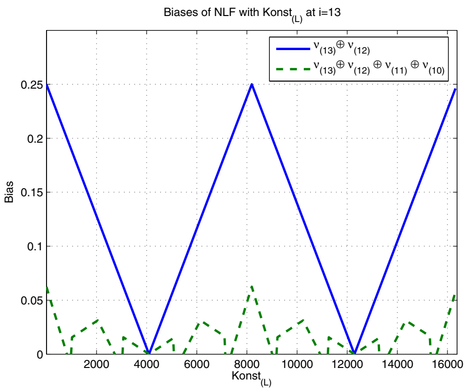


Fig. 4. Variation of biases of NLF with $Konst_{(L)}$ at $i = 13$

Note that all biases are $Konst$-dependent values. Since the $Konst$ is generated by randomization process at the initialization stage, it is reasonable assumption that all the $Konst$ values are equiprobable.

Hence, the bias of the distinguisher ϵ can be calculated as follows.

$$\epsilon = 2^{-32} \sum_{k=0}^{2^{32}-1} (\epsilon_1^5 \cdot \epsilon_{2,13} \cdot \epsilon_{2,23} | Konst = k)$$

See Appendix A for detail algorithm to compute the bias with a low complexity. Experiments shows that the bias of distinguisher appears to be 2^{-30}.

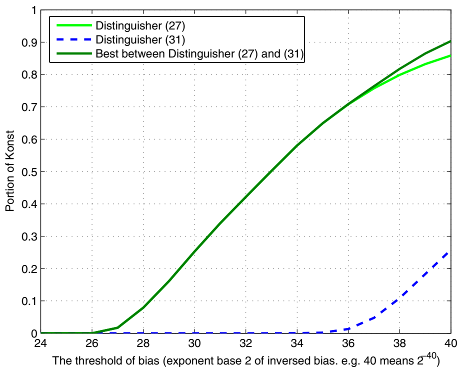


Fig. 5. The success rate of attack

6.4 The Success Rate of Distinguishing Attack

Since the specification of the NLS cipher allows the adversary to observe up to 2^{80} keystream words per one key/nonce pair, we assume that our attack is successful if the bias of distinguisher satisfies the following condition:

$$\epsilon_1^5 \cdot \epsilon_{2,13} \cdot \epsilon_{2,23} > 2^{-40}. \tag{28}$$

The experiments show that the bias of Distinguisher (27) satisfies the condition (28) on around 85.9% of $Konst$. See Figure 5.

7 Improving Distinguishing Attack by Multiple Distinguishers

In this section, we present multiple distinguishing attack for the purpose of reducing the portion of $Konst$ for which our attack fails. multiple approximations of $\alpha_{(0)}$. Since the NLS produces 32-bit keystream word per a clock, the actual volume of data required for the attack with multiple distinguishers is not increased even though more computation is required.

The motivation for multiple distinguishers is the fact that Approximation (19) is not always best for a distinguisher for all the possible values of $Konst$. For instance, the bias of the distinguisher based on Approximation (19) is very small for some values of $Konst_{(H)}$ (e.g. $Konst_{(H)} = 51$ or 179). In order to address this problem, we choose the fourth approximation from Table 1. Then, we have

$$\alpha_{t,(0)} = r_t[0]_{(12)} \oplus r_t[0]_{(11)} \oplus r_t[0]_{(10)} \oplus r_t[15]_{(22)} \oplus r_t[15]_{(21)} \oplus r_t[15]_{(20)} \tag{29}$$

which has the smallest bias when $Konst_{(H)}$ is around 41, 139 and 169 whereas the biggest when $Konst_{(H)}$ is around 57 and 185. The average bias of (29) is $2^{-6.2}$ when only absolute values are taken (see Figure 3).

Using this approximation, we build an approximation of NFSR as follows

$$r_t[0]_{(10)} \oplus r_t[0]_{(11)} \oplus r_t[0]_{(12)} \oplus r_t[0]_{(13)} \oplus r_t[15]_{(20)} \oplus r_t[15]_{(21)} \oplus r_t[15]_{(22)} \oplus r_t[15]_{(23)} \oplus Konst_{(0)} \oplus r_t[4]_{(0)} \oplus r_{t+1}[16]_{(0)} = 0. \tag{30}$$

Then, we can construct a new distinguisher by combining Approximation (13) on NLF. We omit the detail process due to the similarity of Distinguisher (27). In result, we have the following new distinguisher

$$\nu_{t,(10)} \oplus \nu_{t,(11)} \oplus \nu_{t,(12)} \oplus \nu_{t,(13)} \oplus \nu_{t+15,(20)} \oplus \nu_{t+15,(21)} \oplus \nu_{t+15,(22)} \oplus \nu_{t+15,(23)} \oplus \nu_{t+4,(0)} \oplus \nu_{t+17,(0)} = 0. \tag{31}$$

The bias of Distinguisher (31) can be calculated with a similar way to Section 6.3. In result, the bias of distinguisher appears to be $2^{-27.8} \cdot 2^{-10} = 2^{-37.8}$.

By observing two distinguishers together and selecting always the better bias among them, we improve the success rate of the distinguishing attack. The experiments show that the combined bias for Distinguishers (27) and (31) satisfies the condition (28) for around 90.3% of $Konst$ (see Figure 5).

8 The CP Attack on NLSv2

NLSv2 is a tweaked version of NLS [5]. The major difference from NLS is that $Konst$ is set to the output of the non-linear filter at every 65537 clock of the NFSR. This output is not used in the keystream.

Since Approximation (19) is biased positively or negatively according to $Konst$ (see Figure 3), the sign of the bias of distinguisher (27) also varies with $Konst$ value. Since the distinguisher (27) uses Approximation (20) five times, randomly changed $Konst$ could reduce the bias of distinguisher on the average.

However, this tweak version does not seem to have enough security margin against the CP attack. If a distinguisher uses the "even" number of linear approximations for NFSR then, the bias of the distinguisher becomes always positive irrespective of the sign of Approximation (27).

The smallest even number of approximations we found is eight, which is obtained by the addition of two consecutive outputs of NLF. Then, we apply the CP attack to NLSv2 with eight approximations of NFSR where the state positions are determined by two consecutive outputs of NLF. For detail approach to the CP attack against NLSv2, see Appendix B.

In summary, we estimate the bias of distinguisher by the similar way to Section 6.3. Experiments using the algorithm in Appendix A show that the bias is around $2^{-37.6} \cdot 2^{-10.4} = 2^{-48}$. Note that the bias of the distinguisher is always positive since $\epsilon_1^8 = (-\epsilon_1)^8 > 0$.

9 Conclusion

We presented a distinguishing attack on the NLS cipher using Crossword Puzzle attack. The bias of our distinguisher appears to be 2^{-30} so the NLS cipher is distinguishable from a random function by observing 2^{60} keystream words. Even though there is a fraction of the $Konst$ values which requires the data complexity bigger than 2^{80}, we show that it is possible for attacker to reduce this fraction of $Konst$ substantially by combining multiple distinguishers which have biases of less than 2^{-40}. We also have constructed a distinguisher for the tweaked version of the cipher called NLSv2. Although the distinguisher does not break the cipher, it shows that the security margin is too small to guarantee the claimed security level for the near future.

Acknowledgment. We are very grateful to Philip Hawkes and anonymous referees of SASC 2006 and SAC 2006 for their invaluable comments. The second author acknowledges the support received from Australian Research Council (projects DP0451484 and DP0663452).

References

1. eSTREAM project. `http://www.ecrypt.eu.org/stream/`
2. Cho, J.Y., Pieprzyk, J.: Linear distinguishing attack on NLS. In: SASC 2006 workshop (2006)
3. Coppersmith, D., Halevi, S., Jutla, C.: Cryptanalysis of stream ciphers with linear masking. In: Yung, M. (ed.) CRYPTO 2002. LNCS, vol. 2442, pp. 515–532. Springer, Heidelberg (2002)
4. Hawkes, P., Paddon, M., Rose, G., de Vries, M.W.: Primitive specification for NLS (April 2005), `http://www.ecrypt.eu.org/stream/nls.html`
5. Hawkes, P., Paddon, M., Rose, G., de Vries, M.W.: Primitive specification for NLSv2 (March 2006), `http://www.ecrypt.eu.org/stream/nls.html`
6. Matsui, M.: Linear cryptanalysis method for DES cipher. In: Helleseth, T. (ed.) EUROCRYPT 1993. LNCS, vol. 765, pp. 386–397. Springer, Heidelberg (1994)

A Low Complexity Algorithm for the Bias of Distinguisher

According to Section 6.2, the bias of the distinguisher (27) can be computed using the following algorithm. Note that $Konst$ is expressed in hexadecimal.

1. Set $Konst = 01000000h$ (Note that non-zero $Konst_{(H)}$ is allowed in NLS.)
2. Find the bias ϵ_1 of Approximation (19) for NFSR.
3. Find the bias ϵ_2 of Approximation (12) for NLF.
4. Compute and store the bias ϵ of the distinguisher (27) by $\epsilon = \epsilon_{\mathbf{1}}^{\mathbf{5}} \cdot \epsilon_{\mathbf{2}}^{\mathbf{2}}$.
5. Increase $Konst$ by 1 and repeat Step 2,3 and 4 until $Konst = ffffffffh$.
6. Compute the estimation of ϵ.

In order to reduce the complexity of computing the estimation of ϵ, we assume that ϵ_1 is affected by only $Konst_{(H)}$, not by $Konst_{(L)}$ in Step 2. Then, ϵ_1 and

ϵ_2 can be computed independently. Therefore, the above algorithm is amended as follows.

1. Set $Konst_{(H)} = 01h$
2. Find the bias ϵ_1 of Approximation (19) and store $\epsilon_1^* = \epsilon_1^5$.
3. Increase $Konst_{(H)}$ by 1 and repeat Step 2 until $Konst_{(H)} = ffh$.
4. Set $Konst_{(L)} = 000000h$
5. Find two biases of Approximation (12) at $i = 13$ and $i = 23$, which is called $\epsilon_{2,13}$ and $\epsilon_{2,23}$ respectively.
6. Store ϵ_2^* by calculating $\epsilon_2 = \epsilon_{2,13} \cdot \epsilon_{2,23}$.
7. Increase $Konst_{(L)}$ by 1 and repeat Step 5 and 6 until $Konst_{(L)} = 00ffffffh$.
8. Compute the estimation of the bias of distinguisher (27) under the assumption that $Konst$ is equiprobable.

B The CP Attack on NLSv2

NLSv2 is a tweaked version of NLS [5]. We apply the CP attack to NLSv2 with eight approximations of NFSR where the state positions are determined by two consecutive outputs of NLF.

B.1 Linear Approximations of NLSv2

Suppose we have two consecutive outputs of NLF as follows.

$$\begin{aligned}
\nu_{t,(i)} \oplus \nu_{t,(i-1)} &= (r_t[0]_{(i)} \oplus r_t[16]_{(i)} \oplus r_t[0]_{(i-1)} \oplus r_t[16]_{(i-1)}) \\
&\oplus (r_t[1]_{(i)} \oplus r_t[13]_{(i)} \oplus r_t[1]_{(i-1)} \oplus r_t[13]_{(i-1)}) \\
&\oplus (r_t[6]_{(i)} \oplus Konst_{(i)} \oplus r_t[6]_{(i-1)} \oplus Konst_{(i-1)}) \\
\nu_{t+1,(i)} \oplus \nu_{t+1,(i-1)} &= (r_{t+1}[0]_{(i)} \oplus r_{t+1}[16]_{(i)} \oplus r_{t+1}[0]_{(i-1)} \oplus r_{t+1}[16]_{(i-1)}) \\
&\oplus (r_{t+1}[1]_{(i)} \oplus r_{t+1}[13]_{(i)} \oplus r_{t+1}[1]_{(i-1)} \oplus r_{t+1}[13]_{(i-1)}) \\
&\oplus (r_{t+1}[6]_{(i)} \oplus Konst_{(i)} \oplus r_{t+1}[6]_{(i-1)} \oplus Konst_{(i-1)}) \\
&= (r_t[1]_{(i)} \oplus r_t[17]_{(i)} \oplus r_t[1]_{(i-1)} \oplus r_t[17]_{(i-1)}) \\
&\oplus (r_t[2]_{(i)} \oplus r_t[14]_{(i)} \oplus r_t[2]_{(i-1)} \oplus r_t[14]_{(i-1)}) \\
&\oplus (r_t[7]_{(i)} \oplus Konst_{(i)} \oplus r_t[7]_{(i-1)} \oplus Konst_{(i-1)})
\end{aligned} \tag{32}$$

By adding up two approximations, we have

$$\begin{aligned}
&\nu_{t,(i)} \oplus \nu_{t,(i-1)} \oplus \nu_{t+1,(i)} \oplus \nu_{t+1,(i-1)} = \\
&= (r_t[0]_{(i)} \oplus r_t[16]_{(i)} \oplus r_t[0]_{(i-1)} \oplus r_t[16]_{(i-1)}) \\
&\oplus (r_t[1]_{(i)} \oplus r_t[13]_{(i)} \oplus r_t[1]_{(i-1)} \oplus r_t[13]_{(i-1)}) \\
&\oplus (r_t[6]_{(i)} \oplus Konst_{(i)} \oplus r_t[6]_{(i-1)} \oplus Konst_{(i-1)}) \\
&\oplus (r_t[1]_{(i)} \oplus r_t[17]_{(i)} \oplus r_t[1]_{(i-1)} \oplus r_t[17]_{(i-1)}) \\
&\oplus (r_t[2]_{(i)} \oplus r_t[14]_{(i)} \oplus r_t[2]_{(i-1)} \oplus r_t[14]_{(i-1)}) \\
&\oplus (r_t[7]_{(i)} \oplus Konst_{(i)} \oplus r_t[7]_{(i-1)} \oplus Konst_{(i-1)}) \\
&= (r_t[0]_{(i)} \oplus r_t[0]_{(i-1)}) \oplus (r_t[2]_{(i)} \oplus r_t[2]_{(i-1)}) \\
&\oplus (r_t[6]_{(i)} \oplus r_t[6]_{(i-1)}) \oplus (r_t[7]_{(i)} \oplus r_t[7]_{(i-1)}) \\
&\oplus (r_t[13]_{(i)} \oplus r_t[13]_{(i-1)}) \oplus (r_t[14]_{(i)} \oplus r_t[14]_{(i-1)}) \\
&\oplus (r_t[16]_{(i)} \oplus r_t[16]_{(i-1)}) \oplus (r_t[17]_{(i)} \oplus r_t[17]_{(i-1)})
\end{aligned} \tag{33}$$

The experiment shows that Approximation (33) has bias of around $2^{-5.2}$. Since $r_t[1]_{(i)} \oplus r_t[1]_{(i-1)}$ and $Konst_{(i)} \oplus Konst_{(i-1)}$ are canceled out, the bias of (33) is higher than the multiplication of bias of two approximation in (32). Hence, we use (33) for the approximation of NLF where the state position are $0, 2, 6, 7, 13, 14, 16, 17$. Note that the bias of (33) is still dependent on the value of $Konst$.

According to these state positions, the least significant bits have the following relation.

$$\nu_{t,(0)} \oplus \nu_{t+1,(0)} = r_t[0]_{(0)} \oplus r_t[2]_{(0)} \oplus r_t[6]_{(0)} \oplus r_t[7]_{(0)} \oplus r_t[13]_{(0)} \oplus r_t[14]_{(0)} \oplus r_t[16]_{(0)} \oplus r_t[17]_{(0)} \tag{34}$$

This relation also holds with probability one.

B.2 Building Distinguisher

This section is similar to Section 6 except that the eight (instead of five) approximations from the state position $0, 2, 6, 7, 13, 14, 16, 17$ are used for the CP attack.

Let us recall Approximation (21). For the least significant bits, we can write

$$\begin{array}{ll} l_1(r_t) &= r_t[4]_{(0)} \oplus r_{t+1}[16]_{(0)} \\ l_1(r_{t+2}) &= r_{t+2}[4]_{(0)} \oplus r_{t+3}[16]_{(0)} \\ l_1(r_{t+6}) &= r_{t+6}[4]_{(0)} \oplus r_{t+7}[16]_{(0)} \\ l_1(r_{t+7}) &= r_{t+7}[4]_{(0)} \oplus r_{t+8}[16]_{(0)} \\ l_1(r_{t+13}) &= r_{t+13}[4]_{(0)} \oplus r_{t+14}[16]_{(0)} \\ l_1(r_{t+14}) &= r_{t+14}[4]_{(0)} \oplus r_{t+15}[16]_{(0)} \\ l_1(r_{t+16}) &= r_{t+16}[4]_{(0)} \oplus r_{t+17}[16]_{(0)} \\ l_1(r_{t+17}) &= r_{t+17}[4]_{(0)} \oplus r_{t+18}[16]_{(0)} \end{array} \tag{35}$$

If we add up all approximations of (35), then, by applying Equation (34), we obtain

$$l_1(r_t) \oplus l_1(r_{t+2}) \oplus l_1(r_{t+6}) \oplus l_1(r_{t+7}) \oplus l_1(r_{t+13}) \oplus l_1(r_{t+14}) \oplus l_1(r_{t+16}) \oplus l_1(r_{t+17}) = \nu_{t+4,(0)} \oplus \nu_{t+5,(0)} \oplus \nu_{t+17,(0)} \oplus \nu_{t+18,(0)} \tag{36}$$

If we focus on $l_2(r_t)$ where the bit positions are 12, 13, 22, and 23, then,

$$\begin{array}{ll} l_2(r_t) &= r_t[0]_{(12)} \oplus r_t[0]_{(13)} \oplus r_t[15]_{(22)} \oplus r_t[15]_{(23)} \\ l_2(r_{t+2}) &= r_{t+2}[0]_{(12)} \oplus r_{t+2}[0]_{(13)} \oplus r_{t+2}[15]_{(22)} \oplus r_{t+2}[15]_{(23)} \\ l_2(r_{t+6}) &= r_{t+6}[0]_{(12)} \oplus r_{t+6}[0]_{(13)} \oplus r_{t+6}[15]_{(22)} \oplus r_{t+6}[15]_{(23)} \\ l_2(r_{t+7}) &= r_{t+7}[0]_{(12)} \oplus r_{t+7}[0]_{(13)} \oplus r_{t+7}[15]_{(22)} \oplus r_{t+7}[15]_{(23)} \\ l_2(r_{t+13}) &= r_{t+13}[0]_{(12)} \oplus r_{t+13}[0]_{(13)} \oplus r_{t+13}[15]_{(22)} \oplus r_{t+13}[15]_{(23)} \\ l_2(r_{t+14}) &= r_{t+14}[0]_{(12)} \oplus r_{t+14}[0]_{(13)} \oplus r_{t+14}[15]_{(22)} \oplus r_{t+14}[15]_{(23)} \\ l_2(r_{t+16}) &= r_{t+16}[0]_{(12)} \oplus r_{t+16}[0]_{(13)} \oplus r_{t+16}[15]_{(22)} \oplus r_{t+16}[15]_{(23)} \\ l_2(r_{t+17}) &= r_{t+17}[0]_{(12)} \oplus r_{t+17}[0]_{(13)} \oplus r_{t+17}[15]_{(22)} \oplus r_{t+17}[15]_{(23)} \end{array} \tag{37}$$

Since $r_{t+p}[0] = r_t[p]$, the above approximations can be presented as follows.

$$\begin{array}{ll} l_2(r_t) & = r_t[0]_{(12)} \oplus r_t[0]_{(13)} \oplus r_{t+15}[0]_{(22)} \oplus r_{t+15}[0]_{(23)} \\ l_2(r_{t+2}) & = r_t[2]_{(12)} \oplus r_t[2]_{(13)} \oplus r_{t+15}[2]_{(22)} \oplus r_{t+15}[2]_{(23)} \\ l_2(r_{t+6}) & = r_t[6]_{(12)} \oplus r_t[6]_{(13)} \oplus r_{t+15}[6]_{(22)} \oplus r_{t+15}[6]_{(23)} \\ l_2(r_{t+7}) & = r_t[7]_{(12)} \oplus r_t[7]_{(13)} \oplus r_{t+15}[7]_{(22)} \oplus r_{t+15}[7]_{(23)} \\ l_2(r_{t+13}) & = r_t[13]_{(12)} \oplus r_t[13]_{(13)} \oplus r_{t+15}[13]_{(22)} \oplus r_{t+15}[13]_{(23)} \\ l_2(r_{t+14}) & = r_t[14]_{(12)} \oplus r_t[14]_{(13)} \oplus r_{t+15}[14]_{(22)} \oplus r_{t+15}[14]_{(23)} \\ l_2(r_{t+16}) & = r_t[16]_{(12)} \oplus r_t[16]_{(13)} \oplus r_{t+15}[16]_{(22)} \oplus r_{t+15}[16]_{(23)} \\ l_2(r_{t+17}) & = r_t[17]_{(12)} \oplus r_t[17]_{(13)} \oplus r_{t+15}[17]_{(22)} \oplus r_{t+15}[17]_{(23)} \end{array} \tag{38}$$

If we combine (38) with (33), then we have the following approximation.

$$l_2(r_t) \oplus l_2(r_{t+2}) \oplus l_2(r_{t+6}) \oplus l_2(r_{t+7}) \oplus l_2(r_{t+13}) \oplus l_2(r_{t+14}) \oplus l_2(r_{t+16}) \oplus l_2(r_{t+17}) = \\ \nu_{t,(12)} \oplus \nu_{t,(13)} \oplus \nu_{t+1,(12)} \oplus \nu_{t+1,(13)} \oplus \nu_{t+15,(22)} \oplus \nu_{t+15,(23)} \oplus \nu_{t+16,(22)} \oplus \nu_{t+16,(23)} \tag{39}$$

By combining the approximations (36) and (39), we obtain the final approximation that defines our distinguisher, i.e.

$$\begin{array}{l} \quad l_1(r_t) \oplus l_1(r_{t+1}) \oplus l_1(r_{t+6}) \oplus l_1(r_{t+13}) \oplus l_1(r_{t+16}) \\ \quad \oplus l_2(r_t) \oplus l_2(r_{t+1}) \oplus l_2(r_{t+6}) \oplus l_2(r_{t+13}) \oplus l_2(r_{t+16}) \\ = \nu_{t+4,(0)} \oplus \nu_{t+5,(0)} \oplus \nu_{t+17,(0)} \oplus \nu_{t+18,(0)} \oplus \nu_{t,(12)} \oplus \nu_{t,(13)} \\ \quad \oplus \nu_{t+1,(12)} \oplus \nu_{t+1,(13)} \oplus \nu_{t+15,(22)} \oplus \nu_{t+15,(23)} \oplus \nu_{t+16,(22)} \oplus \nu_{t+16,(23)} \\ = 0 \end{array} \tag{40}$$

The second part of the approximation is observable to the adversary.

The bias of the distinguisher. We compute the bias of Approximation (40) with a similar way to Section 6.3. Let us denote ϵ_1 as the bias of Approximation (20) for NFSR, the bias of Approximation (33) for NLF at $i = 13$ and $i = 23$ by $\epsilon_{3,13}$ and $\epsilon_{3,23}$ respectively.

Then, the bias of the distinguisher ϵ can be calculated as follows.

$$\epsilon = 2^{-32} \sum_{k=0}^{2^{32}-1} (\epsilon_1^8 \cdot \epsilon_{3,13} \cdot \epsilon_{3,23} | Konst = k)$$

Experiment shows that the bias of Approximation (40) is around 2^{-48}.

When Stream Cipher Analysis Meets Public-Key Cryptography

Matthieu Finiasz and Serge Vaudenay

EPFL
CH-1015 Lausanne – Switzerland
http://lasecwww.epfl.ch/

Abstract. Inspired by fast correlation attacks on stream ciphers, we present a stream cipher-like construction for a public-key cryptosystem whose security relies on two problems: finding a low-weight multiple of a given polynomial and a Hidden Correlation problem. We obtain a weakly secure public-key cryptosystem we call TCHo (as for Trapdoor Cipher, Hardware Oriented). Using the Fujisaki-Okamoto construction, we can build an hybrid cryptosystem, TCHo^n-FO, resistant against adaptive chosen ciphertext attacks.

1 Introduction

Many of nowadays cryptosystems rely on the problem of factoring numbers. With the RSA [24] cryptosystem, the key recovery problem is equivalent to factoring a modulus into prime numbers. The message recovery problem is an ad-hoc problem, assumed to be hard, but potentially easier than factoring. To setup the cryptosystem we first generate secret prime numbers and then multiply them together to get the public key. Although the hardness of the factoring problem is an important open problem, it is known to be easy by using quantum computers [26].

With polynomials, the factoring problem is essentially easy. However, the problem of finding a multiple with low degree and weight is presumably hard. This problem occurs in correlation attacks on stream ciphers, and no polynomial time algorithm exists to solve it. Therefore, we can setup a new trapdoor system by first generating a secret low-weight polynomial and then looking for a suitable factor to produce a public key. We thus derive a new cryptosystem consisting simply of the XOR of two LFSRs with a random noise source. One LFSR is used to encode the data, the other contains the trapdoor. It is only used to hide the data, and the noise source provides non-linearity. The ciphertext is the result of this XOR. The key recovery problem is equivalent to finding a low-weight multiple of a given polynomial. The message recovery problem is an ad-hoc problem, potentially easier but still (assumed to be) hard: the Hidden Correlation (HC) problem. It further may remain a hard problem with quantum computers.

We think that analyzing this cryptosystem would be important in *any* case because this would either provide us with a secure post-quantum cryptosystem

© Springer-Verlag Berlin Heidelberg 2007

(if not an efficient pre-quantum one), or with new improvements for existing correlation attacks against stream ciphers. We aim at making the first step in this direction.

In this paper we review existing algorithms for finding low-weight multiples of a given polynomial and for the decoding of a noisy LFSR. This provides heuristics to select parameters for TCHo, a one-way public-key encryption scheme. It can encrypt small blocks of up to 30 bits into ciphertexts of a few thousand bits. TCHo^n encrypts n such blocks independently. The stream-cipher-like structure of our construction makes it most suitable for hardware implementation: an ASIC of a few thousand gates (equipped with a randomness generator) at 4MHz should be able to encrypt a block in less than 4ms.

We believe TCHo could also fit on a passive RFID tag. Providing public-key encryption to RFID tags leads to new opportunities to solve privacy/security issues in RFID protocols. Current protocols with readers connected to a single server [1,23] are based on symmetric cryptography and either compromise privacy or induce an important overhead complexity on the reader side. Public-key cryptography is the simplest (and most reliable) solution to solve these issues, however, no public-key encryption primitive can fit on an RFID tag. TCHo could possibly be the first construction to achieve this.

We have implemented the TCHo construction in C++ using `NTL` [27]. Key generation is pretty slow (about 4 minutes) as it requires to factorize polynomials with huge degrees. Encryption takes a fraction of a second in software, but this time is not significant for hardware implementation: LFSRs are not well suited for use with a CPU. Finally, our decryption implementation takes a few seconds.

Previous work. Quite a lot of work has been done trying to attack various stream ciphers like constructions. The first attacks were the (fast) correlation attack, invented by Siegenthaler [28] and improved by Meier and Staffelbach [20]. The main idea is to isolate one linear feedback shift-registers (LFSR) inside a stream cipher and approximate the rest of the construction by another large LFSR with some noise, then try to recover the initialization of the first LFSR and get a part of the key. Many improvements and applications to specific constructions have been made [4,5,13,14,15,18,21]. This work was inspired by a preliminary attack on Bluetooth E0 [17,19]. Some of the most efficient techniques are compared in Section 2.2.

A key problem in most of these techniques is the possibility to find low-weight multiples of a given polynomial. The best techniques rely either on exhaustive search or on decoding techniques [3,4,22,30]. Section 2.1 of this article is devoted to this problem.

More recently, algebraic attacks have been applied to break some constructions [6,7,8,10,11]. These attack can be very efficient but, even if some bounds have been proven [7,9], predicting their efficiency is quite a difficult problem. Throughout this article we will neglect this category of attacks as they are not well suited for large randomized constructions like ours.

The concept of trapdoor in a stream cipher was already brought up by Camion, Mihaljevic and Imai [2]. They showed that using specific retroaction polynomials in LFSRs, some bits of the output stream could depend on only some bits of the initialization, making it possible to speed up the recovery process for constructions which otherwise seemed perfectly sound. This can be well suited to kleptographic attack but it seems hard to build a public-key cryptosystem this way.

In Section 2, we define new computational problems and provide heuristics to solve them. This suggests some parameters ranges in which they can be reasonably assumed to be hard on average. Then we define a raw encryption scheme in Section 3 and prove that it is one-way under chosen plaintext attacks.

2 New Computational Problems

In this section, we formalize some computational problems which will be used and review existing algorithms to solve them. We focus on *exact complexities* as opposed to asymptotic ones. For this, we use two constants c_{easy} and c_{hard} to indicate that a complexity below $2^{c_{\mathsf{easy}}}$ operations is *essentially easy in practice* and efficient, and that a complexity over $2^{c_{\mathsf{hard}}}$ is *intractable in practice*. In practice, we can take $c_{\mathsf{easy}} = 30$ and $c_{\mathsf{hard}} = 80$.

Throughout this paper, we will define the bias of a source as $\text{bias}(\mathcal{S}) = P(\mathcal{S} = 0) - P(\mathcal{S} = 1)$. This bias is hence taken between -1 and 1 and is null when the source is unbiased. In our system, we will only consider positive biases, that is, sources which produce more 0's than 1's.

2.1 The Low-Weight Polynomial Multiple Problem

As opposed to integers, factoring polynomials is essentially easy. But when it comes to finding low-weight polynomial multiples, the problem becomes harder (at least, it is not known to be easy). Here, we will only focus on polynomials over $\mathbf{F}_2$.

Problem 1 (Low-Weight Polynomial Multiple). We consider the following problem on average over the random selection of an instance P, issued by a given instance generator Gen. We say that the problem is hard for Gen if solving it requires more than $2^{c_{\mathsf{hard}}}$ complexity on average. We consider two kinds of generator. Let Gen_1 be a generator which selects a random primitive polynomial P of degree d_P. Let Gen_2 be a generator which selects a random polynomial K of degree d_K and weight w until it has a primitive factor P, produced as the output, whose degree d_P is in a given interval $[d_{\min}, d_{\max}]$.

Parameters: a weight w, two degrees $d_P < d_K$
Instance: a binary polynomial P of degree d_P
Problem: find a multiple of P with degree at most d_K and weight at most w

The Gen_1 generator simulates which P we can meet when we do stream cipher cryptanalysis. The Gen_2 generator simulates which P we can meet when we construct a public-key cryptosystem.

Note that if P is irreducible, if x has order $n \leq d_K$ in the group $(\mathbf{F}_2[x]/P(x))^*$, and if $w \geq 2$, then $K = x^n + 1$ solves the problem. Taking P primitive ensures that the order $n = 2^{d_P} - 1$ is maximal. Also, for values of w greater than the Hamming weight of P, then P itself solves the problem. However, for d_K and w small, the problem is believed to be hard. There are several strategies to solve it. Heuristically, if P is generated by Gen_1, the average number of solutions with nonzero constant term is

$$\mathcal{N}_{\mathsf{sol}} = 2^{-d_P} \sum_{i=0}^{w-1} \binom{d_K}{i} \approx 2^{-d_P} \binom{d_K}{w-1}. \tag{1}$$

When using Gen_2, the average number of solutions becomes $1 + \mathcal{N}_{\mathsf{sol}}$.

Strategy 1 – The birthday paradox. This strategy consists in building two lists of polynomials which are sums respectively of 0 or 1 and of a polynomial with weight $\frac{w-1}{2}$ and null constant term, all reduced modulo P. Once this is done, one simply looks for collisions. The lists have a size of $L = \binom{d_K}{(w-1)/2}$ and the complexity is $\mathcal{O}\left(L(\log(L) + d_P)\right)$. This strategy is always faster than exhaustive search, but requires a lot of memory.

Strategy 2 – Wagner's generalized birthday paradox. When the number of solutions becomes large enough (of order $\mathcal{O}\left(2^{d_P/3}\right)$), techniques based on Wagner's generalized birthday paradox [30] can become more efficient. This algorithm is not fit for finding all possible polynomials (or the hidden one from Gen_2) but can find one solution among many. If there exists $a \geq 2$ such that $\binom{d_K}{(w-1)/2^a} \geq 2^{d_P/(a+1)}$, then one solution can be found with a complexity $\mathcal{O}\left(2^a 2^{d_P/(a+1)}\right)$.

For instance, when $d_K \geq 2^{d_P/(1+\log_2(w-1))}$, we can use $a = \log_2(w-1)$ and find a multiple within $\mathcal{O}\left((w-1)d_K\right)$. Clearly, such a low complexity cannot be reached for any $w \leq d_P$ when $d_K \leq 2^{d_P/(1+\log_2(d_P-1))}$.

Strategy 3 – Syndrome decoding. Solving the problem can be done using a syndrome decoding algorithm. Compute the matrix of all the $x^i \mod P(x)$ for i from 1 to d_K and then find a low-weight word in the preimages of 1 of this matrix. When a single solution exists, this has a cost of:

$$\mathcal{O}\left(\mathcal{P}oly\left(d_K\right)\left(\frac{d_K}{d_P}\right)^{w-1}\right), \tag{2}$$

where $\mathcal{P}oly\left(d_K\right)$ is a polynomial of degree 2 or 3 in d_K (see [16] for example). We neglect this polynomial part as improved algorithm like [3] can compensate it. This complexity holds when there is a unique multiple polynomial of degree d_K and weight $\leq w$. When there are more solutions this cost is approximately divided by $\mathcal{N}_{\mathsf{sol}}$.

Strategy 4 – Exhaustive search. When looking for multiples of degree just above d_P, an exhaustive search on Q such that $K = P \times Q$ can be faster. The complexity of finding all multiples is $\mathcal{O}\left(\mathcal{P}oly\left(d_K\right) 2^{d_K - d_P}\right)$.

The best algorithm for finding low-weight multiples depends on both the parameters and our objective (whether we want to find a single, many, or all solutions). When a single solution exists, the best choice is Strategy 3. This leads us to the following assumption

Assumption 2 (Low-Weight Polynomial Multiple). *When* $w \log_2 \frac{d_K}{d_P} \geq c_{\mathsf{hard}}$ *and* $\binom{d_K}{w-1} \leq 2^{d_P}$*, the low-weight polynomial multiple problem is hard on average for* Gen_1*. When* $w \log_2 \frac{d_K}{d_{\min}} \geq c_{\mathsf{hard}}$ *and* $\binom{d_K}{w-1} \leq 2^{d_{\min}}$*, the low-weight polynomial multiple problem is hard on average for* Gen_2*.*

2.2 The Noisy LFSR Decoding Problem

A binary linear code of length ℓ is a vector subspace of $\{0,1\}^\ell$. Elements are codewords. We consider the problem of decoding *noisy* strings, i.e. decoding the XOR of a codeword together with the output of a random source $\mathcal{S}_\gamma$ with bias γ. This source represents the error produced by a binary symmetric channel. In what follows we concentrate on codes which consist of all possible ℓ-bit strings which can be output from an LFSR $\mathcal{L}_P$ with a fixed retroaction polynomial P of degree d_P, i.e. sequences $Z = (z_1, \ldots, z_\ell)$ such that $(z_t, \ldots, z_{t+d_P}) \bullet P = 0$ for $t = 1, \ldots, \ell - d_P$ (where $\bullet$ denotes a scalar product, which means we consider P as a binary $(d_P + 1)$-tuple). We formalize the decoding problem of the noisy LFSR channel as follows.

Problem 3 (Noisy LFSR Decoding). We consider the following problem on average over the random selection of P, X, and the biased noise. We say that the problem is hard if getting a single bit of X (that is, decoding X with probability higher than 2^{1-d_P}) requires over $2^{c_{\mathsf{hard}}}$ complexity on average.

Parameters: a length ℓ, a polynomial P of degree d_P, a bias γ
Noisy LFSR channel: given a uniformly distributed random seed X of length d_P, generate Y, the XOR of the output of length ℓ of $\mathcal{L}_P$ initialized with X and a random noise generated by $\mathcal{S}_\gamma$
Problem: given Y, recover X

When γ is so close to 1 that errors are unlikely (e.g. $1-\gamma \ll \ell^{-1}$), the problem can easily be solved with high probability of success by Gaussian elimination. When γ is so small that we cannot even distinguish $\mathcal{S}_\gamma$ from an unbiased source (e.g. $\gamma \ll \ell^{-\frac{1}{2}}$), the problem is impossible to solve with relevant probability of success. In what follows we may assume that the channel transmits less data than its capacity[1] $C(\gamma) = 1 + \left(\frac{1}{2} + \frac{\gamma}{2}\right) \log_2 \left(\frac{1}{2} + \frac{\gamma}{2}\right) + \left(\frac{1}{2} - \frac{\gamma}{2}\right) \log_2 \left(\frac{1}{2} - \frac{\gamma}{2}\right)$, i.e. $\frac{d_P}{\ell} \leq C(\gamma)$. Thus, with unbounded computational power, decoding is possible. However, this is not always the case in only $\mathcal{O}\left(2^{c_{\mathsf{hard}}}\right)$ operations.

Three main classes of algorithms exist to solve this problem: those based on information set decoding, those trying to perform maximum likelihood (noted

[1] When γ is small, we have $C(\gamma) \approx \frac{\gamma^2}{2 \log 2}$ (with less than 1% error for $\gamma \leq \frac{1}{4}$).

ML hereafter) decoding, and those based on iterative decoding techniques. Note that by simply guessing the noise weight, the problem reduces to a permuted kernel problem [25].

Information Set Decoding. consists in picking d_P bits at random among the ℓ output bits and perform a Gaussian elimination on the corresponding columns of the generator matrix of the LFSR. Decoding succeeds if there are no error among the selected bits, namely with probability $\left(\frac{1}{2}+\frac{\gamma}{2}\right)^{d_P}$. By iterating this simple algorithm enough time to get the correct decoding we can decode within a complexity roughly $\left(\frac{1}{2}+\frac{\gamma}{2}\right)^{-d_P}$ (improved decoding algorithms like [3] make it possible to neglect the cost of the Gaussian elimination). When the bias γ is too small, namely for $\gamma \leq 2^{1-c_{\mathsf{hard}}/d_P}-1$, this requires over $2^{c_{\mathsf{hard}}}$ iterations.

The converse approach consisting in trying to guess the error bits could seem interesting when the level of noise is low (γ close to 1), but it is never more efficient than Information Set Decoding.

Maximum Likelihood Decoding. consists in trying to find the most likely X, given the ℓ-bit output stream we have. As the source $\mathcal{S}_\gamma$ is memoryless with a positive bias, the maximum likelihood corresponds to the X generating the closest (in terms of Hamming distance) codeword to Y.

This gives us a basic ML decoding algorithm: try all the possible initializations for $\mathcal{L}_P$ and sort them according to their distance to Y. This is however very costly (about $\mathcal{O}\left(2^{d_P}\ell\right)$) and can be slightly improved to $\mathcal{O}\left(2^{d_P}d_P\right)$ by using a Walsh transform as done in [19]. This method is successful if the solution is on top of the list. This is the case only if we try to transmit less data than the channel capacity. For $d_P \leq c_{\mathsf{easy}}$ and $\frac{1}{4} \geq \gamma \geq \sqrt{\frac{d_P}{\ell}2\log 2}$ (the $\frac{1}{4}$ is here to ensure that the approximation is valid), we can thus efficiently solve the problem. On the contrary, decoding can be impossible for two different reasons: either the noise is too high (that is, the bias is too small) and we sent more data than the channel capacity (for instance, if $\gamma \leq 1.18\,\ell^{-\frac{1}{2}}$, the channel cannot transmit more than one bit of information, which means Y cannot contain more than one bit of information on X), or the cost of the maximum likelihood decoding algorithm is too high.

First case – If we send more data than the capacity of the channel we will not be able to recover X completely, but we can still get some information on it. This would not be sufficient to solve a decoding problem, but this is already enough to threaten a cryptographic construction.

The difference between the distance of Y to any incorrect codeword and the distance to the correct one can be approximated to a normal law of expected value $-\gamma\frac{\ell}{2}$ and variance $\frac{\ell}{2}$ so the expected rank of the correct X in the maximum likelihood list is approximately $1+(2^{d_P}-1)\varphi\left(\frac{-\gamma}{2}\sqrt{\ell}\right)$ where

$$\varphi(x)=\frac{1}{\sqrt{2\pi}}\int_{-\infty}^{x}e^{-\frac{t^2}{2}}dt.$$

The amount of information one can get on X depends directly on this rank.

Second case – If the complexity of the ML decoding algorithm is too high, we can try too improve it. More subtle algorithms exist to recover the initialization of $\mathcal{L}_P$: they cannot decode as much noise as the basic algorithm but can have a significantly lower complexity. The basic idea is that instead of decoding in the full code it is possible to decode only in a subcode and then extend the decoding to the rest of the code. For example, one could write the generator matrix of the code defined by P and only consider columns ending with $d_P - k$ zeros: this way one would have to decode in a code of dimension k (instead of d_P) but with less bits, that is a length $\ell.2^{-(d_P-k)}$ on average. When γ is small, the right X tops the ML list if $\frac{k}{\ell.2^{-(d_P-k)}} \leq \frac{\gamma^2}{2\log 2}$. In general, it takes time $2^k\varphi\left(-\frac{\gamma}{2}\sqrt{\ell 2^{-(d_P-k)}}\right)$ to find the right k bits of X.

This improvement makes it possible to recover the initialization of an LFSR at a lower cost, provided enough output bits are available. This approach can still be improved: if too few output bits are available, we can compute new ones from those we have. If we write all the output bits which are the XOR of d different bits, starting from ℓ bits we can obtain $\binom{\ell}{d} \simeq \frac{\ell^d}{d!}$ bits instead of ℓ, but with a noise of bias γ^d instead of γ. This means that, for any value of d (the size of the combinations we consider) and k (the dimension of the subcode we obtain) we can decode if:

$$\frac{kd!}{\ell^d \cdot 2^{-(d_P-k)}} \leq \frac{\gamma^{2d}}{2\log 2}. \tag{3}$$

In general, even if the previous bound is not reached, the correct X has rank:

$$2^k\varphi\left(-\frac{\gamma^d}{2}\sqrt{\frac{\ell^d}{d!}2^{-(d_P-k)}}\right).$$

The complexity of the decoding is then the sum of the three following steps:

- Computing the combinations: $\mathcal{O}\left(\frac{\ell^d}{d!} \times k\right)$
- Decoding (with a Walsh Transform): $\mathcal{O}\left(k2^k\right)$.
- Finding the right codeword in the ML list: $\mathcal{O}\left(2^k\varphi\left(-\frac{\gamma^d}{2}\sqrt{\frac{\ell^d}{d!}2^{-(d_P-k)}}\right)\right)$

If one cannot pay more than $\mathcal{O}\left(c_{\mathsf{hard}}2^{c_{\mathsf{hard}}}\right)$ complexity, we must have $k \leq c_{\mathsf{hard}}$ and d roughly less than $c_{\mathsf{hard}}\frac{\log 2}{\log \ell}$ so that $\frac{\ell^d}{d!} \leq 2^{c_{\mathsf{hard}}}$. The rank in the ML list is $2^k\varphi(-t)$ where $t \leq \frac{\gamma^d}{2}2^{c_{\mathsf{hard}}-\frac{d_P}{2}}$. For $d_P \geq 2c_{\mathsf{hard}}$ we have $t \leq \frac{1}{2}$. So the rank is higher than $2^k\varphi(-\frac{1}{2}) \geq 0.3 \cdot 2^k$. So, this algorithm yields less than one bit of information of X.

Iterative Decoding. techniques are less efficient in terms of error correction, but can be applied to longer LFSRs. The idea is to find low-weight multiples of P which form some parity check equations, and use them to decode as in a Low Density Parity Check (LDPC) code.

As stated in [4], iterative decoding using parity check equations of weight $d \geq 4$ (that is multiples of P of weight d), succeeds if it is possible to find enough of these parity check equations for the iterative process to converge. Decoding is thus possible if:

$$\ell \geq 2^{\alpha_d(\gamma)+\frac{d_P}{d-1}} \quad \text{with} \quad \alpha_d(\gamma) = \tfrac{1}{d-1} \log_2 \left[(d-1)! \tfrac{1}{C(\gamma^{d-2})}\right]. \tag{4}$$

From this, we can see that whatever the parameters of the system, there exists a d which makes iterative decoding possible. This means that for the smallest d satisfying this equation, there should be about just enough parity check equations: finding them requires to find nearly all multiples of P of weight d and degree less than ℓ. Among the techniques described in Section 2.1, the Strategy 1 based on the birthday paradox is the most efficient. It has a cost:

$$\mathcal{C}_{\mathsf{parity}} = \binom{\ell-1}{\lceil \frac{d-1}{2} \rceil} \quad \Leftrightarrow \quad \log_2\left(\mathcal{C}_{\mathsf{parity}}\right) \simeq \tfrac{d-1}{2} \log_2 \ell - \log_2\left(\tfrac{d-1}{2}!\right). \tag{5}$$

The cost for the decoding is then negligible compared to this cost. This means that if one cannot pay more than $2^{c_{\mathsf{hard}}}$ to decode, one can only decode if $2^{c_{\mathsf{hard}}} \geq \mathcal{C}_{\mathsf{parity}}$. Mixing Equations (4) and (5) we see that one can decode only if:

$$c_{\mathsf{hard}} \geq \tfrac{1}{2}\left[\log_2 (d-1)! - \log_2 C(\gamma^{d-2}) + d_P\right] - \log_2\left(\tfrac{d-1}{2}!\right).$$

Roughly, we get the same constraint: if $d_P \geq 2c_{\mathsf{hard}}$, decoding is not possible.

Property 4 (Noisy LFSR Decoding). *The noisy LFSR decoding problem can efficiently be solved when $d_P \leq c_{\mathsf{easy}}$ and $\frac{1}{4} \geq \gamma \geq \sqrt{\frac{d_P}{\ell} 2 \log 2}$. For $\gamma \leq 1.18\,\ell^{-\frac{1}{2}}$, Y contains less than 1 bit of information of X, that is, the mutual information between X and Y is smaller than 1.*

Assumption 5 (Noisy LFSR Decoding). *The noisy LFSR decoding is hard on average when P is generated by Gen_1 (as specified in Problem 1) and when $d_P \geq 2c_{\mathsf{hard}}$ and $\gamma \leq 2^{1-c_{\mathsf{hard}}/d_P} - 1$. When $d_{\min} \geq 2c_{\mathsf{hard}}$, $\gamma \leq 2^{1-c_{\mathsf{hard}}/d_{\min}} - 1$, P is generated by Gen_2, and w and d_K are such that the Low-Weight Polynomial Multiple is hard, the noisy LFSR decoding is hard as well.*

2.3 The Hidden Correlation Problem

We combine the previous problems into Hidden Correlation (HC) problem.

Problem 6 (Hidden Correlation).

Parameters: a length ℓ, two relatively prime[2] polynomials P and Q of degree d_P and d_Q respectively, a bias γ

[2] When P and Q have a common factor, the decoding problem is ambiguous so we exclude those cases. As we will see later, we will always choose distinct primitive polynomials P and Q so they never have a common factor.

HC channel: given a uniformly distributed random seed X of length d_Q, generate Y, the XOR of the output of length ℓ of $\mathcal{L}_Q$ initialized with X, the output of $\mathcal{L}_P$ initialized at random, and a random noise generated by $\mathcal{S}_\gamma$ (both random sources being independent)
Problem: given Y, recover X

As for the noisy LFSR decoding problem, we say that the problem is hard if no algorithm that outputs the correct X with probability higher than $2^{1-c_{\mathsf{hard}}}$ and average complexity less than $2^{c_{\mathsf{hard}}}$ exists.

This problem is meant to be used given an oracle that solves the noisy LFSR decoding problem with parameters (ℓ', Q, γ') where $\ell' = \ell - d_K$ for some degree d_K and $\gamma' = \gamma^w$ for some weight w. There are three main strategies to solve it:

1. consider $\mathcal{L}_P \oplus \mathcal{L}_Q$ as a single LFSR $\mathcal{L}_{P\times Q}$, recover its initializations and deduce the initializations of both $\mathcal{L}_P$ and $\mathcal{L}_Q$ from it, thus recovering X. Achieving this requires to be able to solve the noisy LFSR decoding problem with parameters $(\ell, P \times Q, \gamma)$.
2. suppress the output of $\mathcal{L}_P$ in Y and decode X with a shorter output and a higher noise level. One should first find a polynomial K, multiple of P of degree d_K and weight w (note that choosing $K = P$ is also possible). Then multiply Y by K to suppress the influence of $\mathcal{L}_P$ and try to solve the noisy LFSR decoding problem with parameters $(\ell - d_K, Q, \gamma^w)$.
   As discussed in Section 2.1, the problem of finding K can be hard. The key idea of our construction is that this K can be a trapdoor.
3. suppress the output of $\mathcal{L}_Q$ in Y and recover the initialization of $\mathcal{L}_P$ (with a shorter output and a higher noise level). Once this is done, recovering X consists in decoding in $\mathcal{L}_Q \oplus \mathcal{S}_\gamma$ only, with the full output ℓ and the same bias γ: the oracle can do this. We can use the same method as above and find a multiple K of Q. In the end, we need to solve the noisy LFSR problem with parameters $(\ell - d_K, P, \gamma^w)$.

By taking P random of degree $d_P \geq 2c_{\mathsf{hard}}$ and $\gamma \leq 2^{1-c_{\mathsf{hard}}/d_P} - 1$ we know from Assumption 5 that the decoding problem for $\mathcal{L}_P \oplus \mathcal{S}_\gamma$ is intractable. This renders strategies 1 and 3 impossible.

To build a public-key cryptosystem on this problem, we need to find parameters which render strategy 2 computationally infeasible, and at the same time let us have a trapdoor polynomial K making it possible to solve it. We do this by proving an upper bound on the information one can get on X using strategy 2 in the favorable case where we can compute all multiples of low weight (except the hidden one), and bounded by the maximum number of iterations ($2^{c_{\mathsf{hard}}}$) we can manage. The number of possibles multiples of weight w and degree $d_K \leq \ell$ (except the hidden K) is given by equation (1).

For each multiple K' of P of weight i it is possible to suppress the influence of $\mathcal{L}_P$ and recover a little bit of information on X. This information I_i is bounded by $I_i \leq \ell C(\gamma^i)$.

By ignoring the cost of finding a multiple of weight i we upper bound the cost of recovering the I_i bits of information by $\mathcal{O}(\ell i)$, the cost of computing

$Y' = Y \bullet K'$. Then an adversary can use all possible multiples of a given weight, as long as there are less than $\frac{1}{\ell i} 2^{c_{\mathsf{hard}}}$ such multiples. For large values of i, when there are more multiples of P, he is limited to exactly $\frac{1}{\ell i} 2^{c_{\mathsf{hard}}}$ multiples. The total information one can get is at most:

$$\mathcal{I} = \sum_{i=2}^{\infty} \ell C(\gamma^i) \min\left(\frac{\binom{\ell}{i}}{2^{d_P}}, \frac{2^{c_{\mathsf{hard}}}}{\ell i} \right).$$

We can consider that this attack is not a threat if $\mathcal{I} \leq 1$ (which means that after using about $2^{c_{\mathsf{hard}}}$ polynomials, an adversary gets less than 1 bit of information on X). This bound is not tight since we neglected the cost of finding the multiples.

Assumption 7 (Hidden Correlation Problem). *When $d_P \geq 2c_{\mathsf{hard}}$, $\gamma \leq 2^{1-c_{\mathsf{hard}}/d_P} - 1$, and $\mathcal{I} \leq 1$, the Hidden Correlation problem is hard on average when P is generated by* Gen_1 *as specified in Problem 1. When $d_{\min} \geq 2c_{\mathsf{hard}}$, $\gamma \leq 2^{1-c_{\mathsf{hard}}/d_{\min}} - 1$, $\mathcal{I} \leq 1$ (with d_P replaced by $d_{\min}$), P is generated by* Gen_2*, and w and d_K are such that the Low-Weight Polynomial Multiple is hard, the hidden correlation problem is hard as well.*

3 TCHo Encryption

3.1 Specifications

This construction depends on a number of domain parameters. These are:

- a bias $\gamma \in [0, 1]$ (we only consider positive biases here),
- two integers $d_{\min}$ and $d_{\max}$ bounding the degree of P: $d_P \in [d_{\min}, d_{\max}]$.
- two integers d_Q and d_K: the degrees of polynomials Q and K,
- a primitive polynomial Q of degree d_Q
- an integer w corresponding to the weight of the polynomial K,
- an integer ℓ which is the length of the produced stream.

As we will see later in Table 1, suitable parameters might look like:

$$\gamma = 0.98, d_P \in [6\,000, 6\,600], d_Q = 20, d_K = 11\,560, w = 99, \ell = 13\,080.$$

Key generation. The public key is a primitive polynomial P. The private key is a low-weight multiple K of P.

To generate such a key pair one will first pick a random binary polynomial K coprime with Q of given weight w and degree d_K. Then one factors this polynomial and checks if it has a primitive factor of suitable degree $d_P \in [d_{\min}, d_{\max}]$. If such a factor exists one uses it as the public key P and a key pair was found, otherwise one picks another random polynomial K, factors it, etc. The complexity of this step is detailed in Appendix A. With the above parameters, this takes a little more than 4 minutes using `NTL` [27] on a 1.5GHz Pentium 4.

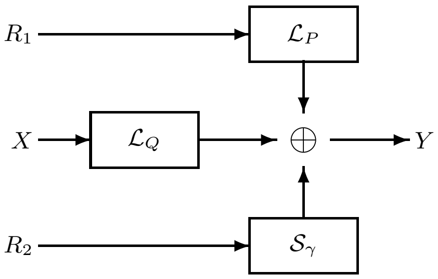


Fig. 1. Encryption

Encryption. The core of the system is a stream-cipher-like construction, built as the XOR of two LFSRs $\mathcal{L}_P$ and $\mathcal{L}_Q$ (with respective retroaction polynomials P and Q) and a biased random source $\mathcal{S}_\gamma$ where $0 \leq \gamma \leq 1$ represents the bias of this source.

The word X of d_Q bits to be encrypted is used to initialize $\mathcal{L}_Q$. $\mathcal{L}_P$ is randomly initialized with a string R_1. R_2 denotes the random coins for $\mathcal{S}_\gamma$. We assume that R_1 and R_2 are independent. The ciphertext is the binary stream Y of length ℓ equal to the XOR of the streams generated by $\mathcal{L}_P$, $\mathcal{L}_Q$ and $\mathcal{S}_\gamma$ (see Fig. 1). We denote $Y = \mathsf{TCHo}_P(X; R)$ for $R = (R_1, R_2)$. The cost of an encryption is $\mathcal{O}\left((d_P + d_Q + \varepsilon_\gamma) \times \ell\right)$, where ε_γ represents the cost of generating one random bit with bias γ. A dedicated hardware of $\mathcal{O}\left(d_P + d_Q + \varepsilon_\gamma\right)$ gates (typically, a few thousand gates) runs in time $\mathcal{O}\left(\ell\right)$ (typically, a few thousand clock cycles).

Decryption. To decrypt, one first uses the private key K to suppress the influence of $\mathcal{L}_P$. For $i = 1, \ldots, \ell - d_K$, compute $Y'_i = (Y_i, Y_{i+1}, \ldots, Y_{i+d_K}) \bullet K$. This results in a new cipher Y' of length $\ell - d_K$ equal to the XOR of a stream generated by $\mathcal{L}_Q$ and a source of bias γ^w. The complexity is $\mathcal{O}\left(d_K \cdot \ell\right)$.

From Property 4 we know that if $\ell - d_K \geq \frac{2 d_Q \log 2}{\gamma^{2w}}$ (and $\gamma^w \leq \frac{1}{4}$) we can recover the initialization of $\mathcal{L}_Q$ (thus X) using an ML decoding algorithm. Using a Walsh transform, this decryption costs $\mathcal{O}\left(d_Q\, 2^{d_Q}\right)$ operations. Finally, X is recovered by solving a linear equation in time $\mathcal{O}\left(d_K^3\right)$. Overall decryption complexity is thus $\mathcal{O}\left(d_K \ell + d_Q 2^{d_Q} + d_K^3\right)$. Decoding is feasible for $d_Q \leq c_{\mathsf{easy}}$. However, that decryption is non-deterministic: there is a probability that S_γ generates too much noise and that decryption returns an incorrect result. Bounds on this failure rate are hard to estimate as the linear code corresponding to a truncated LFSR is not as easy to study as for a full length LFSR. However, during all our tests, no decodings ever failed or returned the wrong plaintext. The bound used in Section 3.2 for parameter selection, taken from information theory, seems to be sufficient to have a negligible decryption failure rate.

Table 1. Some suitable parameters for TCHo

c_{hard}	d_Q	d_P	γ	w	d_K	ℓ	$\mathcal{I}$	key gen. cost[3]
80	20	500 – 600	0.4	6	6 200 000	8 000 000	$2^{-1.7}$	$\sim 2^{56.5}$
		1 000 – 1 100	0.74	13	78 350	148 000	$2^{-1.9}$	$\sim 2^{42.5}$
		3 000 – 3 500	0.93	35	17 100	21 600	2^{-17}	$\sim 2^{36.5}$
		6 000 – 6 600	0.98	99	11 560	13 080	$2^{-1.1}$	$\sim 2^{36}$
		6 000 – 6 060	0.98	99	10 620	12 140	2^{-4}	$\sim 2^{39}$
	30	1 000 – 1 500	0.7	11	232 000	339 000	2^{-7}	$\sim 2^{44.5}$
		6 000 – 6 600	0.979	93	12 150	14 150	$2^{-2.1}$	$\sim 2^{36}$
90	20	7 000 – 7 700	0.98	105	13 950	15 900	$2^{-2.1}$	$\sim 2^{36.5}$
	30	7 000 – 7 700	0.98	99	14 460	16 750	$2^{-0.1}$	$\sim 2^{36.5}$
128	20	10 000 – 11 000	0.977	108	25 050	29 300	$2^{-5.8}$	$\sim 2^{38.5}$
	30	10 000 – 11 000	0.977	102	26 300	31 100	$2^{-2.9}$	$\sim 2^{38.5}$

Encrypting larger blocks. To encrypt n blocks $X_1, \ldots, X_n$ we define TCHo^n, the concatenation of n independent instances of TCHo (thus able to encrypt $n \cdot c_{\mathsf{easy}}$ bits of data), by using independent random coins $R_1, \ldots, R_n$:

$$\mathsf{TCHo}^n_P(X_1, \ldots, X_n; R_1, \ldots, R_n) = \big[Y_1, \ldots, Y_n\big], \quad \text{where} \quad Y_i = \mathsf{TCHo}_P(X_i; R_i).$$

3.2 Parameters Selection

We review here all the constraints on the system parameters in order for a legitimate decryption to be possible and for an attack by a computationally bounded adversary (bounded by $\mathcal{O}(2^{c_{\mathsf{hard}}})$ operations) to be impossible. As constraints need to be satisfied for any $d_P \in [d_{\min}, d_{\max}]$, in the following equations, d_P was replaced either by $d_{\min}$ or by $d_{\max}$ depending on the type of inequality.

- legitimate decryption is possible if: $\ell - d_K \geq \frac{2 d_Q \log 2}{\gamma^{2w}}$, $\gamma^w \leq \frac{1}{4}$, and $d_Q \leq c_{\mathsf{easy}}$ (see Section 2.3).
- recovering K is impossible if the conditions of Assumption 2 are verified, that is: $w \log_2 \frac{d_K}{d_{\max}} > c_{\mathsf{hard}}$ and $\binom{d_K}{w-1} \leq 2^{d_{\min}}$.
- without K, decryption is equivalent to solving an instance of the HC problem. From Assumption 7 we know this is impossible if:
$$d_{\min} \geq 2c_{\mathsf{hard}},\ \gamma \leq 2^{1-\frac{c_{\mathsf{hard}}}{d_{\min}}} - 1, \text{ and } \mathcal{I} = \sum_{i=2}^{\infty} \ell C(\gamma^i) \min\left(\frac{\binom{\ell}{i}}{2^{d_{\min}}}, \frac{2^{c_{\mathsf{hard}}}}{\ell i}\right) \leq 1.$$

We are looking for parameters which satisfy all the inequalities stated above. One should first choose $d_Q \leq c_{\mathsf{easy}}$ and $d_{\min} \geq 2c_{\mathsf{hard}}$, and deduce the largest possible $\gamma = 2^{1-c_{\mathsf{hard}}/d_{\min}} - 1$. Then find a w for which $\ell = (\ell - d_K) + d_K \geq \frac{2 d_Q \log 2}{\gamma^{2w}} + d_{\max} 2^{\frac{c_{\mathsf{hard}}}{w}}$ is small enough and deduce the minimum d_K and ℓ. Once parameters

[3] This is the cost of generating a key when using an algorithm based on linear algebra. See Appendix A for details.

are obtained one can just check whether $\mathcal{I} \leq 1$ or not and maybe reduce γ accordingly. Table 1 gives a few sets of parameters satisfying these inequalities. As one can notice, no small value of d_P appear in the tables. This is mainly because parameters which would otherwise be suitable for all the constraints always lead to a value of $\mathcal{I}$ too large. This however is not a problem as the optimal values (in terms of encryption/decryption complexity and transmission rate) seem to appear for larger values of d_P.

3.3 Security

Key Recovery. The key generation algorithm we use corresponds exactly to the Gen_2 generator of Assumption 2. If the conditions of this assumption are respected, the key recovery problem is exactly the problem of solving a hard instance of the low-weight polynomial multiple problem. For suitable parameters, key recovery should thus be computationally impossible.

Message Recovery. requires to solve the hidden correlation problem. Solving this problem is hard in general, thus with suitable parameters the encryption can be seen as a one-way function. In order to use the Fujisaki-Okamoto construction we need to prove that this encryption is (t, ε)-secure in the sense of One-Way Encryption (OWE) and Γ-uniform (see Appendix B for definitions).

Theorem 8. *Under Assumptions 2, 5, and 7,* TCHo *is* $(2^{\mathsf{C_{hard}}}, \frac{2}{2^{d_Q}})$*-OWE-CPA-secure.*

Proof. We have chosen parameters such that, for any value of P and K, an adversary spending less than $\mathcal{O}(2^{\mathsf{C_{hard}}})$ time always gets less than 1 bit of information on the plaintext X using a chosen plaintext attack (CPA). Therefore, he always has a probability lower than $\frac{2}{2^{d_Q}}$ of guessing the correct X. □

Theorem 9. *The* TCHo *encryption scheme is* $\left(\frac{1}{2} + \frac{\gamma}{2}\right)^{\ell}$*-uniform.*

Proof. We need to upper bound the probability (on the random coins R_1 and R_2 introduced in Section 3.1) that a given plaintext is mapped to a given ciphertext. As we only consider positive biases, for a given initialization of $\mathcal{L}_P$ the most likely ciphertexts correspond to the random coins R_2 giving $\mathcal{S}_\gamma = 0$. This happens with probability $\left(\frac{1}{2} + \frac{\gamma}{2}\right)^{\ell}$. When taking the average on the possible initializations of $\mathcal{L}_P$ (on all the possible random coins R_1) this probability can only decrease. □

Semantic Security. Deciding whether Y encrypts X under TCHo_P or is random is equivalent to deciding whether a binary string is an output form a noisy LFSR channel with parameters ℓ, P, γ or is random. This may be a hard problem as well. This question may deserve further work.

Malleability. Obviously, from $Y = \mathsf{TCHo}_P\big(X; (R_1, R_2)\big)$, an adversary can forge a new ciphertext $Y' = \mathsf{TCHo}_P\big(X \oplus \delta; (R_1 \oplus \rho, R_2)\big)$ without even knowing X, R_1 or R_2. Hence, raw TCHo is clearly not OWE-CCA-secure, just like the raw RSA cryptosystem.

Extension to TCHo^n. Let $\mathsf{TCHo}^n_P(X_1,\dots,X_n;R_1,\dots,R_n) = \left[Y_1,\dots,Y_n\right]$. An adversary A knows $Y_1,\dots,Y_n$ and P and wants to recover $X_1,\dots,X_n$. We know that spending less than $2^{c_{\mathsf{hard}}}$ time, Y_i cannot give A more than 1 bit of information on X_i. The other Y_j cannot help as A can generate as many couples (X_j,Y_j) as he wants by himself (he knows P and nothing else is required for encryption) and this will not give him any information on X_i. As the X_i are independent, if A tries to guess the values of $X_1,\dots X_n$ simultaneously he will not have a better probability of success than if he tries to guess them one after the other: his probability of success will be less or equal to $\left(\frac{2}{2^{d_Q}}\right)^n$. The concatenation TCHo^n is thus $(2^{c_{\mathsf{hard}}},\frac{2^n}{2^{d_Q n}})$-OWE-CPA-secure.

Concerning Γ-uniformity, the results also extend well for TCHo^n. For the concatenation we have:

$$\Gamma(X_1,\dots,X_n,Y_1,\dots,Y_n) = \Pr\left[h \leftarrow_R \mathtt{COINS} : \forall i, Y_i = \mathcal{E}_{pk}(X_i;h)\right].$$

As before, the best probability will be obtained if $\mathcal{S}_\gamma = 0$ for each of the n independent encryptions. This happens with probability $\left(\frac{1}{2}+\frac{\gamma}{2}\right)^{\ell n}$ and thus TCHo^n is $\left(\frac{1}{2}+\frac{\gamma}{2}\right)^{\ell n}$-uniform. We deduce that we can build, in the random oracle model, an IND-CCA secure hybrid cryptosystem **TCHo^n-FO** based on TCHo^n and an FG-secure symmetric encryption, by using the Fujisaki-Okamoto construction [12].

4 Conclusion

We presented the first public-key cryptosystem which resembles a stream cipher. Stream ciphers are usually very efficient when it comes to low-cost hardware implementation, so implementation may be quite competitive on these platforms. Software implementations are pretty fast as well. One issue with our construction might be the need for a huge amount of random coins, but hardware entropy accumulators may supply them efficiently. The main drawback is currently the overhead size and the decryption complexity. Although quite reasonable, future work will decrease it. One option would consist in replacing $\mathcal{L}_Q$ by a better binary code. As suggested by Willi Meier, repetition codes seem to offer a very good decryption complexity and can be available for higher dimensions.

TCHo is currently a prototype. Our concept might still have to be improved before becoming a real product, but TCHo is definitely a first step in a new direction for public-key cryptography. We encourage analysis of TCHo since it will either demonstrate its security or lead to improvements on stream cipher attacks.

References

1. Avoine, G., Oechslin, P.: A scalable and provably secure hash-based RFID protocol. In: PerSec 2005 (2005)
2. Camion, P., Mihaljević, M.J., Imai, H.: Two alerts for design of certain stream ciphers: Trapped LFSR and weak resilient function over GF(q). In: Nyberg, K., Heys, H.M. (eds.) SAC 2002. LNCS, vol. 2595, pp. 196–213. Springer, Heidelberg (2003)

3. Canteaut, A., Chabaud, F.: A new algorithm for finding minimum-weight words in a linear code: Application to McEliece's cryptosystem and to narrow-sense BCH codes of length 511. IEEE Transactions on Information Theory 44(1), 367–378 (1998)
4. Canteaut, A., Trabbia, M.: Improved fast correlation attacks using parity check equations of weight 4 and 5. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 573–588. Springer, Heidelberg (2000)
5. Chepyshov, V.V., Johansson, T., Smeets, B.: A simple algorithm for fast correlation attacks on stream ciphers. In: Schneier, B. (ed.) FSE 2000. LNCS, vol. 1978, pp. 181–195. Springer, Heidelberg (2001)
6. Courtois, N., Klimov, A., Patarin, J., Shamir, A.: Efficient algorithms for solving overdefined systems of multivariate polynomial equations. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 392–407. Springer, Heidelberg (2000)
7. Courtois, N., Meier, W.: Algebraic attacks on stream ciphers with linear feedback. In: Biham, E. (ed.) Advances in Cryptology – EUROCRPYT 2003. LNCS, vol. 2656, pp. 345–359. Springer, Heidelberg (2003)
8. Courtois, N., Pieprzyk, J.: Cryptanalysis of block ciphers with overdefined systems of equations. In: Zheng, Y. (ed.) ASIACRYPT 2002. LNCS, vol. 2501, pp. 267–287. Springer, Heidelberg (2002)
9. Dalai, D.K., Gupta, K.C., Maitra, S.: Results on algebraic immunity for cryptographically significant boolean functions. In: Canteaut, A., Viswanathan, K. (eds.) INDOCRYPT 2004. LNCS, vol. 3348, pp. 92–106. Springer, Heidelberg (2004)
10. Faugère, J.-C.: A new efficient algorithm for computing gröbner bases (F4). Journal of Pure and Applied Algebra 139, 61–88 (1999)
11. Faugère, J.-C.: A new efficient algorithm for computing gröbner bases without reduction to zero (F5). In: ISSAC 2002, Lille, France, July 2002, pp. 75–83. ACM, New York (2002)
12. 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)
13. Johansson, T., Jönsson, F.: Fast correlation attacks based on turbo code techniques. In: Wiener, M.J. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 181–197. Springer, Heidelberg (1999)
14. Johansson, T., Jönsson, F.: Improved fast correlation attacks on stream ciphers via convolutional codes. In: Stern, J. (ed.) EUROCRYPT 1999. LNCS, vol. 1592, pp. 347–362. Springer, Heidelberg (1999)
15. Johansson, T., Jönsson, F.: Fast correlation attacks through reconstruction of linear polynomials. In: Bellare, M. (ed.) CRYPTO 2000. LNCS, vol. 1880, pp. 300–315. Springer, Heidelberg (2000)
16. Lee, P.J., Brickell, E.F.: An observation on the security of McEliece's public-key cryptosystem. In: Günther, C.G. (ed.) EUROCRYPT 1988. LNCS, vol. 330, pp. 275–280. Springer, Heidelberg (1988)
17. Lu, Y.: Applied Stream Ciphers in Mobile Communications. Phd thesis num. 3491, EPFL (2006), http://library.epfl.ch/theses/?nr=3491
18. Lu, Y., Meier, W., Vaudenay, S.: The conditional correlation attack: A practical attack on bluetooth encryption. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 97–117. Springer, Heidelberg (2005)
19. Lu, Y., Vaudenay, S.: Faster correlation attack on bluetooth keystream generator E0. In: Franklin, M. (ed.) CRYPTO 2004. LNCS, vol. 3152, pp. 407–425. Springer, Heidelberg (2004)

20. Meier, W., Staffelbach, O.: Fast correltaion attacks on stream ciphers. In: Günther, C.G. (ed.) EUROCRYPT 1988. LNCS, vol. 330, pp. 301–314. Springer, Heidelberg (1988)
21. Mihaljevic, M.J., Fossorier, M.P.C., Imai, H.: A low-complexity and high-performance algorithm for the fast correlation attack. In: Schneier, B. (ed.) FSE 2000. LNCS, vol. 1978, pp. 196–212. Springer, Heidelberg (2001)
22. Molland, H., Mathiassen, J.E., Helleseth, T.: Improved fast correlation attack using low rate codes. In: Paterson, K.G. (ed.) Cryptography and Coding. LNCS, vol. 2898, pp. 67–81. Springer, Heidelberg (2003)
23. Molnar, D., Wagner, D.: Privacy and security in library RFID: issues, practices, and architectures. In: Atluri, V., Pfitzmann, B., McDaniel, P.D. (eds.) CCS 2004, pp. 210–219. ACM Press, New York (2004)
24. Rivest, R.L, Shamir, A., Adleman, L.M.: A method for obtaining digital signatures and public-key cryptosystems. Communications of the ACM 21(2), 120–126 (1978)
25. Shamir, A.: An efficient identification scheme based on permuted kernels (extended abstract). In: Brassard, G. (ed.) CRYPTO 1989. LNCS, vol. 435, pp. 606–609. Springer, Heidelberg (1990)
26. Shor, P.W.: Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. SIAM J. Comput. 26(5), 1484–1509 (1997)
27. Shoup, V.: NTL: A library for doing number theory. Available online from `http://www.shoup.net/ntl/`
28. Siegenthaler, T.: Cryptanalysts representation of nonlinearly filtered ML-sequences. In: Pichler, F. (ed.) EUROCRYPT 1985. LNCS, vol. 219, pp. 103–110. Springer, Heidelberg (1986)
29. von zur Gathen, J., Gerhard, J.: Modern Computer Algebra, 2nd edn. Cambridge University Press, Cambridge (2003)
30. Wagner, D.: A generalized birthday problem. In: Yung, M. (ed.) CRYPTO 2002. LNCS, vol. 2442, pp. 288–304. Springer, Heidelberg (2002)

A Key Generation

Average number of trials. The cost of a key generation depends directly on the number of trials required before finding a suitable key pair. Here we try to evaluate the probability that a random low-weight binary polynomial has a primitive factor of degree d_P.

It is known that, asymptotically, the number of irreducible binary polynomials of degree d is equivalent to $\frac{2^d}{d}$. If we call P one of these irreducible polynomials, a random binary polynomial K of degree $d_K > d$ is divisible by P (and has P in his factorization) if $K \equiv 0 \mod P$. The probability this happens is 2^{-d}. Taking into account this probability and the number of irreducible polynomials, we obtain the probability that K has an irreducible factor of degree d is approximately $\frac{1}{d}$. Given that P is irreducible, the probability that it is primitive is the probability that the logarithm of x in $(\mathbf{F}_2[x]/P(x))^*$ is coprime with $2^d - 1$. The probability that two random numbers are coprime relates to Buffon's needle problem and is $\frac{6}{\pi^2}$. Given that $2^d - 1$ is odd for sure, the probability that P is primitive given that it is irreducible can be heuristically approximated to $\frac{8}{\pi^2} \approx 81\%$.

This is true asymptotically for random polynomials. In our case we only consider polynomials K of low weight. For this particular case we have no proof that the same result still holds[4]. However, experimental statistics tend to prove that this approximation is still very accurate for low-weight polynomials. The average number of trials is thus approximately $\frac{\pi^2}{8}d_P$.

Factoring algorithm. Concerning the algorithm to be used for the factorization, the choice is very wide. The straightforward method would be to use a generic factoring algorithm like Berlekamp's or Cantor-Zassenhaus (see Chapter 14 of [29] for more details on polynomial factorization). However, when dealing with high degree polynomials, methods based on linear algebra tend to use a lot of memory and degrees above 2^{16} are too high for standard computers. Moreover, in our case, a complete factorization is not required: it is enough to check whether our polynomial has an irreducible factor of degree d_P, and this can be done quite efficiently.

The polynomial $X^{2^d}+X$ is the product of all binary irreducible polynomials whose degrees divide d. Checking if K has a factor of degree d_P reduces to the computation of $\gcd(X^{2^{d_P}}+X, K)$. If this gcd has a degree less than d_P one can be sure that K has no suitable factors, otherwise one simply needs to factor this "low degree" polynomial using whatever algorithm.

Computing this gcd is much faster than a complete factorization for high degree K and a small d_P. The computation of $X^{2^{d_P}}+X \mod K$ using successive squarings can however be quite long when d_P increases, thus, the best solution depends on the system parameters. Using Berlekamp's factoring algorithm the complexity of the factoring is $\mathcal{O}\left({d_K}^{2.4}\right)$ (where 2.4 is the cost for linear algebra) and using the gcd-based algorithm it is $\mathcal{O}\left(d_P\, {d_K}^2\right)$. Taking into account the number of iterations for a key generation we obtain $\mathcal{O}\left(d_P\, {d_K}^{2.4}\right)$ and $\mathcal{O}\left({d_P}^2\, {d_K}^2\right)$. Cantor-Zassenhaus gives $\mathcal{O}\left(d_P d_K^2 \log d_K \log\log d_K\right)$, which is the best complexity, but with a very large constant term.

Using degree ranges. The number of attempts required for the key generation to find a polynomial with degree $d_P \in [d_{\min}, d_{\max}]$ is divided by $\delta = d_{\max} - d_{\min} + 1$. If we use a general factoring algorithm, this directly divides the key generation complexity by δ but if we use the gcd method, the time used to compute $X^{2^{d_P}}+X \mod K$ does not change and we have to compute δ gcds. In both cases this improves the key generation complexity, but for large values of δ, the gcd method becomes very slow. Using Berlekamp's algorithm the complexity becomes $\mathcal{O}\left(\frac{1}{\delta}\, d_P\, {d_K}^{2.4}\right)$, with Cantor-Zassenhaus it is $\mathcal{O}\left(\frac{1}{\delta}\, d_P\, {d_K}^2 \log d_K \log\log d_K\right)$, and for the gcd-based technique it becomes $\mathcal{O}\left(\frac{d_P}{\delta}(d_P+\delta){d_K}^2\right)$.

For small values of δ the best choice is the gcd-based technique, but for larger δ generic factoring algorithms are faster. If d_K is large, Cantor-Zassenhaus is the best algorithm.

[4] In our construction we only considers polynomials K with a non null constant term, which are therefore not divisible by X. If one chooses them with an odd weight they will never be divisible by $X+1$ and will never have an irreducible factor of degree 1. Some similar properties might also hold for other degrees!

Primitivity testing. In general, checking whether an arbitrary polynomial is primitive is hard. However, we only need here to probabilistically filter out random polynomials which are not primitive. This is essentially easy. A probabilistic method for finding a generator of $\mathbf{Z}_p^*$ can be adapted to checking whether a polynomial is primitive. One simply need to find all divisors p_i of $2^{d_P} - 1$ smaller than a given λ, and check if the order of X in $\left(GF(2)[X]/P(X)\right)^*$ divides $\frac{2^{d_P}-1}{p_i}$. If this is the case for none of the p_i, P is primitive with probability $1 - \lambda^{-1}$.

So, with complexity $\mathcal{O}\left(\mathcal{P}oly\,(d_P)\,\sqrt{\lambda}\right)$ it is possible to reduce the risk that his algorithm accepts a non-primitive polynomial to $\mathcal{O}\left(\lambda^{-1}\right)$. It is thus possible to reach error rates as low as 2^{-40} with a cost negligible compared to the factorization step.

B Definitions

We report here definitions taken from the article by Fujisaki and Okamoto [12].

Definition 10 (Asymmetric Encryption). *An asymmetric encryption scheme Π is a triple of algorithms $\mathcal{K}$, $\mathcal{E}$ and $\mathcal{D}$, associated with two finite sets,* $\texttt{COINS}(k)$ *(a set of random coins) and* $\texttt{MSPC}(k)$ *(a message space), for $k \in \mathbb{N}$.*

- *$\mathcal{K}$, the key generation algorithm, is a probabilistic algorithm which on input 1^k $(k \in \mathbb{N})$ outputs a pair of keys, $(pk, sk) \leftarrow \mathcal{K}(1^k)$.*
- *$\mathcal{E}$, the encryption algorithm, is a probabilistic algorithm that takes a key pk, an message $x \in \texttt{MSPC}$ and a string $r \leftarrow_R \texttt{COINS}(k)$, where $\leftarrow_R$ represents a random affectation, and produces a ciphertext $y = \mathcal{E}_{pk}(x; r)$.*
- *$\mathcal{D}$, the decryption algorithm, is a deterministic algorithm that takes a key sk and a ciphertext y and returns a message $x \leftarrow \mathcal{D}_{sk}(y)$.*

It is required that, for any $k \in \mathbb{N}$, if $(pk, sk) \leftarrow \mathcal{K}(1^k)$, $x \in \texttt{MSPC}$, and $y \leftarrow \mathcal{E}_{pk}(x)$, then $\mathcal{D}_{sk}(y) = x$.

Definition 11 (OWE). *Let $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D}, \texttt{COINS}, \texttt{MSPC})$ be an asymmetric encryption scheme. Let A be an adversary knowing y and pk and trying to recover x. We say that Π is (t, ε)-OWE-secure if, for any A running in at most time t and for any $x \in \texttt{MSPC}$:*

$$\Pr\left[(pk, sk) \leftarrow \mathcal{K}(1^k); y \leftarrow \mathcal{E}_{pk}(x) : A(pk, y) = \mathcal{D}_{sk}(y)\right] \leq \varepsilon.$$

In other words, the probability among all the possible key pairs for A of recovering x from its encrypted value is bounded by ε.

Definition 12 (Γ-uniformity). *Let $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D}, \texttt{COINS}, \texttt{MSPC})$ be an asymmetric encryption scheme. For given $(pk, sk) \leftarrow \mathcal{K}(1^k)$, $x \in \texttt{MSPC}$ and $y \in \{0,1\}^*$, define*

$$\Gamma(x, y) = \Pr\left[h \leftarrow_R \texttt{COINS} : y = \mathcal{E}_{pk}(x; h)\right].$$

We say that Π is Γ-uniform, if, for any $(pk, sk) \leftarrow \mathcal{K}(1^k)$, any $x \in \texttt{MSPC}$ and any $y \in \{0,1\}^*$, $\Gamma(x, y) \leq \Gamma$.

Definition 13 (Find-Guess-security). *Let $\Pi = (\mathcal{E}, \mathcal{D}, \mathtt{KSPC}, \mathtt{MSPC})$ be a symmetric encryption scheme where* KSPC *is the key space and* MSPC *is the message space. Let A be an adversary. We say Π is (t, ε)-FG-secure if, for any A running in at most time t:*

$$\Big| 2 \cdot \Pr\Big[a \leftarrow_R \mathtt{KSPC}; (x_0, x_1) \leftarrow A; b \leftarrow_R \{0,1\}; \\ y = \mathcal{E}_a(x_b) : A(x_0, x_1, y) = b \Big] - 1 \Big| \leq \varepsilon.$$

This means that an adversary choosing x_0 and x_1 cannot decide which of them corresponds to a given ciphertext $y = \mathcal{E}_a(x_b)$ with probability more than $\frac{1}{2} + \frac{\varepsilon}{2}$.

Definition 14 (IND-CCA-security). *Let $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D}, \mathtt{COINS}, \mathtt{MSPC})$ be an asymmetric encryption scheme and let A be an adversary having access to a decryption oracle $\mathcal{D}_{sk}$. We say that Π is (t, q_D, ε)-IND-CCA-secure if for any A running in at most time t and asking at most q_D queries to the decryption oracle:*

$$\Big| 2 \cdot \Pr\Big[(pk, sk) \leftarrow \mathcal{K}(1^k); (x_0, x_1) \leftarrow A^{\mathcal{D}_{sk}}; b \leftarrow_R \{0,1\}; \\ y = \mathcal{E}_{pk}(x_b) : A^{\mathcal{D}_{sk}}(x_0, x_1, y) = b \Big] - 1 \Big| \leq \varepsilon.$$

This means that an adversary choosing x_0 and x_1 cannot decide which of them corresponds to a given ciphertext $y = \mathcal{E}_{pk}(x_b)$ with probability more than $\frac{1}{2} + \frac{\varepsilon}{2}$. In the random oracle model, $\mathcal{E}_{sk}$ might also use some random oracles for encryption. We then get a similar definition of IND-CCA security by bounding the number of queries to these random oracles.

On Redundant τ-Adic Expansions and Non-adjacent Digit Sets

Roberto Maria Avanzi[1,⋆], Clemens Heuberger[2,⋆⋆], and Helmut Prodinger[3,⋆⋆⋆]

[1] Faculty of Mathematics and Horst Görtz Institute for IT Security
Ruhr-University Bochum, Germany
`roberto.avanzi AT ruhr-uni-bochum.de`

[2] Institut für Mathematik B, Technische Universität Graz, Austria
`clemens.heuberger AT tugraz.at`

[3] Department of Mathematics, University of Stellenbosch, South Africa
`hproding AT sun.ac.za`

Abstract. This paper studies τ-adic expansions of scalars, which are important in the design of scalar multiplication algorithms on Koblitz Curves, and are less understood than their binary counterparts.

At Crypto '97 Solinas introduced the width-w τ-adic non-adjacent form for use with Koblitz curves. It is an expansion of integers $z = \sum_{i=0}^{\ell} z_i\tau^i$, where τ is a quadratic integer depending on the curve, such that $z_i \neq 0$ implies $z_{w+i-1} = \ldots = z_{i+1} = 0$, like the sliding window binary recodings of integers. We show that the digit sets described by Solinas, formed by elements of minimal norm in their residue classes, are uniquely determined. However, unlike for binary representations, syntactic constraints do not necessarily imply minimality of weight.

Digit sets that permit recoding of all inputs are characterized, thus extending the line of research begun by Muir and Stinson at SAC 2003 to Koblitz Curves.

Two new useful digit sets are introduced: one set makes precomputations easier, the second set is suitable for low-memory applications, generalising an approach started by Avanzi, Ciet, and Sica at PKC 2004 and continued by several authors since. Results by Solinas, and by Blake, Murty, and Xu are generalized.

Termination, optimality, and cryptographic applications are considered. We show how to perform a "windowed" scalar multiplication on Koblitz curves without doing precomputations first, thus reducing memory storage dependent on the base point to just one point.

⋆ Supported by the European Commission through the IST Programme under Contract IST-2002-507932 ECRYPT.

⋆⋆ Supported by the Austrian Science Foundation FWF, project S9606, that is part of the Austrian National Research Network "Analytic Combinatorics and Probabilistic Number Theory."

⋆⋆⋆ Supported by the NRF grant 2053748 of the South African National Research Foundation and by the Center of Experimental Mathematics of the University of Stellenbosch.

© Springer-Verlag Berlin Heidelberg 2007

1 Introduction

Elliptic curves (EC) [15,17] are now a well established cryptographic primitive. The performance of an EC cryptosystem depends on the efficiency of the fundamental operation, the *scalar multiplication*, i.e. the computation of the multiple sP of a point P by an integer s. Among all EC, *Koblitz curves* [16], defined by the equation

$$E_a\colon y^2 + xy = x^3 + ax^2 + 1 \qquad \text{with} \qquad a \in \{0, 1\} \tag{1}$$

over the finite field $\mathbb{F}_{2^n}$, permit particularly efficient implementation of scalar multiplication. Key to their good performance is the Frobenius endomorphism τ, i.e. the map induced on $E_a(\mathbb{F}_{2^n})$ by the Frobenius automorphism of the field extension $\mathbb{F}_{2^n}/\mathbb{F}_2$, that maps field elements to their squares.

Set $\mu = (-1)^{1-a}$. It is known [24, Section 4.1] that τ permutes the points $P \in E_a(\mathbb{F}_{2^n})$, and $(\tau^2 + 2)P = \mu\tau(P)$. Identify τ with a root of

$$\tau^2 - \mu\tau + 2 = 0 \ . \tag{2}$$

If we write an integer z as $\sum_{i=0}^{\ell} z_i\tau^i$, where the digits z_i belong to a suitably defined digit set $\mathcal{D}$, then we can compute zP as $\sum_{i=0}^{\ell} z_i\tau^i(P)$ via a Horner scheme. The resulting method [16,23,24] is called a "τ-and-add" method since it replaces the doubling with a Frobenius operation in the classic double-and-add scalar multiplication algorithm. Since a Frobenius operation is much faster than a group doubling, scalar multiplication on Koblitz curves is a very fast operation.

The elements dP for all $d \in \mathcal{D}$ are computed before the Horner scheme. Larger digit sets usually correspond to representations $\sum_{i=0}^{\ell} z_i\tau^i$ with fewer non-zero coefficients i.e. to Horner schemes with fewer group additions. Optimal performance is attained upon balancing digit set size and number of non-zero coefficients.

Solinas [23,24] considers the residue classes in $\mathbb{Z}[\tau]$ modulo τ^w which are coprime to τ, and forms a digit set comprising the zero and an element of minimal norm from each residue class coprime to τ. We prove (Theorem 2) that such elements are unique, hence this digit set is uniquely determined. It has cardinality $1 + 2^{w-1}$. Solinas' recoding enjoys the *width-w non-adjacent property*

$$z_i \neq 0 \qquad \text{implies} \qquad z_{w+i-1} = \ldots = z_{i+1} = 0 \ , \tag{3}$$

and is called the τ-adic width-w non-adjacent form (or τ-w-NAF for short). Every integer admits a unique τ-w-NAF.

We call a digit set allowing to recode all integers satisfying property (3) a *(width-w) non-adjacent digit set*, or w-NADS for short. Theorem 1 is a criterion for establishing whether a given digit set is a w-NADS, which is very different in substance from the criterion of Blake, Murty, and Xu [8]. The characterisation of digit sets which allow recoding with a non-adjacency condition is a line of research started by Muir and Stinson in [18] and continued, for example by Heuberger and Prodinger in [11].

Our criterion is applied to digit sets introduced and studied in §§ 2.3 and 2.4. We can prove under which conditions the first set is a w-NADS (Theorem 3), and give precise estimates of the length of the recoding (Theorem 4). The second digit set corresponds, in a suitable sense, to "repeated point halvings" (cf. Theorem 5) and is used to design a width-w scalar multiplication algorithm without precomputations. Among the other results in Section 2 are the facts that the τ-adic w-NAF as defined by Solinas is not optimal, and that it is not possible to compute minimal expansions by a deterministic finite automaton. In Section 3 we discuss the relevance of our results for cryptographic applications and performance. We conclude in Section 4. Due to space constraints, most proofs have been omitted. They will be given in the extended version of the paper.

2 Digit Sets

Let $\mu \in \{\pm 1\}$, τ be a root of equation (2) and $\bar{\tau}$ the complex conjugate of τ. Note that $2/\tau = \bar{\tau} = \mu - \tau = -\mu(1+\tau^2)$. We consider expansions to the base of τ of integers in $\mathbb{Z}[\tau]$. It is well known that $\mathbb{Z}[\tau]$, which is the ring of algebraic integers of $\mathbb{Q}(\sqrt{-7})$, is a unique factorization domain.

Definition 1. *Let $\mathcal{D}$ be a (finite) subset of $\mathbb{Z}[\tau]$ containing 0 and $w \geqslant 1$ be an integer. A $\mathcal{D}$-*expansion *of $z \in \mathbb{Z}[\tau]$ is a sequence $\boldsymbol{\varepsilon} = (\varepsilon_j)_{j\geqslant 0} \in \mathcal{D}^{\mathbb{N}_0}$ such that*

1. *Only a finite number of the digits ε_j is nonzero.*
2. *$\mathsf{value}(\boldsymbol{\varepsilon}) := \sum_{j\geqslant 0} \varepsilon_j \tau^j = z$, i.e., $\boldsymbol{\varepsilon}$ is indeed an expansion of z.*

The Hamming weight *of $\boldsymbol{\varepsilon}$ is the number of nonzero digits ε_j. The* length *of $\boldsymbol{\varepsilon}$ is defined as*

$$\mathsf{length}(\boldsymbol{\varepsilon}) := 1 + \max\{j : \varepsilon_j \neq 0\} \ .$$

*A $\mathcal{D}$-expansion of z is a $\mathcal{D}$-w-*Non-Adjacent-Form ($\mathcal{D}$-w-NAF) of *z, if*

3. *Each block $(\varepsilon_{j+w-1}, \dots, \varepsilon_j)$ of w consecutive digits contains at most one nonzero digit ε_k, $j \leqslant k \leqslant j+w-1$.*

A $\{0, \pm 1\}$-2-NAF is also called a τ-NAF.

The set $\mathcal{D}$ is called a w-Non-Adjacent-Digit-Set (w-NADS)*, if each $z \in \mathbb{Z}[\tau]$ has a $\mathcal{D}$-w-NAF.*

Typically, $\mathcal{D}$ will have cardinality $1 + 2^{w-1}$, but we do not require this in the definition. One of our aims is to investigate which $\mathcal{D}$ are w-NADS, and we shall usually restrict ourselves to digit sets formed by adjoining the 0 to a reduced residue system modulo τ^w, which is defined as usual:

Definition 2. *Let $w \geqslant 1$ a natural number. A reduced residue system $\mathcal{D}'$ for the number ring $\mathbb{Z}[\tau]$ modulo τ^w is a set of representatives for the congruence classes of $\mathbb{Z}[\tau]$ modulo τ^w that are coprime to τ.*

For a digit set $\mathcal{D}$ for $\mathbb{Z}[\tau]$ formed by 0 together with a reduced residue system, Algorithm 1 either recodes an integer $z \in \mathbb{Z}[\tau]$ to the base of τ, or enters in a infinite loop for some inputs when $\mathcal{D}$ is not a NADS.

Algorithm 1. General windowed integer recoding

INPUT: An element z from $\mathbb{Z}[\tau]$, a natural number $w \geqslant 1$ and a reduced residue system $\mathcal{D}'$ for the number ring R modulo τ^w.
OUTPUT: A representation $z = \sum_{j=0}^{\ell-1} z_j \tau^j$ of length ℓ of the integer z with the property that if $z_j \neq 0$ then $z_{j+i} = 0$ for $1 \leqslant i < w$.

```
1. $j \leftarrow 0, u \leftarrow z$
2. while $u \neq 0$ do
3.     if $\tau \mid u$ then
4.         $z_j \leftarrow 0$                                   [Output 0]
5.     else
6.         Let $z_j \in \mathcal{D}'$ s.t. $z_j \equiv z \pmod{\tau^w}$    [Output $z_j$]
7.     $u \leftarrow u - z_j,\ u \leftarrow u/\tau,\ j \leftarrow j+1$
8. $\ell \leftarrow j$
9. return $(\{z_j\}_{j=0}^{\ell-1}, \ell)$
```

Example 1. A digit set obtained by adjoining the zero to a reduced residue system is not necessarily a NADS. This fact has been observed in the binary case in [18]. If we take $w = 1$ and the digit set $\{0, 1-\tau\}$ (here the reduced residue set modulo $\tau = \tau^1$ comprises the single element $1-\tau$) we see that the element 1 has an expansion $(1-\tau) + (1-\tau)\tau + (1-\tau)\tau^2 + (1-\tau)\tau^3 + \cdots$. Algorithm 1 does not terminate in this case.

2.1 Algorithmic Characterization

As already mentioned, one aim of this paper is to investigate which digit sets $\mathcal{D}$ are in fact w-NADS. For concrete $\mathcal{D}$ and w, this question can be decided algorithmically:

Theorem 1. *Let $\mathcal{D}$ be a finite subset of $\mathbb{Z}[\tau]$ containing 0 and $w \geqslant 1$ be an integer. Let*

$$M := \left\lfloor \frac{\max\{N(d) : d \in \mathcal{D}\}}{(2^{w/2}-1)^2} \right\rfloor ,$$

where $N(z)$ denotes the norm of z, i.e., $N(a+b\tau) = (a+b\tau)(a+b\bar{\tau}) = a^2 + \mu ab + 2b^2$ for $a, b \in \mathbb{Z}$.

Consider the directed graph $G = (V, A)$ defined by its set of vertices

$$V := \{0\} \cup \{z \in \mathbb{Z}[\tau] : N(z) \leqslant M\,, \tau \nmid z\}$$

and set of arcs

$$A := \{(y, z) \in V^2 : \text{There exist } d \in \mathcal{D} \setminus \{0\}, \text{ and } v \geqslant w \text{ s.t. } z = \tau^v y + d\} .$$

Then $\mathcal{D}$ is a w-NADS iff the following conditions are both satisfied.

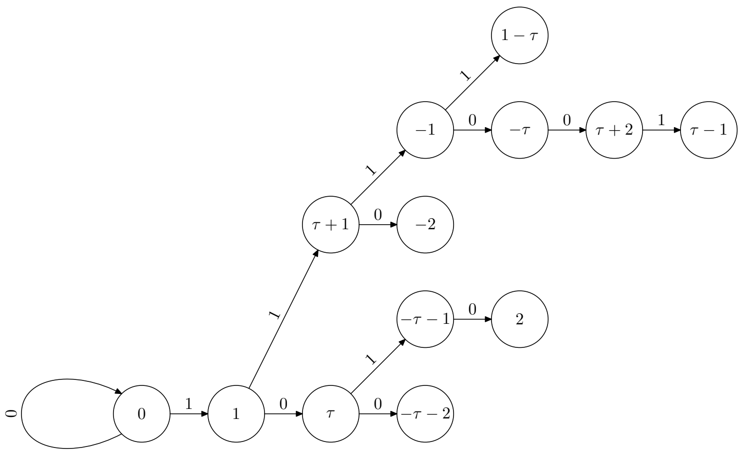


Fig. 1. Directed Graph G for $\mu = -1$, $w = 1$, $\mathcal{D} = \{0, 1\}$. The arcs are labeled with (v, d) as in the definition of the graph, i.e. $y \xrightarrow{(v,d)} z$ means that $z = \tau^v y + d$.

1. The set $\mathcal{D}$ contains a reduced residue system modulo τ^w.
2. In $G = (V, A)$, each vertex $z \in V$ is reachable from 0.

If $\mathcal{D}$ is a w-NADS and $\mathcal{D} \setminus \{0\}$ is a reduced residue system modulo τ^w, then each $z \in \mathbb{Z}[\tau]$ has a unique $\mathcal{D}$-w-NAF.

We now make some remarks and discuss two well-known examples.

Remark 1. A number $a + \tau b \in \mathbb{Z}[\tau]$ is relatively prime to τ iff a is odd. This follows from the fact that τ is a prime element in $\mathbb{Z}[\tau]$ and that τ divides a rational integer iff the latter is even.

Example 2. Let $w = 1$ and $\mathcal{D} = \{0, 1\}$. By Remark 1, there is only one residue class prime to τ. In this case $M = 5$, so $V = \{0, \pm 1, \pm 1 \pm \tau\}$. The corresponding directed graph in the case $\mu = -1$ is shown in Figure 1. The case $\mu = 1$ is similar.

We see that all 7 states are reachable from 0. Thus, $\{0, 1\}$ is a 1-NADS. This is equivalent to saying that τ is the base of a canonical number system in $\mathbb{Z}[\tau]$ in the sense of [13], and is a particular case of results from [12].

Remark 2. Example 2 implies that there are exactly 2^w residue classes modulo τ^w; a complete residue system is: $\bigl\{\sum_{j=0}^{w-1} \varepsilon_j \tau^j$ with $\varepsilon_j \in \{0, 1\}$ for $0 \leqslant j < w\bigr\}$. There are 2^{w-1} residue classes coprime to τ^w, a reduced residue system is: $\bigl\{1 + \sum_{j=1}^{w-1} \varepsilon_j \tau^j$ with $\varepsilon_j \in \{0, 1\}$ for $1 \leqslant j < w\bigr\}$.

Example 3. Let $w = 2$ and $\mathcal{D} = \{0, \pm 1\}$. Using Remark 2, it is easily seen that $\{\pm 1\}$ is a reduced residue system modulo τ^2. In this case, $M = 1$, the graph G consists of the three states $V = \{0, \pm 1\}$ only, and those are obviously reachable from 0. Thus $\{0, \pm 1\}$ is a 2-NADS. This has been proved by Solinas [23,24].

Example 4. Let us consider the digit set $\mathcal{D} = \{0\} \cup \{\pm 1, \pm 3, \ldots, \pm(2^{w-1} - 1)\}$. The odd digits form a reduced residue system modulo τ^w, since τ^w divides a rational integer if and only if 2^w divides it (note that τ and $\bar{\tau}$ are coprime primes in $\mathbb{Z}[\tau]$). However, this digit set is not a w-NADS for all w. For instance, for $w = 6$, the number $1 - \mu\tau$ has no $\mathcal{D}$-6-NAF. Using Theorem 1, we can verify that for $w \in \{2, 3, 4, 5, 7, 8, 9, 10\}$, this set $\mathcal{D}$ is a w-NADS.

2.2 Representatives of Minimal Norm

Theorem 2. *Let τ, $w \geqslant 2$ be as above, and $\mathcal{D}$ a digit set consisting of 0 together with one element of minimal norm from each odd residue class modulo τ^w.*

The digit set $\mathcal{D}$ is uniquely determined. In other words, in each odd residue class modulo τ^w there exists a unique element of minimal norm.

In [5,6] it has been shown that the τ-NAF has minimal weight among all the τ-adic expansions with digit set $\{0, \pm 1\}$. Since the digit set $\mathcal{D} = \{0, \pm 1 \pm \bar{\tau}\}$ is also Solinas' set for $w = 3$, in the same paper it is in fact shown that a $\mathcal{D}$-w-NAF with this digit set is a $\mathcal{D}$-expansion of minimal weight. For the radix 2 the analogous result is known to be true for all positive w [1,19]. So one might conjecture that the same holds for our choice of τ. But, the following example shows that this is not the case:

Example 5. Consider $\mu = -1$, $w = 4$, and the set $\mathcal{D}$ of minimal norm representatives modulo τ^w. We have $\mathcal{D} = \{0, \pm 1, \pm 1 \pm \tau, \pm(3 + \tau)\}$ and

$$\mathsf{value}(1, 0, 0, 0, -1 - \tau, 0, 0, 0, 1 - \tau) = -9 = \mathsf{value}(-3 - \tau, 0, 0, -1) \ .$$

The first expansion is the $\mathcal{D}$-w-NAF and has Hamming weight 3. The second expansion does not satisfy the w-NAF condition, has Hamming weight 2 and is even shorter than the first expansion.

Even worse, we exhibit chaotic behaviour in the following sense: for every integer $k > 0$, a pair of numbers can be found which are congruent modulo τ^k, but whose optimal $\mathcal{D}$-expansions differ even at the least significant position. Thus it is impossible to compute an optimal $\mathcal{D}$-expansion of z by a deterministic transducer automaton or an online algorithm.

Proposition 1. *Let $w = 4$, and $\mathcal{D} = \{0, \pm 1, \pm 1 \pm \tau, \pm(3 - \mu\tau)\}$ (all signs are independent) be the set of minimal norm representatives modulo τ^w. For every nonnegative integer ℓ, we define*

$$\begin{aligned} z_\ell &:= \mathsf{value}(\ \ 0, 0, 0, 0, \mu - \tau, (0, 0, 0, -3\mu + \tau)^{(\ell)}, 0, 0, 0, 0, 1 - \mu\tau, 0, 0, 0, -1) \ , \\ z'_\ell &:= \mathsf{value}(-\mu, 0, 0, 0, \mu - \tau, (0, 0, 0, -3\mu + \tau)^{(\ell)}, 0, 0, 0, 0, 1 - \mu\tau, 0, 0, 0, -1) \ , \end{aligned} \qquad (4)$$

where $(0, 0, 0, -3\mu + \tau)^{(\ell)}$ means that this four-digit block is repeated ℓ times. Then $z_\ell \equiv z'_\ell \pmod{\tau^{4\ell+13}}$. All $\mathcal{D}$-optimal expansions of z_ℓ are given by

$$((0, 0, 0, 3 - \mu\tau)^{(\ell_2)}, 0, 0, \mu - \tau, (0, 0, 0, -3\mu + \tau)^{(\ell_1)}, 0, 0, 0, 0, 1 - \mu\tau, 0, 0, 0, -1) \ ,$$

where ℓ_1 and ℓ_2 are nonnegative integers summing up to ℓ. There is only one $\mathcal{D}$-optimal expansion of z'_ℓ, it is given by

$$\big((0,0,0,-3+\mu\tau)^{(\ell+1)},0,0,0,0,-3\mu+\tau,0,0,1+\mu\tau\big) \ .$$

Note that the $\mathcal{D}$-optimal expansion of z'_ℓ has Hamming weight $\ell+3$, whereas the $\mathcal{D}$-w-NAF of z'_ℓ given in (4) has Hamming weight $\ell+4$. The proof is based on the search of shortest paths in an auxiliary automaton.

2.3 Syntactic Sufficient Conditions

The aim of this section is to prove sufficient conditions for families of sets $\mathcal{D}$ to be a w-NADS at the level of digits of the τ-NAF. In contrast to Theorem 1, where a decision can be made for any concrete set $\mathcal{D}$, we will now focus on families of such sets. Blake, Murty, and Xu [8] gave sufficient conditions based on the norm of the numbers involved.

Proposition 2. *Let $w \geqslant 1$ and $\boldsymbol{\varepsilon}$, $\boldsymbol{\varepsilon}'$ two τ-NAFs. Then* $\mathsf{value}(\boldsymbol{\varepsilon}) \equiv \mathsf{value}(\boldsymbol{\varepsilon}') \pmod{\tau^w}$ *if and only if*

$$\varepsilon_j = \varepsilon'_j \text{ for } 0 \leqslant j \leqslant w-2 \text{ and } |\varepsilon_{w-1}| = |\varepsilon'_{w-1}| \ . \tag{5}$$

Definition 3. *Let w be a positive integer and $\mathcal{D}$ be a subset of*

$$\{\,0\,\} \cup \{\, \mathsf{value}(\boldsymbol{\varepsilon}) : \boldsymbol{\varepsilon} \text{ is a } \tau\text{-NAF of length at most } w \text{ with } \varepsilon_0 \neq 0 \,\}$$

consisting of 0 *and a reduced residue system modulo τ^w. Then $\mathcal{D}$ is called a* set of short τ-NAF representatives *for τ^w.*

By Proposition 2, an example for a set of short τ-NAF representatives is

$$\mathcal{D} = \{\,0\,\} \cup \Big\{\, \mathsf{value}(\boldsymbol{\varepsilon}) : \boldsymbol{\varepsilon} \text{ is a } \tau\text{-NAF of length at most } w \\ \text{with } \varepsilon_0 \neq 0 \text{ and } \varepsilon_{w-1} \in \{0, \varepsilon_0\} \,\Big\} \ . \tag{6}$$

All other sets of short τ-NAF representatives are obtained by changing the signs of ε_{w-1} without changing ε_0 in some of the $\boldsymbol{\varepsilon}$. It is easy to check that the cardinality of $\mathcal{D}$ is indeed $1+2^{w-1}$.

The main result of this section is the following theorem, which states that in almost all cases, a set of short τ-NAF representatives is a w-NADS:

Theorem 3. *Let w be a positive integer and $\mathcal{D}$ a set of short τ-NAF representatives. Then $\mathcal{D}$ is a w-NADS iff it is not in the following table*

w	μ	$\mathcal{D}$	*Remark*
3	-1	$\{1,-1,-\tau^2+1,-\tau^2-1\}$	$(-\tau-1)\left(1-\tau^3\right) = -\tau^2+1$
3	-1	$\{1,-1,-\tau^2+1,\ \tau^2-1\}$	$(-\tau-1)\left(1-\tau^3\right) = -\tau^2+1$
3	-1	$\{1,-1,\ \tau^2+1,\ \tau^2-1\}$	$(\tau+1)\left(1-\tau^3\right) = \tau^2-1$
3	1	$\{1,-1,-\tau^2+1,\ \tau^2-1\}$	$(-\tau+1)\left(1-\tau^6\right) = \left(-\tau^2+1\right)\tau^3+\tau^2-1$

(the "Remark" column contains an example of an element which cannot be represented). In particular, if $w \geqslant 4$, then $\mathcal{D}$ is always a w-NADS.

The following result is concerned with the lengths of recodings that make use of the set of short τ-NAF representatives.

Theorem 4. *Let $w \geqslant 2$ be a positive integer, $\mathcal{D}$ a set of short τ-NAF representatives, and ε a $\mathcal{D}$-w-NAF of some $z \in \mathbb{Z}[\tau]$.*

Then the length of ε can be bounded by

$$\begin{aligned} 2\log_2|z| - w - 0.18829 &< \mathsf{length}(\varepsilon) < 2\log_2|z| + 7.08685\ , &&\text{if } w \geqslant 4\ , && (7)\\ 2\log_2|z| - 2.61267 &< \mathsf{length}(\varepsilon) < 2\log_2|z| + 5.01498\ , &&\text{if } w = 3\ , && (8)\\ 2\log_2|z| - 0.54627 &< \mathsf{length}(\varepsilon) < 2\log_2|z| + 3.51559\ , &&\text{if } w = 2\ . && (9)\end{aligned}$$

Note that (9) is Solinas' [24] Equation (53).

2.4 Point Halving

For any given point P, point halving [14,22] consists in computing a point Q such that $2Q = P$. This operation applies to all elliptic curves over binary fields. Its evaluation is (at least two times) faster than that of a doubling and a halve-and-add scalar multiplication algorithm based on halving instead of doubling can be devised. This method is not useful for Koblitz curves because halving is slower than a Frobenius operation.

In [3] it is proposed to insert a halving in the "τ-and-add" method to speed up Koblitz curve scalar multiplication. This approach brings a non-negligible speedup and was refined in [5,6], where the insertion of a halving was interpreted as a digit set extension as follows: Inserting a halving in the scalar multiplication is equivalent to adding $\pm\bar{\tau}$ to the digit set $\{0, \pm 1\}$. Note that, by Theorem 3, $\mathcal{D} = \{0, \pm 1, \pm\bar{\tau}\}$ is the only 3-NADS of short τ-NAF representatives. In particular $\mathcal{D}' = \{\pm 1, \pm\bar{\tau}\}$ is a reduced residue system modulo τ^3.

The next two theorems state that more powers of $\bar{\tau}$ still produce reduced residue systems $\mathcal{D}'$, which in some cases give rise to w-NADS.

Theorem 5. *Let $w \geqslant 2$. Then $\mathcal{D}' := \{\pm\bar{\tau}^k : 0 \leqslant k < 2^{w-2}\}$ is a reduced residue system modulo τ^w.*

Theorem 6. *Let $\mathcal{D} := \{0\} \cup \{\pm\bar{\tau}^k : 0 \leqslant k < 2^{w-2}\}$. If $w \in \{2, 3, 4, 5, 6\}$ then $\mathcal{D}$ is a w-NADS. If $w \in \{7, 8, 9, 10, 11, 12\}$ then $\mathcal{D}$ is not a w-NADS.*

Sketch of the Proof of Theorem 6. For every pair (w, μ) with $w \leqslant 6$ the conditions of Theorem 1 have been verified by heavy symbolic computations.

For $7 \leqslant w \leqslant 12$ and both values of μ the graph G contains loops that are not reachable from 0. In other words, there are elements in $\mathbb{Z}[\tau]$ that have periodic expansions. For example, if $w = 7$ and $\mu = 1$ we have

$$-9 + 34\,\tau = \frac{\bar{\tau}^{27} - \bar{\tau}^6\tau^7}{1 - \tau^{16}} = \bar{\tau}^{27} - \bar{\tau}^6\tau^7 + \bar{\tau}^{27}\tau^{16} - \bar{\tau}^6\tau^{23} + \bar{\tau}^{27}\tau^{32} - \cdots$$

and the number $371 - 20\,\tau$ has for all w with $8 \leqslant w \leqslant 12$ (also with $\mu = 1$) periodic expansion $(\bar{\tau}^{41} - \bar{\tau}^5\tau^{12})(1 - \tau^{24})^{-1}$. $\square$

2.5 Comparing the Digit Sets

So far, three digit sets have been studied: the minimal norm representatives, short NAF representatives, and powers of $\bar{\tau}$. It is a natural question to ask what are the relations between these sets when they are w-NADS.

The minimal norm representatives are exactly the powers of $\bar{\tau}$ for $w \leqslant 4$. For the same range of w, all digits of these digit sets have a τ-NAF of length at most w, which implies that they are also digit sets of short NAF representatives.

If symmetry is required, i.e., if d is a digit, then $-d$ must also be a digit, by Theorem 3 there is only *one* w-NADS of short NAF representatives for $w \leqslant 3$: it coincides with the digit sets of minimal norm representatives and powers of $\bar{\tau}$. For $w = 4$, however, there is a symmetric w-NADS of short NAF representatives distinct from the other two digit sets. For $w \geqslant 5$, the three concepts are different: the lengths of the τ-NAFs of the powers of $\bar{\tau}$ grow exponentially in w, and the lengths of some minimal norm representatives exceed w slightly (at most by 2).

The table below summarizes the above considerations and provides further information. "MNR" stands for the minimal norm representatives digit set, whereas "P$\bar{\tau}$" stands for the powers of $\bar{\tau}$. The last two rows show the maximum length of the τ-NAFs of the digits.

w	2	3	4	5	6
MNR=P$\bar{\tau}$	True	True	True	False	False
Max τ-NAF length MNR	1	3	4	6	8
Max τ-NAF length P$\bar{\tau}$	1	3	4	8	17

3 Applications

All digit sets seen so far can be used in a τ-and-add scalar multiplication, where we first precompute dP for all $d \in \mathcal{D} \setminus \{0\}$ and then we evaluate the scheme $\sum z_i \tau^i(P)$; in fact, only a half of the precomputations usually suffice since in all cases that we explicitly described the non-zero elements of the digit set come in pairs of elements of opposite sign.

The digit set from §2.3 simplifies the precomputation phase. The digit set from §2.4 allows us to perform precomputations very quickly or to get rid of them completely. In the next two subsections we shall consider these facts in detail. In §3.3 we explain how to use digit sets which are not w-NADS when they contain a subset that is a k-NADS for smaller k.

3.1 Using the Short-NAF Digit Set

Let us consider here the digit set $\mathcal{D}$ defined in (6). With respect to Solinas' set it has the advantage of being syntactically defined. If a computer has to work with different curves, different scalar sizes and thus with different optimal choices for the window size, the representatives in Solinas' set must be recomputed – or they must be retrieved from a set of tables. In some cases, the time to compute representatives of minimal norm may have to be subsumed in the total scalar multiplication time. This is not the case with our set. This flexibility is also particularly important for computer algebra systems.

The scalar is first recoded as a τ-NAF, and the elements of $\mathcal{D}$ are associated to NAFs of length at most w with non-vanishing least significant digit, and thus to certain *odd integers* in the interval $[-a_w, a_w]$ where $a_w = \frac{2^{w+1}-2(-1)^w}{3} - 1$. These integers can be used to index the elements in the precomputation table. We need only to precompute the multiples of the base point by "positive" short NAFs (i.e. with most significant digit equal to 1) – and the corresponding integers are the odd integers in the interval $[0, a_{w-1}]$ together with the integers $\equiv 1 \pmod 4$ in $[a_{w-1}+2, a_w]$. The indices in the table are then obtained by easy compression. The precomputed elements for the scalar multiplication loop can thus be retrieved upon direct reading the τ-NAF, of which we need only to compute the least w significant places. If the least and the w-th least significant digits of this segment of the τ-NAF are both non-zero and have different signs, a carry is generated: Thus, the computation of the τ-NAF should be interleaved with the parsing for short NAFs.

3.2 τ-Adic Scalar Multiplication with Repeated Halvings

Let $w \geqslant 2$ be an integer and $\mathcal{D}$ the digit set defined in §2.4. Let P be a point on an elliptic curve and $Q_j := \tau^j(2^{-j}P)$ for $0 \leqslant j < 2^{w-2}$ and $R := Q_{2^{w-2}-1}$. To compute zP, we have to compute yR for $y := \bar{\tau}^{2^{w-2}-1}z$. Computing a $\mathcal{D}$-w-NAF of y, this can be done by using the points Q_j, $0 \leqslant j < 2^{w-2}$ as precomputations.

Now, a point halving on an elliptic curve is much faster than a point doubling, and a point addition is not faster than a doubling. Now, with, say, Solinas' set or the short τ-NAF representatives the precomputations always involve at least one addition per digit set element. With our set we require a halving per digit set element. Hence, our approach with the points Q_j and halvings is already faster than traditional ones.

But we can do even better, especially if normal bases are used to represent the field $\mathbb{F}_{2^n}$. Algorithm 2 computes zP using an expansion $y = \sum_{i=0}^{\ell} y_i\tau^i$ of the integer $y := \bar{\tau}^{2^{w-2}-1}z$ where the digits y_i belong to the digit set introduced in Theorem 5, i.e. $\mathcal{D} := \{0\} \cup \{\pm\bar{\tau}^k : 0 \leqslant k < 2^{w-2}\}$.

To explain how it works we introduce some notation. Write $y_i = \varepsilon_i\bar{\tau}^{k_i}$ with $\varepsilon_i \in \{0, \pm 1\}$. We also define

$$y^{(k)} = \sum_{i\,:\,0\leqslant i\leqslant \ell,\ y_i=\pm\bar{\tau}^k} \varepsilon_i\tau^i \quad .$$

Now $y = \sum_{k=0}^{2^{w-2}-1} y^{(k)}\bar{\tau}^k$ and therefore

$$\begin{aligned} zP = \bar{\tau}^{-(2^{w-2}-1)}yP &= \left(\sum_{m=0}^{2^{w-2}-1} y^{(m)}\bar{\tau}^m\right)\bar{\tau}^{-(2^{w-2}-1)}P \\ &= \sum_{m=0}^{2^{w-2}-1} y^{(m)}\bar{\tau}^{m-(2^{w-2}-1)}P = \sum_{m=0}^{2^{w-2}-1} \left(\frac{\tau}{2}\right)^{2^{w-2}-1-m} (y^{(m)})P \quad . \end{aligned}$$

Algorithm 2. τ-adic Scalar Multiplication with Repeated Halvings

INPUT: A Koblitz curve E_a, a point P of odd order on it, and a scalar z.
OUTPUT: zP

```
1.  $y \leftarrow \bar{\tau}^{2^{w-2}-1} z$
    Write $y = \sum_{i=0}^{\ell} y_i \tau^i$ where $y_i \in \mathcal{D} := \{0\} \cup \pm\{\bar{\tau}^k : 0 \leqslant k < 2^{w-2}\}$
    Write $y_i = \varepsilon_i \bar{\tau}^{k_i}$ with $\varepsilon_i \in \{0, \pm 1\}$
2.  $\ell_k \leftarrow \max\left(\{-1\} \cup \{i : y_i = \pm\bar{\tau}^k \text{ for some } k\}\right)$
3.  $X \leftarrow 0$
4.  for $k = 0$ to $2^{w-2} - 1$ do
5.      if $k > 0$ then $X \leftarrow \tau^{n-\ell_k} X$, $X \leftarrow \frac{1}{2} X$
6.      for $i = \ell_k$ to $0$ do
7.          $X \leftarrow \tau X$
8.          if $y_i = \pm\bar{\tau}^k$ then $X \leftarrow X + \varepsilon_i P$
9.  return $(X)$
```

The last expression is evaluated by a Horner scheme in $\frac{\tau}{2}$, i.e. by repeated applications of τ and a point halving, interleaved with additions of $y^{(0)}P$, $y^{(1)}P$, etc. The elements $y^{(k)}P$ are computed by a τ-and-add loop as usual. To save a memory register, instead of computing $y^{(k)}P$ and then adding it to a partial evaluation of the Horner scheme, we apply τ to the negative of the length of $y^{(k)}$ (which is $1 + \ell_k$) to the intermediate result X and perform the τ-and-add loop to evaluate $y^{(k)}P$ starting with this X instead of a "clean" zero. In Step 5 there is an optimization already present in [3]: n is added to the exponent (since $n \approx \ell_k$ and τ^n acts like the identity on the curve) and the operation is also partially fused to the subsequent $\frac{\tau}{2}$. At the end of the internal loop the relation $X = \sum_{m=0}^{k} \left(\frac{\tau}{2}\right)^{k-m} y^{(m)}P$ holds, thus proving the correctness.

Apart from the input, we only need to store the additional variable X and the recoding of the scalar. The multiplication of z by $\bar{\tau}^{2^{w-2}-1}$ is an easy operation, and the negative powers of τ can be easily eliminated by multiplying by a suitable power of τ^n which operates trivially on the points of the curve. Reduction of this scalar by $(\tau^n - 1)/(\tau - 1)$ following [23,24] is also necessary.

An issue with Algorithm 2 is that the number of Frobenius operations may increase exponentially with w, since the internal loop is repeated up to 2^{w-2} times. This is not a problem if a normal basis is used to represent the field, but may induce a performance penalty with a polynomial basis. A similar problem was faced by the authors of [20], and they solved it adapting an idea from [21]. The idea consists in keeping a copy R of the point P in normal basis representation. Instead of computing $y^{(k)}P$ by a Horner scheme in τ, the summands $\varepsilon_i \tau^i P$ are just added together. The power of the Frobenius is applied to R *before* converting the result back to a polynomial basis representation and accumulating it. According to [10] a basis conversion takes about the same time as one polynomial basis multiplication, and the two conversion routines require each a matrix that occupies $O(n^2)$ bits of memory.

Algorithm 3. Low-memory τ-adic Scalar Multiplication on Koblitz Curves with Repeated Halvings, for Fast Inversion

```
INPUT: $P \in E(\mathbb{F}_{2^n})$, scalar $z$
OUTPUT: $zP$

1.  $y \leftarrow \bar{\tau}^{2^{w-2}-1} z$
    Write $y = \sum_{i=0}^{\ell} y_i \tau^i$ where $y_i \in \mathcal{D} := \{0\} \cup \pm\{\bar{\tau}^k : 0 \leqslant k < 2^{w-2}\}$
    Write $y_i = \varepsilon_i \bar{\tau}^{k_i}$ with $\varepsilon_i \in \{0, \pm 1\}$
2.  $R \leftarrow \mathrm{normal\_basis}(P)$
3.  $Q \leftarrow 0$
4.  for $k = 0$ to $2^{w-2} - 1$
5.      if $k > 0$ then $Q \leftarrow \tau Q,\ Q \leftarrow \frac{1}{2} Q$
6.      for $i = 0$ to $\ell$
7.          if $y_i = \pm\bar{\tau}^k$ then $Q \leftarrow Q + \varepsilon_i \mathrm{polynomial\_basis}(\tau^i R)$
8.  return $Q$
```

Algorithm 4. Low-memory τ-adic Scalar Multiplication on Koblitz Curves with Repeated Doublings, for Slow Inversion

```
INPUT: $P \in E(\mathbb{F}_{2^n})$, scalar $z$
OUTPUT: $zP$

1.  Write $z = \sum_{i=0}^{\ell} z_i \tau^i$ where $z_i \in \mathcal{D} := \{0\} \cup \pm\{\bar{\tau}^k : 0 \leqslant k < 2^{w-2}\}$
    Write $z_i = \varepsilon_i \bar{\tau}^{k_i}$ with $\varepsilon_i \in \{0, \pm 1\}$
2.  $R \leftarrow \mathrm{normal\_basis}(P)$                                   [Keep in affine coordinates]
3.  $Q \leftarrow 0$                                                   [$Q$ is in Lopez-Dahab coodinates]
4.  for $k = 2^{w-2} - 1$ to $0$
5.      if $k > 0$ then $Q \leftarrow \tau^{-1} Q,\ Q \leftarrow 2Q$                [$\tau^{-1}$ is three square roots]
6.      for $i = 0$ to $\ell$
7.          if $z_i = \pm\bar{\tau}^k$ then $Q \leftarrow Q + \varepsilon_i \mathrm{polynomial\_basis}(\tau^i R)$     [Mixed coord.]
8.  return $Q$                                             [Convert to affine coordinates]
```

Algorithm 3 is a realisation of this approach. It is suited in the context where a polynomial basis is used for a field and the cost of an inversion is not prohibitive. The routines normal_basis and polynomial_basis convert the coordinates of the points between polynomial and normal bases.

Algorithm 4 is the version for fields with a slow inversion (such as large fields). It uses inversion-free coordinate systems and, since no halving formula is known for such coordinates, a doubling is used instead of a halving. This is not a problem, since using Projective or López-Dahab coordinates (see [9, § 15.1]) a doubling followed by an application of τ^{-1} (which amounts to three square root extractions) is about twice as fast as a mixed-coordinate addition preceded by a basis conversion, hence the situation is advantageous as the previous one. This also dispenses us with the need of using an auxiliary scalar y.

Algorithm 5. Windowed Integer Recoding With Termination Guarantee

INPUT: An element z from $\mathbb{Z}[\tau]$, a natural number $w \geqslant 1$ and a set of reduced residue systems $\mathcal{D}'_k \subset \mathcal{D}'_{k+1} \subset \ldots \subset \mathcal{D}'_w$ modulo $\tau^k, \tau^{k+1}, \ldots, \tau^w$ respectively, ($1 \leqslant k < w$) where $\mathcal{D}'_k \cup \{0\}$ is a k-NADS.
OUTPUT: A representation $z = \sum_{j=0}^{\ell-1} z_j \tau^j$ of length ℓ.

```
1.  $j \leftarrow 0, u \leftarrow z, v \leftarrow w$
2.  while $u \neq 0$ do
3.      if $\tau \mid u$ then
4.          $z_j \leftarrow 0$
5.      else
6.          Let $z_j \in \mathcal{D}'_v$ s.t. $z_j \equiv u \pmod{\tau^v}$
7.          if $( |z_j| \geqslant |u|(2^{v/2} - 1)$ AND $v > k )$ then decrease $v$ and retry:
8.              $v \leftarrow v - 1$, go to Step 6
9.      $u \leftarrow u - z_j,\ u \leftarrow u/\tau,\ j \leftarrow j + 1$
10. $\ell \leftarrow j$
11. return $(\{z_j\}_{j=0}^{\ell-1}, \ell)$
```

Although the digit set $\mathcal{D}$ introduced in Theorem 5 is not a w-NADS for all w, in the next subsection we show how to save the situation.

3.3 Stepping Down Window Size

Let $\mathcal{D}_w$ be a family of digit sets, parametrized by an integer w, which are w-NADS for some small values of w, but not in general, and such that $\mathcal{D}_{v-1} \subset \mathcal{D}_v$ for all v. Then, Algorithm 1 may enter a loop for a few inputs. This can be caused by the appearance of "large" digits towards the end of the main loop of the recoding algorithm. Then the norm of the variable u gets too small in comparison to the chosen digit, and $|u| \leqslant \left|\frac{u - z_j}{\tau^w}\right| \leqslant \frac{|u| + |z_j|}{2^{w/2}}$. For most other inputs the algorithm terminated and delivers the expected low density. How can we save it? One possibility is to decrease w for the rest of the computation, so that the corresponding digit set is a NADS. We call this operation *stepping down*. The resulting recoding may have a slightly higher weight, but the algorithm is guaranteed to terminate. One possible implementation is presented as Algorithm 5.

Solinas can prove that his τ-adic w-NAF terminates because his digits are representants of minimal norm, and have norm bounded by $\frac{4}{7}2^w$. The presence of digits of non-minimal norm is a necessary but not sufficient condition for non-termination. In fact, the digit sets from Example 4 and from §2.4 are w-NADS with some digits of norm larger than 2^w.

Remark 3. The digit set from Example 4, the syntactically defined set of §2.3 and the set of Theorem 5 all have the property that each set is contained in the sets with larger w – hence Algorithm 5 can be used.

Remark 4. Checking an absolute value (or a norm) in Algorithm 5, Step 7 is expensive. Hence we need an alternative strategy. Let M_w be defined as M in Theorem 1 for the digit set we are considering, with parameter w. Consider an easy function that is bounded by the norm: for example, if $z = a + b\tau$, $\lambda(z) = \max\{\lceil |a + \frac{\mu}{2} b| \rceil^2, 2\lceil |\frac{\mu}{4} a + b| \rceil^2\}$. It is easy to check that $\lambda(z) \leqslant N(z)$ and that $\lambda(z) = 0$ iff $z = 0$. Therefore, if $\lceil \log_2(M_v) \rceil \geqslant \lfloor \log_2(\lambda(z)) \rfloor$ we step down to a new value of v with $\lceil \log_2(M_v) \rceil < \lfloor \log_2(\lambda(z)) \rfloor$. These checks are quickly computed only by using the bit lengths of a and b and performing additions, subtractions and bit shifts (but no expensive multiplication). The values $\lceil \log_2(M_v) \rceil$ are precomputed in an easy way.

Remark 5. In our experiments, the recodings done with the different digit sets have similar length, the average density is $1/(w+1)$ (see also § 3.4), and stepping down only marginally increases the weight. Thus, the new digit sets bring their advantages with *de facto* no performance penalty.

3.4 A Performance Remark

Algorithms 2, 3 and 4 compute scalar multiplications by performing $2^{w-2} - 1$ "faster" operation blocks and (roughly) $n/(w+1)$ "slower" operation blocks. In Algorithm 2 (with normal bases) the two block types are given by a halving, resp. an addition. In Algorithm 3 (resp. 4) these two block types are given by a Frobenius operation plus a halving (resp. by an inverse Frobenius plus a doubling), and by a basis conversion followed by an addition (for both algorithms). In all cases we can see that computing the first block takes α times the time for the second block, where $\alpha \leqslant 1/2$.

We now determine asymptotically optimal values for w in these algorithms in terms of n, where n is assumed to be large. This will lead to large values w, such that the digit set from § 2.4 is probably not a w-NADS. We will therefore have to use Algorithm 5 (or a variant of it). For the sake of simplicity, we do not decrease v step by step depending on the norm of $|z_j|$, but we use $v = w$ for $j < L$ and $v = 6$ for $j \geqslant L$, where the parameter L will be chosen below.

Let z be a random integer in $\mathbb{Z}[\tau]$ with $|z| \leqslant |\tau|^n$. Here "random" means that for every positive integer m, every residue class modulo τ^m is equally likely. Let $y = \sum_{j=0}^{L-1} z_j \tau^j$ where the z_j are calculated by Algorithm 5. Then $y \equiv z \pmod{\tau^L}$ and $|y| \leqslant |\tau|^{2^{w-2}-1+L-1}(1 + |\tau|^{-w})^{-1}$. Thus $|(z-y)/\tau^L| \leqslant |\tau|^{n-L} + |\tau|^{2^{w-2}-2}$. The choice

$$L = n - 2^{w-2} + 2$$

implies that $|(z-y)/\tau^L| \leqslant 2|\tau|^{n-L}$. The expected length of the $\mathcal{D}_6$-6-NAF of $(z-y)/\tau^L$ is $n - L + O(1)$. Here, $\mathcal{D}_6 = \{0\} \cup \{\pm\bar{\tau}^k : 0 \leqslant k < 16\}$. We conclude that the expected Hamming weight of the expansion returned by Algorithm 5 is

$$\frac{L}{w+1} + \frac{n-L}{7} + O(1) \ .$$

Here, we use the well-known fact that a v-NAF of length m has expected Hamming weight $m/(v+1) + O(1)$.

Algorithm 2 performs $2^{w-2}-1$ point halvings, the number of additions being given by the Hamming weight of the expansion. With α as above, the total costs of the curve operations (measured in additions) is

$$\alpha 2^{w-2} + \frac{L}{w+1} + \frac{n-L}{7} + O(1) = 2^{w-2}\left(\alpha + \frac{1}{7}\right) + \frac{n-2^{w-2}}{w+1} + O(1) \ .$$

Balancing the two main terms gives

$$\hat{w} = \frac{1}{\log 2} W\left(\frac{7 \cdot 2^{\frac{21\alpha+10}{7\alpha+1}} \log 2}{7\alpha+1} n\right) - \frac{7\alpha+8}{7\alpha+1} \ .$$

where W is the main branch of Lambert's W function. Asymptotically, this is $\hat{w} = \log_2 n - \log_2 \log_2 n + 2 - \log_2\left(\alpha + \frac{1}{7}\right) + O\left(\frac{\log\log n}{\log n}\right)$. Thus we choose

$$w = \left\lfloor \log_2 n - \log_2 \log_2 n + 2 - \log_2\left(\alpha + \frac{1}{7}\right) \right\rfloor$$

and see that the expected number of curve additions asymptotically equals

$$\frac{n}{\log_2 n}\left(1 + c + O\left(\frac{\log\log n}{\log n}\right)\right) \tag{10}$$

with $\frac{1}{2} < c = 2^{-\{\log_2 n - \log_2\log_2 n + 2 - \log_2\left(\alpha+\frac{1}{7}\right)\}} \leqslant 1$.

For Algorithms 3 and 4, the unit in the cost (10) is given by the cost of a group addition and a base conversion – the latter being comparable to a field multiplication. We thus have the following result:

Theorem 7. *Algorithms 2, 3 and 4 are sublinear scalar multiplication algorithms on Koblitz Curves with constant input-dependent memory consumption.*

Note that here *sublinear* refers to the number of group operations, and "constant memory consumption" refers to the number of registers required for temporary variables – each one taking of course $O(n)$ bits. Usual windowed methods with precomputations have, of course, similar time complexity but use storage for $2^{w-2}-1$ points [23,24] and thus $O(n2^w) = O(n^2/\log n)$ bits of memory. Algorithms 3 and 4 need $O(n^2)$ bits of field-dependent (but not point-dependent) data for base conversion (as in [21,20]) that can be stored statically (such as in ROM).

For the same values of w, our algorithms perform better than techniques storing precomputations. The precomputation stage with Solinas' digit set takes one addition and some Frobenius operations per precomputation. Using the digit set from § 2.4 these additions can be replaced with cheaper operations (halvings or doublings depending on the coordinate system), whereas in Algorithms 3 and 4 the cost of the basis conversion associated to each addition in the main loop is relatively small. In all cases, the increase in recoding weight is marginal. A more precise performance evaluation (including small values of n and w) lies beyond the scope of this paper; however, in [2] some simple operation counts and comparisons with other methods can be found. The method in [7] is also sublinear, but its applicability still has to be assessed – the authors warn that the involved constants may be quite large. See [4] for another approach.

4 Conclusions

The paper at hand presents several new results about τ-adic recodings.

Digit sets allowing a w-NAF to be computed for all inputs are characterised. We study digit sets with interesting properties for Koblitz curves.

We prove that Solinas' digit set, characterised by the property that the elements have minimal norm, is uniquely determined. We show, by means of an example, that the non adjacency property does not imply minimality of weight, and enunciate a result implying that optimal expansions cannot be computed by a deterministic finite automaton.

In § 2.3 we introduce a new digit set characterised by syntactic properties. Its usage is described in § 3.1.

The digit set introduced in § 2.4 together with Algorithms 2, 3 and 4 permit to perform a "windowed" τ-adic scalar multiplication without requiring storage for precomputed points. This is potentially useful for implementation on restricted devices. Our methods can perform better than previous methods that make use of precomputations. Some operation counts (based on the performance of real-world implementations of finite field arithmetic) comparing our algorithms with other methods can be found in [2]. Better performance assessments are part of future work.

Acknowledgements. This paper was partly written during a joint visit of the first two authors to the Department of Mathematical Sciences, Stellenbosch University, and during a visit (supported by the FWF project S9606) of the first author to the Institut für Mathematik B, Technische Universität Graz. The authors thank these institutions for their hospitality.

Disclaimer. The information in this document reflects only the authors' views, is provided as is and no guarantee or warranty is given that the information is fit for any particular purpose. The user thereof uses the information at its sole risk and liability.

References

1. Avanzi, R.M.: A Note on the Signed Sliding Window Integer Recoding and its Left-to-Right Analogue. In: Handschuh, H., Hasan, M.A. (eds.) SAC 2004. LNCS, vol. 3357, pp. 130–143. Springer, Heidelberg (2004)
2. Avanzi, R.M.: Delaying and Merging Operations in Scalar Multiplication: Applications to Curve-Based Cryptosystems. In: Biham, E., Youssef, A.M. (eds.) SAC 2006. LNCS. vol. 4356, pp. 203–219, Springer, Heidelberg
3. Avanzi, R.M., Ciet, M., Sica, F.: Faster Scalar Multiplication on Koblitz Curves combining Point Halving with the Frobenius Endomorphism. In: Bao, F., Deng, R., Zhou, J. (eds.) PKC 2004. LNCS, vol. 2947, pp. 28–40. Springer, Heidelberg (2004)
4. Avanzi, R.M., Dimitrov, V., Doche, C., Sica, F.: Extending Scalar Multiplication using Double Bases. In: ASIACRYPT 2006, LNCS. vol. 4284, pp. 130–144, Springer, Heidelberg (2006)

5. Avanzi, R.M., Heuberger, C., Prodinger, H.: Minimality of the Hamming Weight of the τ-NAF for Koblitz Curves and Improved Combination with Point Halving. In: Preneel, B., Tavares, S. (eds.) SAC 2005. LNCS, vol. 3897, pp. 332–344. Springer, Heidelberg (2006)
6. Avanzi, R.M., Heuberger, C., Prodinger, H.: Scalar Multiplication on Koblitz Curves Using the Frobenius Endomorphism and its Combination with Point Halving: Extensions and Mathematical Analysis. Algorithmica 46, 249–270 (2006)
7. Avanzi, R.M., Sica, F.: Scalar Multiplication on Koblitz Curves Using Double Bases. Cryptology ePrint Archive. In: VIETCRYPT 2006, LNCS. vol. 4341, pp. 131–146, Springer, Heidelberg (2006)
8. Blake, I.F., Murty, V.K., Xu, G.: A note on window τ-NAF algorithm. Information Processing Letters 95, 496–502 (2005)
9. Cohen, H., Frey, G. (eds.): The Handbook of Elliptic and Hyperelliptic Curve Cryptography. CRC Press, Boca Raton (2005)
10. Coron, J.-S., M'Raïhi, D., Tymen, C.: Fast generation of pairs $(k, [k]p)$ for Koblitz elliptic curves. In: Vaudenay, S., Youssef, A.M. (eds.) SAC 2001. LNCS, vol. 2259, pp. 151–164. Springer, Heidelberg (2001)
11. Heuberger, C., Prodinger, H.: Analysis of Alternative Digit Sets for Nonadjacent Representations. Monatshefte für Mathematik, pp. 219–248 (2006)
12. Kátai, I., Kovács, B.: Canonical number systems in imaginary quadratic fields. Acta Math. Hungar. 37, 159–164 (1981)
13. Kátai, I., Szabó, J.: Canonical Number Systems for Complex Integers. Acta Scientiarum Mathematicarum 1975, 255–260
14. Knudsen, E.W.: Elliptic Scalar Multiplication Using Point Halving. In: Lam, K.-Y., Okamoto, E., Xing, C. (eds.) ASIACRYPT 1999. LNCS, vol. 1716, pp. 135–149. Springer, Heidelberg (1999)
15. Koblitz, N.: Elliptic curve cryptosystems. Math. Comp. 48, 203–209 (1987)
16. Koblitz, N.: CM-curves with good cryptographic properties. In: Feigenbaum, J. (ed.) CRYPTO 1991. LNCS, vol. 576, pp. 279–287. Springer, Heidelberg (1992)
17. Miller, V.S.: Use of elliptic curves in cryptography. In: Williams, H.C. (ed.) CRYPTO 1985. LNCS, vol. 218, pp. 417–426. Springer, Heidelberg (1986)
18. Muir, J.A., Stinson, D.R.: Alternative digit sets for nonadjacent representations. In: Matsui, M., Zuccherato, R.J. (eds.) SAC 2003. LNCS, vol. 3006, pp. 306–319. Springer, Heidelberg (2004)
19. Muir, J.A., Stinson, D.R.: Minimality and other properties of the width-w nonadjacent form. Math. Comp. 75, 369–384 (2006)
20. Okeya, K., Takagi, T., Vuillaume, C.: Short Memory Scalar Multiplication on Koblitz Curves. In: Rao, J.R., Sunar, B. (eds.) CHES 2005. LNCS, vol. 3659, pp. 91–105. Springer, Heidelberg (2005)
21. Park, D.J., Sim, S.G., Lee, P.J.: Fast scalar multiplication method using change-of-basis matrix to prevent power analysis attacks on Koblitz curves. In: Chae, K.-J., Yung, M. (eds.) Information Security Applications. LNCS, vol. 2908, pp. 474–488. Springer, Heidelberg (2004)
22. Schroeppel, R.: Elliptic curve point ambiguity resolution apparatus and method. International Application Number PCT/US00/31014 (filed 9 November, 2000)
23. Solinas, J.A.: An improved algorithm for arithmetic on a family of elliptic curves. In: Kaliski Jr., B.S. (ed.) CRYPTO 1997. LNCS, vol. 1294, pp. 357–371. Springer, Heidelberg (1997)
24. Solinas, J.A.: Efficient Arithmetic on Koblitz Curves. Designs, Codes and Cryptography 19(2/3), 125–179 (2000)

Pairing Calculation on Supersingular Genus 2 Curves

Colm Ó hÉigeartaigh and Michael Scott

School of Computing, Dublin City University.
Ballymun, Dublin 9, Ireland
{coheigeartaigh,mike}@computing.dcu.ie

Abstract. In this paper we describe how to efficiently implement pairing calculation on supersingular genus 2 curves over prime fields. We find that, contrary to the results reported in [8], pairing calculation on supersingular genus 2 curves over prime fields is efficient and a viable candidate for the practical implementation of pairing-based cryptosystems. We also show how to eliminate divisions in an efficient manner when computing the Tate pairing, assuming an even embedding degree, and how this algorithm is useful for curves of genus greater than one.

Keywords: Tate pairing, hyperelliptic curves, pairing computation.

1 Introduction

Following a seminal paper by Boneh and Franklin [5] in 2001, there has been an explosion of interest in the exploitation of the properties of bilinear pairings on elliptic curves for cryptographic protocols. Naturally, there has also been much focus on the efficient implementation of pairings. Victor Miller gave the first algorithm [20,21] for computing a bilinear pairing, specifically the Weil pairing. However in practice the Tate pairing is used, as it is computationally less expensive.

In an important paper, Barreto et al. [2] gave criteria under which divisions in Miller's algorithm can be eliminated entirely. According to [25], this reduces the calculation time by almost 50%. Other papers which describe important improvements to computing the Tate pairing on elliptic curves are [12] and [3]. Numerous papers describe the actual implementational details, for example see [26].

Although the vast majority of work has been done using elliptic curves, an increasing amount of attention is being focused on computing pairings using hyperelliptic curves of genus 2. Choie and Lee [8] investigate the implementation of the Tate pairing on supersingular genus 2 curves of embedding degree 4, over large prime fields. Barreto et. al. [1] describe an efficient implementation of the Tate pairing using the eta pairing construct on supersingular genus 2 curves, over fields of characteristic 2. The significance of this paper is that it not only shows that pairing computation is comparable on genus 2 curves to elliptic curves, but that it can in fact be even faster.

© Springer-Verlag Berlin Heidelberg 2007

In this paper, we first of all give an improvement to Miller's algorithm for calculating the Tate pairing for arbitrary curves of even embedding degree. This algorithm is more efficient than the GHS algorithm [12], but not as efficient as the denominator elimination technique of Barreto et. al [2] in the general case. However, we show that the new algorithm is more efficient than the denominator elimination algorithm for special cases using curves of genus greater than one.

We then report an efficient implementation of the Tate pairing on genus 2 curves over prime fields. We detail various enhancements to Miller's algorithm that are in the literature, explaining how to apply them to the genus 2 case using prime fields for the first time. We give more efficient formulae for computing the functions required in Miller's algorithm for the genus 2 case than that reported in [8], the saving being a squaring and a multiplication in each iteration. Finally, we report timings for computing the Tate pairing, that establishes new benchmarks for pairing implemention on genus 2 curves over large prime fields.

This paper is organised as follows. Section 2 gives an overview of the Tate pairing and Miller's Algorithm. Section 3 details an algorithm for computing the Tate pairing, assuming an even embedding degree, without using the two-variable approach of [12]. Section 4 shows how to apply various techniques from the literature for speeding up pairing computation to the specific genus 2 case over prime fields. Section 5 gives experimental results, and we draw our conclusions in section 6.

Section 3 was partly presented in a short paper on the ePrint archive (see [15]), and was presented in the rump session at the ECC 2005 conference. We note that some of the ideas in this section were derived independently in Kobayashi et al. [17].

2 The Tate Pairing

We say that a subgroup of the degree zero divisor class group of a hyperelliptic curve C defined over a finite field $\mathbb{F}_p$ has embedding degree k, if the order r of the subgroup divides $p^k - 1$, but does not divide $p^i - 1$ for any $0 < i < k$. The Tate pairing maps the discrete logarithm in the subgroup to the discrete logarithm in the finite field $\mathbb{F}_{p^k}$, which is the basis of the Frey-Rück attack [11].

Using the notation above, let r be a prime number which is coprime to p. Let $G = J_C(\mathbb{F}_{p^k})$ be the Jacobian Variety of C over $\mathbb{F}_{p^k}$, which is isomorphic to the degree zero divisor class group, and let $G[r]$ be the r-torsion group and G/rG the quotient group. Then the Tate pairing is defined as;

$$\langle \cdot, \cdot \rangle_r : G[r] \times G/rG \rightarrow \mathbb{F}_{p^k}^* / (\mathbb{F}_{p^k}^*)^r$$

We follow Galbraith et al. [12] in defining the first argument over the smaller field $\mathbb{F}_p$ instead of $\mathbb{F}_{p^k}$. This greatly improves computational efficiency, as all the coefficients of the functions in Miller's Algorithm will then also be defined over the field $\mathbb{F}_p$. To allow for an efficient implementation of the extension field arithmetic, we also assume that the embedding degree k is even. The Tate pairing as detailed above is only defined up to r-th powers. As a unique value is required for cryptographic purposes, we define the *reduced* pairing;

$$\langle D_1, D_2\rangle_r^{(p^k-1)/r}.$$

The Tate pairing is both well-defined and non-degenerate. However its most important property is *bilinearity*. We define bilinearity for any integer n as, $\langle [n]P, Q\rangle \equiv \langle P, [n]Q\rangle \equiv \langle P, Q\rangle^n$ (modulo r^{th} powers). The Tate pairing can be computed using an algorithm due to Miller [20,21], as described in Algorithm 1 for an arbitrary hyperelliptic curve. This algorithm is basically the usual "double and add" algorithm combined with an evaluation of certain intermediate functions (see chapter 9 of [4] for more details).

In Algorithm 1 the divisions are postponed until the end of the loop, to avoid performing a division each loop iteration. To do this we use two variables in the loop, to effectively replace a division with a squaring each loop iteration, which is considerably less expensive to compute. This is an idea given by Galbraith et. al. [12]. However as we shall see in the next section, when the embedding degree is even, this optimisation is unnecessary.

After the main loop, the final exponentiation of $(p^k - 1)/r$ is performed to obtain a unique value over $\mathbb{F}_{p^k}$. It is this unique value which can then be used for cryptographic purposes. It is well known that if arithmetic in $\mathbb{F}_{p^k}$ is implemented using quadratic extensions, an element in this field can be exponentiated to the power of $p^{k/2}$ using a simple conjugation in $\mathbb{F}_{p^k}$ over $\mathbb{F}_{p^{k/2}}$. Conjugation in the quadratic extension is denoted by $\overline{x} = (a - bi)$ for an element $x = (a + bi) \in \mathbb{F}_{p^k}$.

Taking advantage of this, it is standard to efficiently compute the final exponentiation as $f = \overline{f}/f$ followed by $f = f^{(p^{k/2}+1)/\phi_k(p)}$ and $f = f^{(\phi_k(p)/r)}$, where $\phi_d(x)$ is the d^{th} cyclotomic polynomial. The final $f = f^{(\phi_k(p)/r)}$ exponentiation can be computed efficiently using either Lucas Sequences [28,29] or the multi-exponentiation approach [16,14].

An important improvement on Miller's Algorithm as detailed above was given in [2] for elliptic curves. If the x-coordinate of the image point Q is defined over a subfield of $\mathbb{F}_{p^k}$, then the denominator, or the f_d variable in algorithm 1, will also be defined over a subfield of $\mathbb{F}_{p^k}$. This is because the denominator function relies solely on the evaluation at the x-coordinate of Q each iteration. As any value defined over a subfield of $\mathbb{F}_{p^k}$ will be annihilated by the final exponentiation of $(p^{k/2} - 1)$, the f_d variable can be removed from algorithm 2 completely.

Some distortion maps naturally map the x-coordinate of a point to a subfield of $\mathbb{F}_{p^k}$. When this is not the case, a simple transformation of the point as detailed in the next section will have the desired effect. However, as will be seen, this approach is problematic for curves of genus greater than one.

3 Eliminating Divisions in Miller's Algorithm

In this section, we show how the denominator elimination technique is problematic with curves of genus greater than one, assuming that a suitable distortion

Algorithm 1. Miller's algorithm to compute the Tate pairing, as per Galbraith et al. [12]

```
INPUT: $P \in J_C(\mathbb{F}_p), Q \in J_C(\mathbb{F}_{p^k})$ where $P$ has order $r$.
OUTPUT: $\langle P, Q \rangle_r^{(q^k-1)/r}$
 1: $f_c \leftarrow 1, f_d \leftarrow 1$
 2: $T \leftarrow P$
 3: for $i \leftarrow \lfloor \log_2(r) \rfloor - 1$ downto $0$ do
 4:     ▷ Compute $T' = [2]T - div(c/d)$
 5:     $T \leftarrow [2]T$
 6:     $f_c \leftarrow f_c^2 \cdot c(Q)$
 7:     $f_d \leftarrow f_d^2 \cdot d(Q)$
 8:     if $r_i = 1$ then
 9:         ▷ Compute $T' = T + P - div(c/d)$
10:         $T \leftarrow T + P$
11:         $f_c \leftarrow f_c \cdot c(Q)$
12:         $f_d \leftarrow f_d \cdot d(Q)$
13:     end if
14: end for
15: $f \leftarrow f_c/f_d$
16: $f \leftarrow ((\overline{f}/f)^{(p^{k/2}+1)/\phi_k(p)})^{(\phi_k(p)/r)} = f^{(p^k-1)/r}$
17: return $f$
```

map does not exist. We then present a more efficient algorithm for computing the Tate pairing over quadratic extension fields than is given in algorithm 1, which overcomes the problems associated with denominator elimination in certain contexts.

As seen in the previous section, another approach is required to get the denominator elimination technique to work, when no distortion map exists that maps the x-coordinate of a point from the ground field to a subfield of the field $\mathbb{F}_{p^k}$. Instead we must use an idea given in a paper by Barreto et. al. [3]. To apply denominator elimination in this case, generate a "distorted point" Q over $\mathbb{F}_{p^k}$ and get a trace-zero point with: $R = Q - Q^{p^{k/2}}$. The point R will have an x-coordinate defined over a subfield, and so the denominator elimination technique can be used.

However, this technique is problematic when the genus is greater than one, as it increases the weight of the image divisor. For example, if we are evaluating at a degenerate divisor in the genus 2 setting, which consists of a single point on the support, then the above mapping will result in a more general divisor with two points. This will not happen using elliptic curves, as each class in the divisor class group can be represented by a divisor with a single point in the support. So in the genus 2 setting, instead of evaluating the functions in Miller's algorithm at a single point, they must be evaluated at two points to use the denominator elimination technique. This drastically reduces the efficiency of denominator elimination.

We now present an alternative way to proceed, by introducing a new variant of Miller's algorithm, assuming that the embedding degree of the curve is even,

Table 1. Complexity of function calculation per iteration in Miller's Algorithm

case	description	complexity
1	Original Approach	1I, 2M, 1S
2	Two-variable Approach	2M, 2S
3	Algorithm 2	2M, 1S
4	Denominator Elimination	1M, 1S

as is almost always the case for practical implementations. The finite field $\mathbb{F}_{p^k}$ is then typically represented as a quadratic extension of $\mathbb{F}_{p^{k/2}}$. It is well known that for an element $x = (a + bi) \in \mathbb{F}_{p^k}$, then $(\frac{1}{x})^{p^{k/2}-1} = (\overline{x})^{p^{k/2}-1}$. This effectively replaces an expensive operation with one that is free to compute.

This technique is exploited by Scott [27] when computing the Weil pairing. Scott proposes exponentiating the pairing value to the power of $p^{k/2} - 1$, which means that the inversion in the Miller loop can be replaced with a conjugation. However, no one has observed that it is possible to use this idea when computing the Tate pairing, without any extra computation being involved. When computing the Tate pairing, the final exponentiation to obtain a unique r^{th} root of unity includes the factor $(p^{k/2} - 1)$, as $(p^k - 1)/r = (p^{k/2} - 1)(p^{k/2} + 1)/r$. So as the output of the loop is implicitly raised to the power of $(p^{k/2} - 1)$, there is no need of the strategy of using two variables to eliminate divisions, as a division in the main loop can be replaced by a multiplication and a conjugation. The new algorithm is described in Algorithm 2.

As the variable f_d is eliminated from the pairing calculation, the saving is a squaring over $\mathbb{F}_{p^k}$ each iteration of the loop compared to the GHS approach. This is still not as efficient as performing denominator elimination, which would save a multiplication over this again each iteration. However, when computing the Tate pairing with curves of genus greater than one, and using a distortion map that does not allow denominator elimination directly, algorithm 2 is a slightly more efficient algorithm.

The reason for this is that the denominator elimination algorithm consists of two evaluations at the line function each iteration (or one evaluation of a more complicated form if Mumford representation (see Cantor [6]) is used). Algorithm 2 consists of one evaluation at the line function, and one evaluation at the vertical line function, which requires less computation to evaluate than the line function. Algorithm 2 is also less restrictive than using denominator elimination, as it places no conditions on the form of the image divisor. Table 1 illustrates the complexity of the different algorithms in more detail.

4 Implementing the Pairing

In this section, various techniques that allow for an efficient implementation of the Tate pairing using supersingular genus 2 curves over prime fields are described. Timings are given in section 5.

Algorithm 2. An improved algorithm for computing the Tate Pairing

```
INPUT: $P \in J_C(\mathbb{F}_p), Q \in J_C(\mathbb{F}_{p^k})$ where $P$ has order $r$.
OUTPUT: $\langle P, Q\rangle_r^{(q^k-1)/r}$
 1: $f \leftarrow 1$
 2: $T \leftarrow P$
 3: for $i \leftarrow \lfloor \log_2(r) \rfloor - 1$ downto 0 do
 4:     ▷ Compute $T' = [2]T - div(c/d)$
 5:     $T \leftarrow [2]T$
 6:     $f \leftarrow f^2 \cdot c(Q) \cdot \overline{d(Q)}$
 7:     if $r_i = 1$ then
 8:         ▷ Compute $T' = T + P - div(c/d)$
 9:         $T \leftarrow T + P$
10:         $f \leftarrow f \cdot c(Q) \cdot \overline{d(Q)}$
11:     end if
12: end for
13: $f \leftarrow ((\overline{f}/f)^{(p^{k/2}+1)/\phi_k(p)})^{(\phi_k(p)/r)} = f^{(p^k-1)/r}$
14: return $f$
```

4.1 The Curve

Following [7] and [8], we implement the Tate pairing on the supersingular genus 2 curve;

$$H : y^2 = x^5 + a,\ a \in \mathbb{F}_p^*,\ p \equiv 2, 3 \mod 5$$

In practice, we take $a = 1$ for convenience. The other supersingular genus 2 curve defined over a prime field with a low embedding degree that was given in [7], is unsuitable for cryptography as it is isogenous to a product of elliptic curves.

The order of the Jacobian of this curve is $\#J_C(\mathbb{F}_p) = p^2 + 1$, and hence the embedding degree of the curve is 4. The *distortion map* that maps points on the curve defined over the field $\mathbb{F}_p$ to the larger field $\mathbb{F}_{p^4}$ is given as;

$$\phi(x, y) = (\zeta_5 x, y)$$

where ζ_5 is a primitive 5^{th} root of unity defined over $\mathbb{F}_{p^4}$. Note that ζ_5 maps the x-coordinate to $\mathbb{F}_{p^4}$, and hence does not give denominator elimination directly. Choie and Lee give explicit formulae [8] for calculating the functions required for the genus 2 Tate pairing, which are derived from Lange's explicit genus 2 formulae [18] and Miyamoto et al.'s formulae [22].

Doubling a divisor is by far the most important part of the group arithmetic for pairings, under the assumption that we are using a prime-order subgroup which has an order of low Hamming Weight. In [8] the given cost of doubling a general divisor (in the overwhelmingly common case) and extracting the functions required for Miller's algorithm is 1 inversion, 23 multiplications and 5 squarings over $\mathbb{F}_p$. However, we save a squaring and a multiplication over this. In table 6 of the appendix, we give the formulae for doubling a general divisor as per [8], with these

Table 2. Comparison of the cost of doubling in J_H

	doubling	l(x)
[22]	1I, 23M, 4S	3M
[18]	1I, 22M, 5S	3M
[8]	1I, 23M, 5S	no cost
our work	1I, 22M, 4S	no cost

optimisations built in (the multiplication is saved in step 8). Note our assumption that, as the characteristic of the field is odd, the h polynomial is zero, where the h polynomial comes from the definition of the curve as $y^2 + h(x)y = f(x)$. Table 2 summarises the computational cost of doubling a general divisor.

We believe that the formulae in table 6 are optimal, as they have the same computational cost as simply doubling a divisor as given in [18], ie. calculating the functions required for Miller's algorithm is for free.

4.2 Prime-Order Subgroup

We use the conventions suggested by Lenstra and Verheul [19] and used by Scott [27], to define the levels of security required. For a more thorough comparison of security levels we refer the reader to Galbraith et al. [13]. Here the security levels are defined as (160/1024), (192/2048) and (224/4096), where the first number in each term is the group size, and the second number is the size of the field $\mathbb{F}_{p^k}$. As our embedding degree is $k = 4$, we are required to work with finite fields $\mathbb{F}_p$, where $p \sim 256$, 512 and 1024-bits.

When considering what group size to use there are two options, either to use a prime-order subgroup or the full order of the Jacobian. The latter has the advantage that the order of the Jacobian often has a small Hamming weight, and the final exponentiation can be far less expensive. However, if we choose the order of the prime-order subgroup such that it has a low Hamming weight, and if it is far smaller than the order of the Jacobian, then the former method is better.

Rather than using a random prime-order subgroup, we choose a special prime of low Hamming Weight known as a *Solinas* prime [30]. These primes require only two additions in Miller's algorithm. As Duursma and Lee noted [9], the final addition can be skipped assuming that denominator elimination is applied. See section 5 for further details on the parameters used.

4.3 Finite Field Representation

The best way to represent elements of the field $\mathbb{F}_{p^4}$ is to represent them as a quadratic extension of $\mathbb{F}_{p^2}$, which is in turn a quadratic extension of $\mathbb{F}_p$, as opposed to representing $\mathbb{F}_{p^4}$ as a quartic extension of $\mathbb{F}_p$. If the prime p is congruent to 5 mod 8, then the irreducible polynomial $x^2 + 2$ can be used to represent the quadratic extension field $\mathbb{F}_{p^2}$. So, assuming that $\beta = -2$ is a quadratic non-residue, we represent elements of the field $\mathbb{F}_{p^2}$ as $(a + b\sqrt{\beta})$, where $a, b \in \mathbb{F}_p$, and we represent elements of the field $\mathbb{F}_{p^4}$ as $(c + d\sqrt[4]{\beta})$, where $c, d \in \mathbb{F}_{p^2}$. An advantage of using a prime $p \equiv 5 \mod 8$ is the resulting simple formula for modular square roots, as required for generating points on the curve.

Using our representation, a multiplication of two elements in $\mathbb{F}_{p^2}$ is computed using the Karatsuba method as

$$\begin{aligned}(a + b\sqrt{\beta})(x + y\sqrt{\beta}) &= (ax - 2by + (ay + bx)\sqrt{\beta}) \\ &= (ax - 2by + ((a + b)(x + y) - ax - by)\sqrt{\beta})\end{aligned}$$

which takes $3M$, where M is a multiplication over $\mathbb{F}_p$. Similarly, a multiplication of two elements in $\mathbb{F}_{p^4}$ takes $9M$. Squaring an element in $\mathbb{F}_{p^2}$ is computed as

$$\begin{aligned}(a + b\sqrt{\beta})^2 &= (a^2 - 2b^2 + 2ab\sqrt{\beta}) \\ &= ((a + b)(a - 2b) + ab + 2ab\sqrt{\beta})\end{aligned}$$

which takes $2M$ - slightly cheaper than a general multiplication in $\mathbb{F}_{p^2}$. Similarly, squaring an element in $\mathbb{F}_{p^4}$ takes $6M$, which is considerably cheaper than the $9M$ required for a general multiplication in $\mathbb{F}_{p^4}$. We also note that our method for squaring an element in $\mathbb{F}_{p^4}$ is cheaper than the $8M$ given by Choie and Lee [8].

4.4 Using Degenerate Divisors

Duursma and Lee [9] introduced the notion of working with a degenerate divisor for pairing applications with curves of genus greater than 1. In the genus 2 context, we will define a degenerate divisor as a divisor with only one point in its support, rather than the more general two. There is no advantage to be gained in using a degenerate divisor as the first argument to Miller's algorithm, as with the first doubling the divisor will turn into a more general divisor with two points on the support, unless one takes advantage of an automorphism that keeps the divisor in its special shape (eg. see [1]).

However, as we are evaluating the second divisor at a function, we achieve a speedup by evaluating at a degenerate divisor (ie. a single point). Evaluating at a more general divisor requires evaluation at two points, or else using the divisor's Mumford representation. Pairing-based cryptosystems, such as the Identity-Based Encryption scheme of Boneh and Franklin [5], can be easily modified to take advantage of the form of degenerate divisors to speed up the encryption process. However, in this case the decryption process involves pairing two general divisors, so it is important to give timings for both cases. Frey and Lange [10] discuss in more detail in which cases the second argument to Miller's algorithm can be chosen to be a degenerate divisor.

4.5 Evaluating Functions

Each iteration of the loop requires the evaluation of the function $f_c = y_1 - ((x_1\zeta_5)^3 s_1 + (x_1\zeta_5)^2 l_2 + (x_1\zeta_5) l_1 + l_0)$, where s_1, l_0, l_1, l_2 are from Cantor's algorithm, ζ_5 is a primitive 5^{th} root of unity defined over $\mathbb{F}_{p^4}$ and x_1 is the x-coordinate of the point at which we are evaluating. As $(x_1\zeta_5)^3$, $(x_1\zeta_5)^2$ and

$(x_1\zeta_5)$ can be precomputed, this leaves 12 multiplications over $\mathbb{F}_p$ to be computed each time the function is evaluated. However, a multiplication may be saved by examining relations between various powerings of the 5^{th} root of unity.

If ζ_n is a primitive n^{th} root of unity in a field K, then its conjugates over the prime subfield K_0 of K are also primitive n^{th} roots of unity [23]. Also, ζ_n^a is a primitive n^{th} root of unity if and only if a and n are coprime. In our case, the third power of a 5^{th} primitive root of unity over $\mathbb{F}_{p^4}$ is related to the second power by conjugation: $\zeta_5^3 = \overline{\zeta_5^2}$

So instead of evaluating an equation of the form $a+b\zeta_5+c\zeta_5^2+d\zeta_5^3$, where $b = -x_1l_1$, $c = -x_1^2l_2$ and $d = -x_1^3s_1$, let $\zeta_5^2 = (m+n\sqrt[4]{\beta})$ where $m, n \in \mathbb{F}_{p^2}$. Then we can compute $a+b\zeta_5+c\zeta_5^2+d\zeta_5^3$ as $a+b\zeta_5+((c+d)m, (c-d)n)$. Computing c and d takes only two multiplications over $\mathbb{F}_p$ (with a precomputation of 1 squaring and 1 multiplication). Computing $(c+d)m$ and $(c-d)n$ takes 4 multiplications, with a precomputation of 6 multiplications. Computing $b\zeta^5$ takes 4 multiplications, with a precomputation of 4 multiplications. Thus the total multiplication count in evaluating the function is 10 multiplications, a saving of two multiplications, with a precomputation of 11 multiplications and 1 squaring.

Evaluating the image point at the vertical line functions takes 8 multiplications over $\mathbb{F}_p$, assuming a precomputation of 6 multiplications. So the total cost of evaluating a point at the functions is 18 multiplications over $\mathbb{F}_p$ per iteration, with a precomputation of 1 squaring and 17 multiplications.

4.6 Using Denominator Elimination

As detailed in section 3, it is more efficient to use Algorithm 2 than denominator elimination, assuming a distortion map that does not give denominator elimination directly, and that the image divisor is a degenerate divisor. However, it is possible to reduce the performance gap by using customized multiplication routines, as detailed in this section.

Given a point $Q = (x, y) \in \mathbb{F}_{p^4}$, the transformation $R = Q - Q^{p^2}$ gives a divisor R suitable for use with denominator elimination. Writing this as $R = (x, y) + (\overline{x}, -y)$ avoids using Cantor's algorithm over $\mathbb{F}_{p^4}$ and keeps the two points on the support of the divisor separate. A benefit of this approach is that the function calculated in the main doubling loop, $f_c = y_1 - ((x_1\zeta_5)^3s_1 + (x_1\zeta_5)^2l_2 + (x_1\zeta_5)l_1 - l_0)$, can be reused for the calculation of the required function for the second point.

Let the function $f_c = ((a+b\sqrt{\beta})+(c+d\sqrt{\beta})\sqrt[4]{\beta})$ for the first point (x, y). Then, for the second point $(\overline{x}, -y)$, the function $f_c = ((a-2y+b\sqrt{\beta})-(c+d\sqrt{\beta})\sqrt[4]{\beta})$. So the calculation of the second function is effectively for free, as it simply involves two subtractions and a conjugation using the function generated by the first point. However, we still have to multiply the two functions together.

In each iteration of the loop, the function f_c is evaluated twice, ie. at the two points. As seen above, the two functions are closely related. It is possible to exploit these relations to speed-up the calculation. So instead of calculating each function separately and multiplying them separately by the overall accumulating variable, we multiply the functions f_{c_1} and f_{c_2} together first, before multiplying the result with the accumulating function.

Normally, multiplying two general elements in $\mathbb{F}_{p^4}$ can be done with only 9 multiplications over $\mathbb{F}_p$, using the Karatsuba technique. For the functions $f_{c_1} = (a + b\sqrt[4]{\beta})$ and $f_{c_2} = (c - b\sqrt[4]{\beta})$, where $a, b, c \in \mathbb{F}_{p^2}$, the $f_{c_1} f_{c_2}$ multiplication is unrolled as;

$$(a + b\sqrt[4]{\beta})(c - b\sqrt[4]{\beta}) = ac - b^2\sqrt{\beta} + b(c - a)\sqrt[4]{\beta}$$

Note that here $(c - a) \in \mathbb{F}_p$. We can also take advantage of the form of the ac multiplication, where;

$$ac = (e + f\sqrt{\beta})(g + f\sqrt{\beta}) = eg - 2f^2 + f(e + g)\sqrt{\beta}$$

So the total cost is $2M + S$ for the ac multiplication, plus $2M + 2M$ for the overall multiplication, which results in $6M + S$ instead of the general $9M$. When this technique is implemented, we find that although the denominator elimination method is theoretically slightly faster, the performance of denominator elimination and Algorithm 2 is roughly the same, for the genus 2 case under consideration. However, we suggest that Algorithm 2 is a more natural algorithm to use in practice, as it is does not require constructing customised multiplication routines, such as those given in this section.

4.7 Lucas Exponentiation

As detailed in algorithm 1, the final exponentiation is split into two parts. The first part can be computed with a conjugation and division, and then Lucas exponentiation, as detailed in the paper by Scott and Barreto [28], is used for the $(p^2+1)/r$ exponentiation. It is also possible to write the $(p^2+1)/r$ exponentiation to the base p, and take advantage of the Frobenius endomorphism [16]. However, according to Granger et al. [14], it is faster to use the Lucas sequence approach for curves of low embedding degree.

4.8 Coding Issues

We use MIRACL [24] to provide the cryptographic primitives needed. In particular, we make use of special assembly-language routines that MIRACL provides, which can be used when working with moduli with a fixed number of bits. All of the implementation was written in C/C++ and timed on a Pentium IV, 2.8 Ghz.

4.9 Theoretical Analysis

Here we analyse the theoretical cost of computing the Tate pairing. Firstly, we reproduce Choie and Lee's analysis, for the sake of comparison. They estimate the cost of computing the Tate pairing (minus the final exponentiation) as

$$log_2(r)(T_D + T_c + T_d + 2T_{sk} + 2T_{mk}) + 0.5log_2(r)(T_A + T_c + T_d + 2T_{mk})$$

where T_D is the cost of doubling a general divisor, T_A is the cost of adding two general divisors, T_c and T_d the cost of evaluating at the rational functions c and

d, and T_{sk} and T_{mk} the cost of squaring and multiplying respectively in $\mathbb{F}_{p^k}$. Note that the authors assume that there will be $r/2$ additions to be performed, as they employ a random subgroup order with no special conditions.

Let S be a squaring over $\mathbb{F}_p$ and let M be a multiplication over $\mathbb{F}_p$. Choie and Lee then define $T_D = 1I + 23M + 5S$ and $T_A = 1I + 23M + 2S$, using their explicit formulae for calculating the group law, and $T_c + T_d$ as $22M + 5S$, with an initial precompution of $8M + 3S$. Finally, they define $T_{sk} = 8M$ and $T_{mk} = 9M$. The total cost then of computing $\langle P, Q\rangle_r$ given by Choie and Lee, assuming a subgroup order of size $log_2(r) \approx 160$, is $240I + 17688M + 2163S$.

We now give the theoretical cost of computing $\langle P, Q\rangle_r$ using the optimisations given in this paper. Namely, using a more efficient variant of Miller's algorithm, using a subgroup order of low Hamming weight, using more efficient formulae to calculate the group law, evaluating at a degenerate divisor, and implementing finite field arithmetic more efficiently. The cost of computing the Tate pairing in our case is (again without computing the final exponentiation)

$$log_2(r)(T_D + T_c + T_d + T_{sk} + 2T_{mk}) + 2(T_A + T_c + T_d + 2T_{mk})$$

where T_A and T_{mk} are the same as given by Choie and Lee, $T_D = 1I + 22M + 4S$, $T_c + T_d = 18M$ with a precomputation of $17M + 1S$, and $T_{sk} = 6M$. Therefore, the total cost using our optimisations comes to $162I + 10375M + 645S$, a substantial improvement.

5 Experimental Results

In this section, we give experimental results for computing the Tate pairing using the techniques detailed in this paper for the supersingular genus 2 curve defined over $\mathbb{F}_p$, as defined in section 4.1. We will use the three different levels of security defined earlier for testing, namely (160/1024), (192/2048) and (224/4096).

The only condition on our prime-subgroup order r is that it be congruent to 5 mod 8. r, and not r^2, must divide the order of the Jacobian, $p^2 + 1$. The prime p must be congruent to $5 \mod 8$, and also congruent to 2 or $3 \mod 5$, for reasons stated earlier. The parameters used are detailed in appendix A.

Table 3 details the timings for the implementation of the Tate pairing for the (160/1024) security level, table 4 is for the (192/2048) case, and table 5 is for the (224/4096) case. There are four cases in each table. The first is evaluating using a single point. The second case is evaluating at the more general two points, which is the case when the divisor is the sum of two rational points. The third case uses Mumford representation, instead of keeping the points separate. This case has the advantage that it also handles the case when the points on the divisor are defined over a larger field. All these cases use algorithm 2. The fourth cases are timings that are reported in [27] using elliptic curves, with the equivalent level of security.

In the Choie and Lee paper [8], they give implementations of the pairing that range between 500 and 600 ms on a Pentium IV 2 Ghz for a (160/1024) bit security level. However, as seen in table 3, our timings far outperform this. These timings indicate that genus 2 pairings over prime fields are valid candidates for

Table 3. Running times - (160/1024) security level

case	description	pairing time (ms)
1	evaluating at degenerate divisor	16
2	evaluating at general divisor	20.7
3	evaluating using Mumford rep	20.45
4	elliptic curve timing [27]	8.9

Table 4. Running times - (192/2048) security level

case	description	pairing time (ms)
1	evaluating at degenerate divisor	49
2	evaluating at general divisor	62
3	evaluating using Mumford rep	61
4	elliptic curve timing [27]	20.5

Table 5. Running times - (224/4096) security level

case	description	pairing time (ms)
1	evaluating at degenerate divisor	183
2	evaluating at general divisor	232
3	evaluating using Mumford rep	229
4	evaluating at degenerate divisor (denom elim)	175

practical implementations. However, as can be seen from the tables, the elliptic cases are roughly twice as fast as the genus 2 timings. This is what we would expect, due to the more complicated group law in the genus 2 case.

6 Conclusion

We have introduced a new variant of Miller's algorithm to compute the Tate pairing for curves of even embedding degree, by showing how divisions can always be eliminated. This algorithm is not as fast as using denominator elimination in the general case, but can be faster when working with curves of genus greater than one and distortion maps of a certain form.

We have implemented the Tate pairing on supersingular genus 2 curves over prime fields, detailed various optimisations, and showed how genus 2 curves over prime fields are valid candidates for pairing implementation. In particular, our timings are the fastest reported in the literature to date by a considerable margin.

Acknowledgements

We would like to thank Steven Galbraith and Caroline Sheedy for comments on this paper, and Noel McCullagh for pointing out an error in a previous draft. We would also like to thank the anonymous referees for their comments.

References

1. Barreto, P.S.L.M., Galbraith, S.D., Ó hÉigeartaigh, C., Scott, M.: Pairing computation on supersingular abelian varieties. Cryptology ePrint Archive, Report, 2004/375 (2004), Available from http://eprint.iacr.org/2004/375
2. Barreto, P.S.L.M., Kim, H.Y., Lynn, B., Scott, M.: Efficient algorithms for pairing-based cryptosystems. In: Yung, M. (ed.) CRYPTO 2002. LNCS, vol. 2442, pp. 354–368. Springer, Heidelberg (2002)
3. Barreto, P.S.L.M., Lynn, B., Scott, M.: On the selection of pairing-friendly groups. In: Matsui, M., Zuccherato, R.J. (eds.) SAC 2003. LNCS, vol. 3006, pp. 17–25. Springer, Heidelberg (2004)
4. Blake, I.F., Seroussi, G., Smart, N.P.: Advances in elliptic curve cryptography. Cambridge (2005)
5. Boneh, D., Franklin, M.: Identity-based encryption from the weil pairing. SIAM Journal of Computing 32(3), 586–615 (2003)
6. Cantor, D.G.: Computing in the jacobian of a hyperelliptic curve. Mathematics of Computation 48(177), 95–101 (1987)
7. Choie, Y., Jeong, E., Lee, E.: Supersingular hyperelliptic curves of genus 2 over finite fields. Journal of Applied Mathematics and Computation 163(2), 565–576 (2005)
8. Choie, Y., Lee, E.: Implementation of tate pairing on hyperelliptic curves of genus 2. In: Lim, J.-I., Lee, D.-H. (eds.) ICISC 2003. LNCS, vol. 2971, pp. 97–111. Springer, Heidelberg (2004)
9. Duursma, I., Lee, H.-S.: Tate pairing implementation for hyperelliptic curves $y^2 = x^p - x + d$. In: Laih, C.-S. (ed.) ASIACRYPT 2003. LNCS, vol. 2894, pp. 111–123. Springer, Heidelberg (2003)
10. Frey, G., Lange, T.: Fast bilinear maps from the tate-lichtenbaum pairing on hyperelliptic curves. In: Hess, F., Pauli, S., Pohst, M. (eds.) Algorithmic Number Theory VII. LNCS, vol. 4076, pp. 466–479. Springer, Heidelberg (2006)
11. Frey, G., Rück, H.-G.: A remark concerning m-divisibility and the discrete logarithm problem in the divisor class group of curves. Mathematics of Computation 62(206), 865–874 (1994)
12. Galbraith, S., Harrison, K., Soldera, D.: Implementing the tate pairing. In: Fieker, C., Kohel, D.R. (eds.) Algorithmic Number Theory Symposium – ANTS V. LNCS, vol. 2369, pp. 324–337. Springer, Heidelberg (2002)
13. Galbraith, S.D., Paterson, K.G., Smart, N.P.: Pairings for cryptographers. Cryptology ePrint Archive, Report 2006/165 (2006), http://eprint.iacr.org/2006/165
14. Granger, R., Page, D., Smart, N.P.: High security pairing-based cryptography revisited. In: Hess, F., Pauli, S., Pohst, M. (eds.) Algorithmic Number Theory Symposium – ANTS VII. LNCS, vol. 4076, pp. 480–494. Springer, Heidelberg (2006)
15. Ó hÉigeartaigh, C.: Speeding up pairing computation (2005), http://eprint.iacr.org/2005/293
16. Hu, L., Dong, J-W., Pei, D-Y.: Implementation of cryptosystems based on tate pairing. Journal of Computer Science and Technology 20(2), 264–269 (2005)
17. Kobayashi, T., Aoki, K., Imai, H.: Efficient algorithms for tate pairing. IEICE Transactions Fundamentals, E89-A(1) (January 2006)
18. Lange, T.: Formulae for arithmetic on genus 2 hyperelliptic curves. Applicable Algebra in Engineering, Communication and Computing 15(5), 295–328 (2005)

19. Lenstra, A.K., Verheul, E.R.: Selecting cryptographic key sizes. Journal of Cryptology 14(4), 255–293 (2001)
20. Miller, V.S.: Short programs for functions on curves. Unpublished manuscript (1986), http://crypto.stanford.edu/miller/miller.pdf
21. Miller, V.S.: The weil pairing and its efficient calculation. Journal of Cryptology 17(4), 235–261 (2004)
22. Miyamoto, Y., Doi, H., Matsuo, K., Chao, J., Tsuji, S.: A fast addition algorithm of genus two hyperelliptic curve. In: Symposium on Cryptography and Information Security – SCIS 2002, pp. 497–502.
23. Ribenboim, P.: Classical Theory of Algebraic Numbers. Springer, Heidelberg (2001)
24. Scott, M.: Miracl (multiprecision integer and rational arithmetic c/c++ library). Available from http://indigo.ie/~mscott/
25. Scott, M.: Faster identity based encryption. Electronics Letters 40(14), 861 (2004)
26. Scott, M.: Computing the tate pairing. In: Menezes, A.J. (ed.) CT-RSA 2005. LNCS, vol. 3376, pp. 293–304. Springer, Heidelberg (2005)
27. Scott, M.: Scaling security in pairing-based protocols. Cryptology ePrint Archive, Report, 2005/139 (2005), http://eprint.iacr.org/2005/139
28. Scott, M., Barreto, P.: Compressed pairings. In: Franklin, M. (ed.) CRYPTO 2004. LNCS, vol. 3152, pp. 140–156. Springer, Heidelberg (2004)
29. Smith, P., Skinner, C.: A public-key cryptosystem and a digital signature system based on the lucas function analogue to discrete logarithms. In: Safavi-Naini, R., Pieprzyk, J.P. (eds.) ASIACRYPT 1994. LNCS, vol. 917, pp. 357–364. Springer, Heidelberg (1995)
30. Solinas, J.: Generalized mersenne numbers. Technical Report CORR 99-39, University of Waterloo (1999), Available from http://www.cacr.math.uwaterloo.ca/techreports/1999/corr99-39.pdf

A Curve Parameters

Here we give the subgroup and prime field parameters that were used for the different security levels;

$r = 2^{159} + 2^{17} + 1$
$p = 63324531451181148200275171731203125718855624493339065310878459331886717065893$

192/2048 security level:

$r = 2^{191} + 2^{2} + 1$
$p = 8928465122808378842689950368414551548287912471534562510973748060201641117468953363599067244027908076232256944699958875614648564192943960634648749730387013$

224/4096 security level:

$r = 2^{223} + 2^{13} + 1$
$p = 155722884131515840187323558851704700783145211009055018661797977213059964066609221691524801350598779752866480421078369507449219791754684643397404851273095293761493705843127836052457915167872334351960770506641541305942224943595487772602516676106413200532581353024750990143717859982402535061826066311255496083453$

B Doubling Formulae

Table 6. Formulae for doubling when deg $u_1 = 2$, gcd$(u_1, 2v_1) = 1)$

Input Output	$D_1 = [u_1, v_1]$ where $u_1 = x^2 + u_{11}x + u_{10}, v_1 = v_{11}x + v_{10}, f = x^5 + a$ $D_3 = [u_3, v_3], l(x)$ such that $D_3 + div((y - l)/u_3) = 2D_1$.	
Step	Expression	Cost
1	Compute $\tilde{v_1} \equiv (2v_1)(\mod u_1) = \tilde{v_{11}}x + \tilde{v_{10}}$ $\tilde{v_{11}} = 2v_{11}, \tilde{v_{10}} = 2v_{10}$	
2	Compute $r = res(u_1, \tilde{v_1})$ $w_0 = \tilde{v_{11}}^2, w_1 = u_{11}^2, w_2 = 4w_0, w_3 = u_{11}\tilde{v_{11}}$, $r = u_{10}w_2 + \tilde{v_{10}}(\tilde{v_{10}} - w_3)$	2S + 3M
3	Compute almost inverse of $inv' = r(2v_1)^{-1}(\mod u_1)$ $inv'_1 = -\tilde{v_{11}}, inv'_0 = \tilde{v_{10}} - w_3$	
4	Compute $k' = \frac{F - v_1^2}{u_1}(\mod u_1) = k'_1x + k'_0$ $w_3 = w_1, w_4 = 2u_{10}, k'_1 = 2w_1 + w_3 - w_4$ $k'_0 = u_{11}(2w_4 - w_3) - w_0$	1M
5	Compute $s' = k'inv'(\mod u_1)$ $w_0 = k'_0inv'_0, w_1 = k'_1inv'_1$ $s'_1 = \tilde{v_{10}}k'_1 - \tilde{v_{11}}k'_0, s'_0 = w_0 - u_{10}w_1$ If $s'_1 = 0$ then goto step $6'$.	5M
6	Compute $s = s_1x + s_0$ and s_1^{-1} $w_1 = (rs'_1)^{-1}, w_2 = s'_1w_1, w_3 = r^2w_1$, $s_1 = s'_1w_2, s_0 = s'_0w_2$	1I, 1S, 5M
7	Compute $l(x) = su_1 + v_1 = s_1x^3 + l_2x^2 + l_1x + l_0$ $l_2 = s_1u_{11} + s_0, l_0 = s_0u_{10} + v_{10}$ $l_1 = (s_1 + s_0)(u_{11} + u_{10}) - s_1u_{11} - s_0u_{10} + v_{11}$	3M
8	Compute $u' = monic(\frac{F - l^2}{u_1^2}) = x^2 + u_{31}x + u_{30}$ $u_{30} = w_3(2v_{11} + w_3(2u_{11} + s_0^2)$ $u_{31} = 2s_0 - w_3$	1S + 2M
9	Compute $v_3 = -l(\mod u_3) = v_{31}x + v_{30}$ $w_1 = u_{31}, u_{31} = w_3u_{31}, w_3 = l_2 - w_1, w_3 = u_{30}w_2$ $v_{31} = (u_{31} + u_{30})(w_2 + s_1) - w_3 - w_1 - l_1, v_{30} = w_3 - l_0$	3M
		1I, 4S, 22M
6'	Compute $l(x) = s_0u_1 + v_1$ $inv = 1/r, s_0 = s'_0inv, l_1 = s_0u_{11} + v_{11}, l_0 = s_0u_{10} + v_{10}$	1I + 3M
7'	Compute $u_3 = monic(\frac{F - l^2}{u_1^2}) = x + u_{30}$ $u_{30} = -2u_{11} - s_0^2$	1S
8'	Compute $v_3 = -l(\mod u_3) = v_{30}$ $v_{30} = u_{30}(l_1 - u_{30}s_0) - l_0$	2M
		1I, 3S, 14M

Efficient Divisor Class Halving on Genus Two Curves

Peter Birkner

Department of Mathematics, Technical University of Denmark (DTU)
Matematiktorvet, Building 303, DK-2800 Kongens Lyngby, Denmark
peter@mat.dtu.dk

Abstract. Efficient halving of divisor classes offers the possibility to improve scalar multiplication on hyperelliptic curves and is also a step towards giving hyperelliptic curve cryptosystems all the features that elliptic curve systems have. We present a halving algorithm for divisor classes of genus 2 curves over finite fields of characteristic 2. We derive explicit halving formulae from a doubling algorithm by reversing this process. A family of binary curves, that are not known to be weak, is covered by the proposed algorithm. Compared to previous known halving algorithms, we achieve a noticeable speed-up for this family of curves.

Keywords: hyperelliptic curve, divisor class halving, binary fields.

1 Introduction

Since hyperelliptic curve cryptosystems (HECC) gain similar attention as their elliptic counterparts, it is very interesting to investigate, whether ideas and methods can be transferred from the elliptic to the hyperelliptic case. The most important operation used by elliptic curve cryptosystems (ECC) is scalar multiplication which is composed of point addition and doubling (when using an double-and-add algorithm) or point addition and halving (when using an half-and-add algorithm [7,11]). These operations are well investigated and it is likely that the present formulae are the most efficient ones. For HECC explicit formulae for addition, doubling and hence scalar multiplication of divisor classes are also known [1,8].

Efficient halving of divisor classes offers the possibility to improve scalar multiplication on hyperelliptic curves and is also a step towards giving HECC all the features that ECC have. Halving a divisor class of a hyperelliptic curve is the reverse operation to doubling, i. e. given a divisor class $\overline{D}$ one computes another divisor class $\overline{E}$ such that $2\overline{E} = \overline{D}$ or written in slightly informal notation: $\frac{1}{2}\overline{D} = \overline{E}$.

In this paper we present an efficient divisor class halving algorithm for hyperelliptic curves of genus 2 over finite fields of characteristic 2. Covering a large family of curves, that are of cryptographic interest, the complexity of our algorithm (1I, 8M, 5SR, 2S, 1HT, 1TR)[1] is only less higher than the complexity of

[1] To describe the complexity of an algorithm we use the following abbreviations: I – Inversion, M – Multiplication, S – Squaring, SR – Square Root, TR – Trace, HT – Half Trace.

© Springer-Verlag Berlin Heidelberg 2007

the fastest doubling algorithm (1I, 5M, 6S) for divisor classes [9]. The proposed halving method is based on explicit doubling formulae [9] that we use to develop the halving formulae.

The first divisor class halving algorithm for binary curves was proposed by Kitamura, Katagi and Takagi [5,6]. To our knowledge this is the only result on halving of divisor classes so far. Their method covers the general case as well as some exceptional cases which do occur with very low probability. In the general case their complexity is 1I, 18M, 2SR, 2S, 2HT, 2TR in the best case and 1I, 21M, 3SR, 2S, 2HT, 2TR in the worst case.

The special class of curves considered in Appendix D of [6] is covered by our study. However, instead of having a one-parameter family our curves has two free parameters. In the worst case our complexity is 1I, 8M, 5SR, 2S, 1HT, 1TR which is significantly faster even than their formulae.

The remainder of this paper is structured as follows: In the next section we recall some important notions and make a classification of genus 2 curves. In Section 3 we show how we developed the halving formulae by reversing the doubling formulae. The following section contains the actual halving algorithm.

2 Basic Notations and Preliminaries

In this section we briefly recall the definitions of hyperelliptic curves, divisor class groups and the Mumford representation. We also make a classification of hyperelliptic curves over finite fields of characteristic 2 because we focus on a family of curves in this paper. Within the halving algorithm we need to solve a quadratic equation in a finite field of even characteristic. So we will explain how to compute solutions for this at the end of this section.

A comprehensive source for the mathematics of finite fields is [10]. For background on hyperelliptic curves we refer the interested reader to [1], from which the following definitions and notations are taken.

Definition 1 (Hyperelliptic curve). *Let K be a field and let $\overline{K}$ be the algebraic closure of K. A curve C, given by an equation of the form*

$$C : y^2 + h(x)y = f(x), \tag{1}$$

where $h \in K[x]$ is a polynomial of degree at most g and $f \in K[x]$ is a monic polynomial of degree $2g + 1$, is called a hyperelliptic curve of genus g over K *if no point on the curve over $\overline{K}$ satisfies both partial derivatives $2y + h = 0$ and $f' - h'y = 0$.*

The last condition ensures that the curve is nonsingular. In this paper we concentrate on hyperelliptic curves of genus 2 over finite fields of characteristic 2. In this case we need a non-zero polynomial h in the curve equation as will be shown now according to [1, p. 309].

Assuming $h = 0$, the partial derivative for y is equal to zero and the one for x is equal to $f'(x) = 0$, which has $2g$ roots in $\overline{K}$. Let x_1 be one of them. Then we can find an element $y_1 \in \overline{K}$ such that $f(x_1) = y_1^2$ which leads to a singular point $P = (x_1, y_1)$ satisfying the curve equation and both partial derivatives. Hence, there is no hyperelliptic curve with $h = 0$ over a field of characteristic 2.

Definition 2 (Divisor class group). *Let C be a hyperelliptic curve of genus g over a field K. The group of degree zero divisors of C is denoted by* Div_C^0*. The quotient group of* Div_C^0 *by the group of principal divisors of C is called the* divisor class group of C *and is denoted by* Pic_C^0*. It is also called the* Picard group of C.

Theorem 1 (Mumford). *Let C be a hyperelliptic curve of genus g over an arbitrary field K. Each nontrivial divisor class of C over K can be represented by a unique pair of polynomials $u, v \in K[x]$, where*

1. *u is monic,*
2. $\deg v < \deg u \leq g$,
3. *$u \mid v^2 + vh - f$.*

Our proposed halving algorithm in Section 4 expects the input divisor class to be in Mumford representation and works directly on the coefficients of the polynomials u and v. The resulting divisor class is also given in the Mumford form.

2.1 Classification of Genus Two Curves

In this paper we deal with hyperelliptic curves of genus 2 over $\mathbb{F}_{2^d}$. To avoid Weil descent attacks [4] one usually restricts to prime degree field extensions for cryptographic applications. So in the following we particularly assume d to be odd.

The genus 2 curves over $\mathbb{F}_{2^d}$ can be sorted into three different categories depending on the 2-rank of the divisor class group of the curve (see [3]). All points P in the support of a divisor class of order 2 satisfy $P = \iota(P)$, where ι is the hyperelliptic involution. If $P = (x, y)$ is not the point at infinity, its coordinates must satisfy $y = h(x) + y$. So, x is a root of $h(x)$ and we have that the degree of h equals the 2-rank of Pic_C^0. In this paper we focus on curves whose divisor class group has 2-rank equal to one, i. e. in the curve equation we have $\deg h = 1$. In [2,1] these curves are called curves of Type II. Over a field $\mathbb{F}_{2^d}$ with d odd, one can perform the following transformations

$$x \mapsto \mu^2 x' + \lambda \quad \text{and} \quad y \mapsto \mu^5 y' + \mu^4 \alpha x'^2 + \mu^2 \beta x' + \gamma,$$

where μ is such that $\mu^3 = h_1,\ \lambda = h_0 h_1^{-1},\ \alpha = \sqrt{\lambda + f_4},\ \beta$ a root of $x^2 + h_1 x + f_2 + f_3\lambda + \varepsilon h_1^2$ with $\varepsilon = \mathrm{Tr}\left((f_2 + f_3\lambda)h_1^{-2}\right)$ and $\gamma = (\lambda^2 f_3 + \lambda^4 + f_1)h_1^{-1}$, to obtain a unique representative of each isomorphism class given by

$$C : y^2 + xy = x^5 + f_3 x^3 + f_2 x^2 + f_0, \tag{2}$$

where $f_2 \in \mathbb{F}_2$ and $f_0, f_3 \in \mathbb{F}_{2^d}$ [1, Proposition 14.37]. So, for the remainder of this paper we consider (2) as a Type II curve.

Since $f_2 \in \mathbb{F}_2$, we have $2 \cdot 2^d \cdot 2^d = 2^{2d+1}$ different choices for the right-hand side of (2), i. e. Type II covers (up to isomorphism) 2^{2d+1} different curves of genus 2 where our halving algorithm can be applied.

2.2 Quadratic Equations in $\mathbb{F}_{2^d}$

In the halving algorithm we need to solve a quadratic equation in $\mathbb{F}_{2^d}$. Provided a solution exists, we can use a simple formula to compute it.

Consider the quadratic equation $X^2 + aX + b = 0$ over the finite field $\mathbb{F}_{2^d}$. By substituting X by X/a, one gets the simpler equation

$$T^2 + T = c, \quad \text{with } c = b/a^2, \tag{3}$$

which has a solution in the field $\mathbb{F}_{2^d}$ if and only if the trace of c is equal to zero [1, Section 11.2.6]. If d is odd, then a solution of (3) is given by

$$t = \sum_{i=0}^{(d-3)/2} c^{2^{2i+1}}. \tag{4}$$

When t is one solution of the quadratic equation (3), then $t + 1$ is the other one. See [1, Section 11.2.6] for details.

2.3 Choice of the Field Representation

Like in the doubling formulae we have to compute inversions, multiplications and squarings to halve a divisor class. Additionally we need to be able to efficiently compute square roots, traces and half-traces. In order to speed these operations up, one can use a normal basis representation. Having this we can compute the square of a field element simply by shifting the representing vector. Computing a square root works the same way but shifting to the opposite direction. Because traces and half-traces are sums of powers of squares, they can be calculated very efficiently, too. In a hardware implementation, multiplications and inversions in the field can be hard-coded in order to get best performance.

Software libraries like NTL work with a polynomial basis representation and do not provide efficient square root computations in characteristic two. So, we implemented our own square root function for the finite field $\mathbb{F}_{2^{83}}$ and present some timings for field operations. We used the Number Theory Library (NTL 5.4) together with the GNU Multiple Precision Arithmetic Library (GMP 4.2.1) and the GNU Compiler Collection (GCC 4.0.1) on an Apple MacBook with a 2,0 GHz Intel Dual Core CPU to compute the benchmarks. The multiplication, inversion and squaring functions are taken from NTL, the square root function is our own implementation for that particular finite field. We measured the time for 100,000 operations each.

Table 1. Timings of field operations in $\mathbb{F}_{2^{83}}$

Operation	Time [sec.]	# of Multiplications
M	0.065966	1.00
I	0.52136	7.90
SR	0.3775	5.72
S	0.045714	0.69

3 From Doubling to Halving

In this section we derive the halving formulae from the doubling formulae. Therefore, we present first how to double a divisor class given in Mumford representation using explicit formulae. Then we explain how we found the halving formulae by reversing the doubling algorithm.

In the entire section we assume C to be a Type II hyperelliptic curve of genus 2 over $\mathbb{F}_{2^d}$, where d is odd, given by equation (2). In the following we also need to assume that the group order of $\mathrm{Pic}^0_C(\mathbb{F}_{2^d})$ is $2r$, where r is odd.[2] For cryptographic applications one wants to work in a cyclic subgroup of prime order l. So we denote the order l-subgroup of $\mathrm{Pic}^0_C(\mathbb{F}_{2^d})$ by S. Note, however, that the following considerations also hold muta mutandis in the subgroup of order r.

3.1 Doubling of Divisor Classes

Let $\overline{E} = [x^2 + u_1'x + u_0',\, v_1'x + v_0']$ be a divisor class in the order l-subgroup S of $\mathrm{Pic}^0_C(\mathbb{F}_{2^d})$. Because l is prime, there exists no proper subgroup of S and hence it is cyclic. So, each divisor class contained in S is the double of another divisor class in the same subgroup, i. e. each divisor class in S can be doubled and hence also be halved. In [5] the elements of this subgroup are called *proper divisor classes.*

We can compute the doubled divisor class $\overline{D} = 2\overline{E} = [x^2+u_1x+u_0,\, v_1x+v_0]$ using Lange and Steven's explicit formulae (see [9]):

$$u_1 = \left(\frac{{u_0'}^2}{f_0 + {v_0'}^2}\right)^2, \tag{5}$$

$$u_0 = \left(\left({u_1'}^2 + f_3\right)\left(\frac{{u_0'}^2}{f_0 + {v_0'}^2}\right) + u_1'\right)^2 + \left(\frac{{u_0'}^2}{f_0 + {v_0'}^2}\right), \tag{6}$$

$$v_0 = \left(\frac{{u_0'}^2}{f_0 + {v_0'}^2} + {u_1'}^2 + f_3\right) u_0 + {u_0'}^2, \tag{7}$$

[2] This ensures that there is no element of order 4.

$$v_1 = \left(\frac{{u'_0}^2}{f_0 + {v'_0}^2} + {u'_1}^2 + f_3\right)\left({u'_1}^2 + f_3\right)\left(\frac{{u'_0}^2}{f_0 + {v'_0}^2}\right) + \left(\frac{{u'_0}^2}{f_0 + {v'_0}^2}\right)u_1 + f_2 + {v'_1}^2. \tag{8}$$

Notice that we are considering a curve of form (2), i. e. $h(x) = x$. Hence, the coefficient h_1 occurring in Lange and Steven's formulae equals one and does not appear here.

3.2 Halving of Divisor Classes

Now, we turn around and compute the half of a divisor class by applying the doubling formulae in reverse order. Given a divisor class $\overline{D} = [x^2 + u_1x + u_0,\ v_1x + v_0]$, we show how to compute a divisor class $\overline{E} = [x^2 + u'_1x + u'_0,\ v'_1x + v'_0]$ such that $\overline{D} = 2\overline{E}$. Therefore, we need again to say that $\overline{D}$ must be contained in the order l-subgroup S of $\mathrm{Pic}^0_C(\mathbb{F}_{2^d})$ in order to ensure that the double and the half of each divisor class does exist in this particular subgroup.

Taking $\sqrt{u_1}$ from (5), we can write u_0 using (6) as

$$u_0 = \left(({u'_1}^2 + f_3)\sqrt{u_1} + u'_1\right)^2 + \sqrt{u_1}. \tag{9}$$

Now, using the fact that we are in characteristic 2 we can expand the quadratic expression and arrange the terms such that we get a quartic equation in u'_1 on the right-hand side:

$$u_0 = {u'_1}^4u_1 + {u'_1}^2 + \left(f_3^2u_1 + \sqrt{u_1}\right). \tag{10}$$

Substituting $U = {u'_1}^2$ yields a quadratic equation in the variable U:

$$U^2u_1 + U = u_0 + f_3^2u_1 + \sqrt{u_1}. \tag{11}$$

Multiplying both sides by u_1 and substituting $T = U \cdot u_1$ afterwards yields a quadratic equation $T^2 + T = c$ where $c = u_1u_0 + f_3^2u_1^2 + u_1\sqrt{u_1}$ like (3).

Because $\overline{D}$ is an element of the subgroup S, there exists an element $\overline{E} \in S$ with $\overline{D} = 2\overline{E}$. So u_1, u_0, v_1 and v_0 can be written as in (5), (6), (7) and (8). Hence, we know that there exists a solution of $T^2 + T = c$ (because this equation holds if and only if (6) holds). Due to Section 2.2 this implies that the trace of c is equal to zero. We also know that there exist two solutions t and $t + 1$ which can be computed using (4). After adjusting these two solutions by dividing by u_1 we have two solutions of (11). Re-substituting $u'_1 = \sqrt{t/u_1}$ or $u'_1 = \sqrt{(t+1)/u_1}$ respectively yields two possible values for u'_1 in (10). We will show how to figure out which of these two solutions is the right one at the end of this section. For now let us suppose that we already know the correct u'_1.

Taking v_0 from (7) and writing again $\frac{{u'_0}^2}{f_0+{v'_0}^2}$ as $\sqrt{u_1}$, we obtain:

$$v_0 = \left(\sqrt{u_1} + {u'_1}^2 + f_3\right)u_0 + {u'_0}^2, \tag{12}$$

which leads us to a new expression for u'_0 using the already known value u'_1:

$$u'_0 = \sqrt{v_0 + \left(\sqrt{u_1} + {u'_1}^2 + f_3\right) u_0}. \tag{13}$$

Now we are able to compute v'_0 using u'_0 and (5):

$$v'_0 = \sqrt{\frac{{u'_0}^2}{\sqrt{u_1}} + f_0}. \tag{14}$$

The last step is to compute v'_1 using (8):

$$v'_1 = \sqrt{v_1 + \sqrt{u_1}\left((\sqrt{u_1} + {u'_1}^2 + f_3)({u'_1}^2 + f_3) + u_1\right) + f_2}. \tag{15}$$

Let us now come back to figuring out which of the two solutions of the quadratic equation $T^2 + T = c$ is the right one. In order to do that, we use the first solution t and continue computing the halved divisor class as explained above. If this choice was correct, then the halved divisor class is a proper one, i. e. it is contained in the subgroup S. So we have to check this. As we have seen above, the trace of $c = u_1^2 \left(\frac{u_0}{u_1} + f_3^2 + \frac{\sqrt{u_1}}{u_1}\right) = u_1 \left(u_0 + u_1 f_3^2 + \sqrt{u_1}\right)$ is zero if and only if the divisor class is contained in S. We now check if the obtained divisor class can be halved, i. e. whether $\mathrm{Tr}\left(u'_1(u'_0 + u'_1 f_3^2 + \sqrt{u'_1})\right) = 0$. If this holds, then the first solution t was correct and we have computed the correct halved divisor class. If the trace is not zero, we use the other solution $t+1$ of the quadratic equation. So the trace serves as a criteria to determine the right solution of (11). Note, that this test involves computing u'_1 and u'_0. So it should be performed as soon as they are computed. If the other solution turns out to be the correct one, we have to redo the computation of u'_1 and u'_0 using $t+1$ instead of t.

After computing u'_1, u'_0, v'_1 and v'_0 we can write the halved divisor class in Mumford representation: $E = [x^2 + u'_1 x + u'_0,\ v'_1 x + v'_0]$.

3.3 The Case $u_1 = 0$

The formulae presented in the previous section hold in the generic case, i. e. if both the input and output have u and v of full degree and no zero coefficients. To complete the above study we now consider how to compute the half of a divisor class with $u_1 = 0$.

The other cases appear with very low probability and do not belong to the main algorithm. Implementers going for an implementation of all possible cases should consult [9] and [8] for a complete case study.

We now consider the divisor class $\overline{D} = [x^2 + u_0,\ v_1 x + v_0]$, where u_1 equals zero. From equation (5) follows directly $u'_0 = 0$. Using this and equation (6) we get $u_0 = u_1'^2$ and hence, $u'_1 = \sqrt{u_0}$. This shrinks (8) to $v_1 = f_2 + v_1'^2$ and we have $v'_1 = \sqrt{v_1 + f_2}$. Having $u'_0 = 0$, one can see that the four equations (5), (6), (7) and (8) do not depend on v'_0 any longer, so this value becomes arbitrary. So, (14) should not be performed.

The complete procedure explained above leads to the actual halving algorithm presented in the next section.

4 The Divisor Class Halving Algorithm

We present an efficient divisor class halving algorithm for genus 2 curves of Type II (cf. Section 2.1) over $\mathbb{F}_{2^d}$, where d is odd, based on the formulae derived in the previous section. We do not follow the steps literally but change them to allow more efficient computations.

We shortly repeat the prerequisites: The curve parameters must be chosen such that the order of $\mathrm{Pic}^0_C(\mathbb{F}_{2^d})$ is equal to $2r$ for an odd number r. The input divisor class must be contained in the order l-subgroup of $\mathrm{Pic}^0_C(\mathbb{F}_{2^d})$, where l is prime.

Algorithm 1. Divisor Class Halving (HLV)

```
INPUT:  Divisor class $\overline{D} = [u, v]$, where $u = x^2 + u_1x + u_0,\ v = v_1x + v_0$ and the
        pre-computed values $f_3^2, \sqrt{f_0}$
OUTPUT: Halved divisor class $\overline{E} = [u', v']$ such that $\overline{D} = 2\overline{E}$
 1: $q_1 \leftarrow \sqrt{u_1},\ q_2 \leftarrow 1/q_1,\ q_3 \leftarrow q_2^2,\ q_4 \leftarrow u_0q_3,\ q_5 \leftarrow \sqrt{q_2}$        $\triangleright$ 1I, 1M, 2SR, 1S
 2: $q_6 \leftarrow \sqrt{q_4},\ c \leftarrow u_1(q_6 + q_5 + f_3)$                                $\triangleright$ 1SR, 1M
 3: $t' \leftarrow \sum_{i=0}^{(d-3)/2} c^{2^{(2i+1)}}$                                            $\triangleright$ 1HT
 4: $u_1' \leftarrow t'q_2,\ t \leftarrow u_1'^2,\ s_1 \leftarrow v_0 + (q_1 + t + f_3)u_0$                $\triangleright$ 2M, 1S
 5: $u_0' \leftarrow \sqrt{s_1},\ b \leftarrow \mathrm{Trace}(u_1'(u_0' + t + f_3))$                       $\triangleright$ 1M, 1SR, 1TR
 6: if $b = 0$ then
 7:     $v_0' \leftarrow q_5u_0' + \sqrt{f_0}$                                                      $\triangleright$ 1M
 8: else
 9:     $t \leftarrow t + q_3,\ u_1' \leftarrow u_1' + q_2$
10:     $u_0' \leftarrow u_0' + q_6,\ v_0' \leftarrow q_5u_0' + \sqrt{f_0}$                                  $\triangleright$ 1M
11: end if
12: $v_1' \leftarrow \sqrt{v_1 + q_1\Big((q_1 + t + f_3)(t + f_3) + u_1\Big) + f_2}$                      $\triangleright$ 2M, 1SR
13: return $[x^2 + u_1'x + u_0',\ v_1'x + v_0']$                 $\triangleright$ Total: 1I, 8M, 5SR, 2S, 1HT, 1TR
```

We now explain the steps of the algorithm according to the formulae derived in the previous section.

Some expressions in the halving formulae do occur more than once. To avoid recomputations, we replace them by $q_1, \ldots, q_6$ in Step 1 and 2. To solve the quadratic equation (11) we have to compute $u_1(u_0 + u_1f_3^2 + \sqrt{u_1})$. What we actually do in the algorithm is computing $u_1(q_6 + f_3 + q_5)$ which is the square root of $u_1(u_0 + u_1f_3^2 + \sqrt{u_1})$ and use that $\mathrm{Tr}(a^2) = \mathrm{Tr}(a)$ for any $a \in \mathbb{F}_{2^d}$. The reason for this is that we can save 1SR since u_1' is root of the (via multiplication by q_2) adjusted solution t'. In Step 3 the solution of the quadratic equation is computed as in (4) and then adjusted in Step 4. In Step 5 the value u_0' is computed according to (13).

According to the explanation at the end of the previous section, we now have to compute the trace of $u_1'(u_0' + \sqrt{u_1'} + u_1' f_3^2)$ in order to perform the check whether we calculated the right solution of the quadratic equation or not. To reduce the number of operations we compute the trace of $u_1'(u_0' + t + f_3)$ instead. Doing this saves 1M, 1SR and 1S. To see that these two traces are equal, we point out that $\mathrm{Tr}(u_1'\sqrt{u_1'}) = \mathrm{Tr}(u_1' t)$, $\mathrm{Tr}({u_1'}^2 f_3^2) = \mathrm{Tr}(u_1' f_3)$ and that the trace is a linear map.

In Steps 6 to 11 we compute v_0' depending on the trace b. If this trace is equal to zero, we continue by computing v_0' using (14). For $b = 1$ we use $t' + 1$ instead of t' in Step 3. Hence, we have to adjust x by adding q_3, u_1' by adding q_2 and u_0' by adding q_6 in Steps 9 and 10. After that we can compute v_0' in the same way as in Step 7.

The last thing to do is computing v_1' using (15) in Step 12. Finally the algorithm returns the desired halved divisor class in Mumford representation. The steps, considered so far, have a total complexity of 1I, 8M, 5SR, 2S, 1HT, 1TR in both cases $b = 1$ and $b = 0$.

5 Conclusion and Outlook

In this paper we presented an efficient halving algorithm for divisor classes of a family of hyperelliptic curves of genus 2 over binary fields. Compared to the previous result by Kitamura, Katagi and Takagi [5,6] we gained a notable speed-up for this family (see Table 2).

In Appendix D of [6] the authors consider curves of form

$$y^2 + xy = x^5 + f_1 x + f_0.$$

The transformation $y \mapsto \tilde{y} + f_1$ maps to the isomorphic curve $\tilde{y}^2 + x\tilde{y} = x^5 + \tilde{f}_0$, which is a Type II curve. This equation shows that their family of curves has only one parameter that can be chosen freely while our family achieves full generality for Type II curves needing far less operations (cf. Table 2).

We would like to point out that we can improve the efficiency of our algorithm as well as that of doubling by leaving out the computation of v_1 in a Montgomery like scalar multiplication, since the new values u_0, u_1 and v_0 do not depend on it. We are investigating the required addition formulae.

Table 2. Complexity of halving algorithms in worst case

	Type II curve (see Section 2.1) $y^2 + xy = x^5 + f_3x^3 + f_2x^2 + f_0$	**Special curve** (see [6], Appendix D) $y^2 + xy = x^5 + f_0$
Kitamura, Katagi, Takagi [5,6]	1I, 15M, 3SR, 3S, 2HT, 2TR	1I, 12M, 5SR, 2S, 1HT, 1TR
This work	1I, 8M, 5SR, 2S, 1HT, 1TR	1I, 8M, 5SR, 2S, 1HT, 1TR

References

1. Avanzi, R., Cohen, H., Doche, C., Frey, G., Lange, T., Nguyen, K., Vercauteren, F.: The Handbook of Elliptic and Hyperelliptic Curve Cryptography. CRC Press, Boca Raton (2005)
2. Byramjee, B., Duqesne, S.: Classification of genus 2 curves over $\mathbb{F}_{2^n}$ and optimization of their arithmetic. Cryptology ePrint Archive of IACR, Report 2004/107 (2004)
3. Choie, Y., Yun, D.K.: Isomorphism Classes of Hyperelliptic Curves of Genus 2 over $\mathbb{F}_q$. In: Information Security and Privacy – ACISP 2002. LNCS, vol. 2384, pp. 190–202. Springer, Heidelberg (2002)
4. Gaudry, P., Hess, F., Smart, N.P.: Constructive and destructive facets of Weil descent on elliptic curves. Journal of Cryptology 15(1), 19–46 (2002)
5. Kitamura, I., Katagi, M., Takagi, T.: A Complete Divisor Class Halving Algorithm for Hyperelliptic Curve Cryptosystems of Genus Two (for a full version see [6]). In: Boyd, C., González Nieto, J.M. (eds.) ACISP 2005. LNCS, vol. 3574, pp. 146–157. Springer, Heidelberg (2005)
6. Kitamura, I., Katagi, M., Takagi, T.: A Complete Divisor Class Halving Algorithm for Hyperelliptic Curve Cryptosystems of Genus Two. Cryptology ePrint Archive of IACR, Report 2004/255 (2005)
7. Knudsen, E.W.: Elliptic Scalar Multiplication Using Point Halving. In: Lam, K.-Y., Okamoto, E., Xing, C. (eds.) ASIACRYPT 1999. LNCS, vol. 1716, pp. 135–149. Springer, Heidelberg (1999)
8. Lange, T.: Formulae for Arithmetic on Genus 2 Hyperelliptic Curves. Applicable Algebra in Engineering, Communication and Computing 15(5), 295–328 (2005)
9. Lange, T., Stevens, M.: Efficient Doubling for Genus Two Curves over Binary Fields. In: Handschuh, H., Hasan, M.A. (eds.) SAC 2004. LNCS, vol. 3357, pp. 170–181. Springer, Heidelberg (2004)
10. Lidl, R., Niederreiter, H.: Finite Fields. Encyclopedia of Mathematics and its Applications, vol. 20. Addison-Wesley, Reading (1983)
11. Schroeppel, R.: Elliptic curve point halving wins big. In: 2nd Midwest Arithmetical Geometry in Cryptography Workshop, Urbana, November 2000, Illinois (2000)

Message Authentication on 64-Bit Architectures

Ted Krovetz

Department of Computer Science
California State University, Sacramento CA 95819 USA

Abstract. This paper introduces VMAC, a message authentication algorithm (MAC) optimized for high performance in software on 64-bit architectures. On the Athlon 64 processor, VMAC authenticates 2KB cache-resident messages at a cost of about 0.5 CPU cycles per message byte (cpb) — significantly faster than other recent MAC schemes such as UMAC (1.0 cpb) and Poly1305 (3.1 cpb). VMAC is a MAC in the Wegman-Carter style, employing a "universal" hash function VHASH, which is fully developed in this paper. VHASH employs a three-stage hashing strategy, and each stage is developed with the goal of optimal performance in 64-bit environments.

Keywords: Message authentication, universal hashing, architectural optimization.

I personally believe there are two main architectures out there: Power and x86-64 [both of which are 64-bit architectures].
— Linus Torvalds, 2005.

1 Introduction

Over the years, as design and manufacturing techniques have improved, and demand for memory addressability has increased, register lengths have become longer. The recent adoption of 64-bit register architectures for mainstream processors from IBM, Intel and Advanced Micro Devices is a natural evolution in this process. It is reasonable to believe that, just as 32-bit processors did before them, 64-bit processors will become dominant not only in servers but also in desktops and laptops.

Many algorithms, especially in domains where high performance is desirable, are designed with optimizations tailored for particular architectures. Changing architectures while keeping the same designs can easily lead to suboptimal performance. This is the case with high-speed message authentication and the move from 32-bit to 64-bit architectures. The fastest reported software-optimized message authentication algorithms (or MACs) are all designed to run well on 32-bit architectures [5,6,9]. While these MACs generally work equally well on both 32- and 64-bit architectures — because the newer architectures support older instructions at full speed — they are not designed to take advantage of new

© Springer-Verlag Berlin Heidelberg 2007

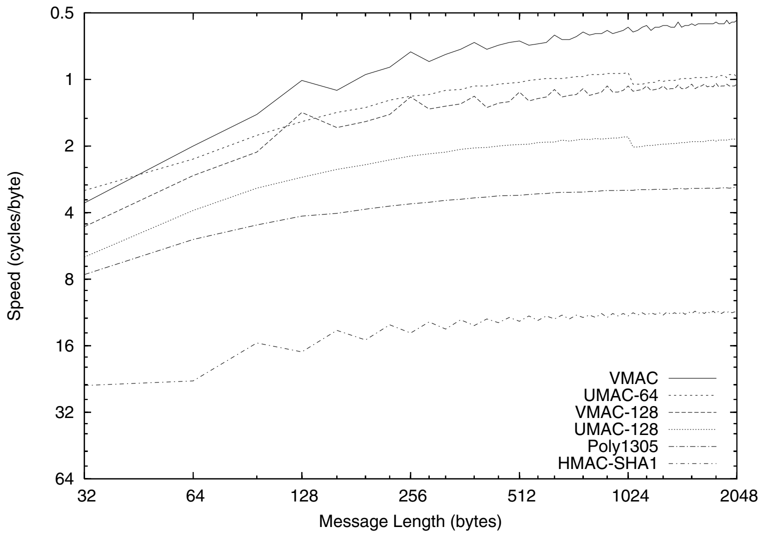


Fig. 1. Efficiency on the Athlon 64 processor of the hash functions underlying recent MACs, measured in CPU cycles per byte of message hashed. VMAC and UMAC-64 produce 64-bits while the others produce 128 (or 160 for SHA-1).

capabilities found in 64-bit processors. As an example consider UMAC, which was designed specifically for optimal performance on 32-bit architectures [5]. On an Athlon 64 processor, UMAC achieves peak speeds of about 1.1 CPU cycles per byte of message authenticated (cpb) when it is restricted to using 32-bit operands, but improves only slightly to 1.0 cpb when allowed full use of 64-bit operands.

This paper presents VMAC, the fastest reported MAC on contemporary 64-bit processors, achieving peak speeds of 0.5 cpb on the Athlon 64 (and as low as 0.3 cpb if one allows larger internal keys, see Figure 4). This compares favorably with other MACs. Figure 1 shows the performance of the internal hash functions of recent high-speed MAC's.[1] (Hash speeds determine a MAC's relative speed because all modern MACs hash their messages first.) Although the main goal of the VMAC design effort is high speed on 64-bit processors, VMAC is careful to avoid some of the perceived deficiencies of recent high-speed MACs, particularly by limiting use of internal hash key (VMAC uses 160 bytes) and avoidance of data-dependant side-channel attacks. VMAC has the desirable properties of being provably secure, parallelizable, and patent-free.

[1] Timings in this paper are generated using `gcc` 4.0 with optimization level `-O4` (or `-fast` if available) and appropriate `-march` or `-mcpu` settings. The bulk of performance claims are based on an AMD Athlon 64 "Manchester" (family 15, model 43, stepping 1, L2 cache 512K). Other architectures are reported in Section 3.6. All data 16-byte aligned and in cache. UMAC and Poly1305 timings are made from code obtained at their author's websites. SHA timings are as reported by OpenSSL.

Message Authentication. Message authentication is used when parties wish to communicate with assurance that received messages come from the claimed sender without alteration. All of the fastest MACs follow principles developed by Wegman and Carter [3,5,6,7,9,14]. The basic Wegman-Carter message authentication paradigm is for the sender first to hash the message with a hash function known only to himself and the receiver. The sender then applies some cryptographic function (usually encryption) to the resulting hash value, which produces a message tag that is sent along with the message to the receiver. The receiver can then repeat the process, verifying that the received tag is valid for the received message. In a correctly designed MAC, only those knowing the secret hash function and cryptographic keys have a reasonable chance of creating a valid tag for any new message. If, however, an adversary is able to produce a valid tag for a new message without knowing the hash function and cryptographic keys, then a forgery has occurred. Due to their ephemeral nature, communication sessions usually need only be secure against forgery during the lifetime of the session. If an adversary cannot forge with a probability of more than about $1/2^{60}$ per attempt, then the MAC is likely suitable for most communications where attackers are not allowed an excessive number of forgery attempts. VMAC and UMAC are flexible in their security levels. They are able to produce 64-bit tags with forgery probabilities close to $1/2^{60}$ twice as fast as versions of VMAC and UMAC which produce 128-bit tags (which have forgery probabilities closer to $1/2^{100}$). Most other MAC schemes, including the ones used for comparison in this paper, are not designed for such flexibility. As a result, when making speed comparisons with VMAC's 64-bit tags, other schemes, producing longer tags, are unavoidable disadvantaged.

The key to speed in a Wegman-Carter MAC is the hash function used. Authentication speeds are determined by the sum of the (length-dependent) time it takes to hash the message being authenticated plus the (constant) time it takes to cryptographically produce the tag, so this paper focuses on the hash function used in VMAC, known as VHASH. This is reasonable because any speed improvements in the cryptographic part of a Wegman-Carter MAC could be applied equally to all such schemes, so improvements relative to other Wegman-Carter MACs will come almost entirely from improvements in hashing.

Notable recent examples of fast hash functions suitable for Wegman-Carter message authentication (and peak speeds reported for Intel Pentium 4 processors by their authors) are hash127 (around 4.4 cpb), hash1305 (3.4 cpb), Badger (2.2 cpb) and UMAC (1.0 cpb). The speeds of all of these favorably compare with popular non-Wegman-Carter MACs such as HMAC-SHA1 and CBC-AES-MAC, both of which require more than 10 cpb.

Universal hashing. The hash function used in a Wegman-Carter MAC must be chosen from a *universal* hash function family. A hash-function family H is a collection of hash functions, each $h \in H$ having domain A and finite codomain B. A hash-function family H is ε-*almost universal* (ε-AU) if the probability is no more than ε that any two distinct inputs m, m' hash to the same output when hashed by a randomly selected member of H. A small value for ε indicates that an adversary is unlikely to be able to choose a pair of inputs that hash to the same output, as long as the hash function is chosen randomly. A stronger notion bounds an adversary's ability to guess differences between hash outputs. A hash-function family H is ε-*almost delta universal* (ε-AΔU) if the probability is no more than ε that $h(m) - h(m') = d$ for any two distinct inputs m, m' and any chosen constant d when hashed by a randomly selected member h of H. There are stronger notions of universal hashing defined by Wegman, Carter and Stinson [7,13,14], but ε-AΔU is adequate for message authentication, and is achieved by VHASH.

2 Three-Stage Hashing

VHASH uses a three-stage hashing strategy where each hash stage is made of a discrete hash function with a particular purpose. The first stage rapidly compresses by a fixed ratio the message to be hashed, thus reducing the data to be processed by later (slower) stages. The second stage hashes the newly compressed message to a fixed length, and the third stage distills the security of the second-stage output into a smaller number of bits. In this section, we investigate appropriate primitive hash functions for each stage and develop them for 64-bit architectures. In the next section, VHASH is assembled from the functions described here and analyzed for the purposes of message authentication. The first MAC to employ a three-stage strategy was UMAC [5,11].

2.1 Stage 1 – Acceleration

The goal of the first stage is to act as an accelerant for later hash stages by compressing, at a constant ratio, long inputs into shorter ones at very high speed. VHASH uses the NH hash family for this purpose, breaking the VHASH input into b-bit blocks (the final block may be shorter) and using NH to hash each into 128 bits. The hashed blocks are then concatenated into a string shorter than the original VHASH input. When b is set at 1,024 bits (as we later recommend), the compression is 8:1 for messages whose length is a multiple of 1,024.

NH was originally designed as a parameterized hash function [5]. Given positive integer parameters n and w and a key K of length nw bits, then NH can hash any string M that is a multiple of $2w$ bits in length but not longer than nw bits. First M and K are broken into w-bit blocks $M_1, M_2, \ldots, M_\ell$ and $K_1, K_2, \ldots K_n$ where $\ell = |M|/w$. Then, each block is interpreted as a w-bit unsigned binary integer $m_1, m_2, \ldots, m_\ell$ and $k_1, k_2, \ldots k_n$. Finally, the hash result is computed as

$$\mathrm{NH}[n,w](K,M) = \sum_{i=1}^{\ell/2} ((m_{2i-1} + k_{2i-1} \ \mathbf{mod}\ 2^w) \times (m_{2i} + k_{2i} \ \mathbf{mod}\ 2^w)) \ \mathbf{mod}\ 2^{2w} .$$

NH is a hash family, and choosing a random function from the hash family is done by choosing a random nw-bit key K. NH is known to be (2^{-w})-AΔU over messages of the same length (ie, M and M' are distinct, but $|M| = |M'|$), and small modifications to the original proof show that NH is (2^{-w})-AΔU over messages that are any multiple of $2w$ bits in length (but still no longer than nw bits). In the context of VHASH, $w = 64$ and $nw = b$ is suggested 1,024.

Characteristics. The chief advantage of NH is extreme speed. Every operation is done naturally and efficiently on contemporary processors if w is chosen appropriately. On 64-bit processors with good support for multiplying 64-bit quantities into a 128-bit result, defining $w = 64$ results in very high speeds.

On a 64-bit architecture, NH performance when $w = 64$ is about four times better than when $w = 32$. If one's goal is a (2^{-64})-AU guarantee over messages of length $128j$ bits, then NH$[n, w]$ achieves this goal using j multiplications when $w = 64$, but requires $4j$ multiplications when $w = 32$. To see this, consider how one would achieve a (2^{-64})-AU guarantee when $w = 32$. Each NH hashing of the message would require $2j$ multiplications and produce a hash value with a (2^{-32})-AU guarantee. This would have to be done twice, under separate keys, to achieve the (2^{-64})-AU goal, whereas only j 64-bit multiplications are needed to achieve the same guarantee on a 64-bit architecture. This is borne out experimentally. Two passes with $w = 32$ takes about 2 cpb while a single pass with $w = 64$ requires only around 0.5 cpb on the Athlon 64.

One of the design goals for VHASH is achieving a balance between performance and internal key requirement. As can be seen in Figure 4, increasing the NH key length in VHASH increases VHASH performance for long messages greatly at first, but performance increases drop-off at around 128–256 bytes. We recommend 128 bytes for the NH hash key for applications which are not extremely memory constrained. This choice harnesses most of the potential speed gains of NH with fairly low key requirement.

2.2 Stage 2 – Fix Length

The first stage produces an output proportional in length to the original input, which means that to achieve a fixed length, further hashing is necessary. Recent research into various polynomial-based hash functions have yielded hash functions appropriate for the task with good speed and universality guarantees [1,3,11,12]. Section 3 will address domain reconciliation necessary between stage-one outputs and stage-two inputs.

A simple and efficient method to hash a string M is to fix prime number p and break M into fixed-length blocks $M_1, M_2, M_3, \ldots, M_\ell$ in such a way that when the blocks are interpreted as unsigned integers $m_1, m_2, m_3, \ldots, m_\ell$, each is less than p (for example, by making each block $\lfloor \log_2 p \rfloor$ bits). Then, choosing an integer key $0 \le k < p$ defines the hash output as

$$h_k(M) = m_1 k^\ell + m_2 k^{\ell-1} + \cdots + m_\ell k^1 \bmod p\,.$$

Two different messages M, M' of the same block length ℓ differ by constant d when hashed by this function if

$$h_k(M) - h_k(M') = (m_1 - m_1')k^\ell + (m_2 - m_2')k^{\ell-1} + \cdots + (m_\ell - m_\ell')k^1 \bmod p = d\,.$$

Because $M \neq M'$, at least one of the coefficients in this polynomial is non-zero. This being a polynomial of degree at most ℓ, there are at most ℓ values for k which cause $h_k(M) - h_k(M') - d \pmod p$ to evaluate to zero. If we define a hash family $H = \{h_k \mid 0 \leq k < p\}$, then H is an ε-AΔU hash family for $\varepsilon = \ell/p$.

Characteristics. With care, polynomial hashing can be made to perform well. Horner's Rule suggests rephrasing $h_k(M)$ as $((\cdots((m_1k + m_2)k + m_3)k\cdots)k + m_\ell)k \bmod p$, which allows $h_k(M)$ to be computed as a sequence of ℓ multiplications and additions modulo p [10]. Those multiplications and additions modulo p can be made efficient by choosing a convenient p and restricting the choice of k to a convenient set.

By choosing p to be of the form $p = 2^a - b$ for some small b, reductions modulo p can be done efficiently in a lazy manner. Each time a value c becomes at least 2^a, it can be rewritten as the (modulo p) equivalent $c - 2^a + b$. For example $p = 2^{61} - 1$ is prime. This means that, in a 64-bit register, a value c greater than p but less than 2^{64} can be reduced by computing $c = (c \textbf{ div } 2^{61}) + (c \textbf{ mod } 2^{61})$. This equality simply recognizes that $c = x2^{61} + y$ for some x and $0 \leq y < 2^{61}$, and replaces 2^{61} with the equivalent (modulo p) value 1. The **div** and **mod** operations extract x and y, and can be computed efficiently using bitwise operations. This process is "lazy" for two reasons. First, numbers can be allowed to get as large as desired before performing a reduction as long as values do not exceed the register's capacity. Second, a reduction to the range $0, \ldots, p-1$ is not necessary until a final result is needed. So, when this method is followed to perform an intermediate reduction, the result need not be in the range $0, \ldots, p-1$. This puts off expensive range checks until the very end of the polynomial hash. Particularly useful primes on a 64-bit architecture are $2^{127} - 1$ and $2^{61} - 1$.

Another source of inefficiencies is register carries during addition. Whenever a number is too large to be represented in a single CPU register, the number is generally split into multiple registers, and arithmetic on the larger number is accomplished by some sequence of smaller operations. For example, if we rewrite 128-bit values j and k as $j = w2^{64} + x$ and $k = y2^{64} + z$ where $0 \leq x, z < 2^{64}$, then $jk = wy2^{128} + (wz + xy)2^{64} + xz$. This means that to compute jk, we can put the top 64-bits of j and k into 64-bit registers w and y, and their low 64-bits into x and z. The result jk is then assembled by appropriately multiplying, shifting and adding wy, wz, xy and xz.

Consider the case where a polynomial is being evaluated modulo prime $p = 2^{127} - 1$ using Horner's Rule with lazy modulo reduction whenever an intermediate value exceeds 128-bits. Each step in the Horner's Rule evaluation is a multiplication and addition of the form $jk + m \bmod p$, with $k, m < p$ and $j < 2^{128}$. As just seen, say that j and k are 128-values spread into registers w, x, y and z so that $jk = wy2^{128} + (wz + xy)2^{64} + xz \pmod p$. Because $2^{128} = 2 \pmod p$, this can be rewritten $jk = ((wz + xy) \textbf{ mod } 2^{64})2^{64} + (2(((wz + xy) \textbf{ div } 2^{64}) + wy) + xz)$ $\pmod p$. If j and k are unrestricted, then every addition could result in a carry beyond 128-bits. These carries must be accumulated and dealt with, which could be inefficient. Ideally this computation of jk would involve no carries beyond 128-bits, allowing a more efficient computation.

Eliminating carries can be done by restricting k. The polynomial hash described in this section is (ℓ/p)-A$\varDelta$U when hashing ℓ-block messages and choosing k from $0, \dots, p-1$. This is due to the fact that there are at most ℓ values in the range $0, \dots, p-1$ that cause $h_k(M) - h_k(M') - d \pmod p$ to evaluate to zero. If k is chosen from some subset $A \subseteq \{0, \dots, p-1\}$ instead, there would still be at most ℓ values that cause $h_k(M) - h_k(M') - d \pmod p$ to evaluate to zero, but because $|A| \leq p$, the probability of randomly choosing one of them increases to at most $\ell/|A|$. This means A can be chosen judiciously to exclude keys which cause excessive carries. In the case of evaluating polynomials modulo $p = 2^{127}-1$, restricting k to elements of $A = \{y2^{64}+z \mid 0 \leq y < 2^{62}, 0 \leq z < 2^{63}\}$ eliminates all but one possible carry beyond 128-bits when computing jk on a 64-bit architecture for any $0 \leq j < 2^{128}$.

Experimentally, we have found that long sequences of cache-resident message blocks, each already less than $2^{127} - 1$, can be hashed at a rate of 1.7 cpb on the Athlon 64 when k is chosen as described to avoid excessive carries. When hashing sequences of values less than $2^{61} - 1$ over modulus $2^{61} - 1$, allowing k to be any value less than $2^{61} - 1$, messages can be hashed at 1.3 cpb on the Athlon 64. It should be noted that hashing an arbitrary string would not be nearly as fast due to the need of breaking the string into appropriate blocks (within the modulus).

2.3 Stage 3 – Distillation

When NH is defined for $w = 64$ and the Polynomial hash is defined over prime modulus $p = 2^{127} - 1$, as is the case in VHASH, the resulting universality guarantee of the first two stages composed can be no better than (2^{-64})-A$\varDelta$U (more on this in Section 3) and yet the output requires 127 bits. To reduce the disparity between the number of bits needed for the hash output and the universality guarantee, one final hash is used to hash the fixed length stage-two output into fewer bits.

Another well known provably universal hashing function is the inner product over a prime modulus [8]. Again, let p be a prime and let M be broken into fixed-length blocks $M_1, M_2, M_3, \dots, M_\ell$ in such a way that when the blocks are interpreted as unsigned integers $m_1, m_2, m_3, \dots, m_\ell$, each is less than p. Then, choosing a vector $\mathbf{k} = (k_1, k_2, \dots, k_\ell)$ with $0 \leq k_i < p$ for all $1 \leq i \leq \ell$ defines the hash output as

$$h_{\mathbf{k}}(M) = m_1 k_1 + m_2 k_3 + \cdots m_\ell k_\ell \bmod p\,.$$

For any two different messages M, M' of the same block length ℓ and integer $0 \leq d < p$, when $\mathbf{k}$ is chosen at random, the probability that

$$h_{\mathbf{k}}(M) - h_{\mathbf{k}}(M') = (m_1 - m_1')k_1 + (m_2 - m_2')k_2 + \cdots + (m_\ell - m_\ell')k_\ell \bmod p = d$$

is exactly $1/p$. It follows that inner product hashing over a prime modulus forms an ε-A$\varDelta$U hash family for $\varepsilon = 1/p$.

```
$\mathsf{VHASH}[b](M, K, k, k_1, k_2)$
Inputs:
    $M$, a string of any length
    $K$, a string of length $b$ bits, where $b = 128i$ for some integer $i > 1$
    $k$, an element of $\{w2^{96} + x2^{64} + y2^{32} + z \mid w, x, y, z \in \mathbb{Z}_{2^{30}}\}$
    $k_1, k_2$, integers in the range $0 \ldots 2^{61} - 2$, inclusive
Output:
    $h$, an integer in the range $0 \ldots 2^{61} - 2$, inclusive
Algorithm:
1.  $n = \max(\lceil |M|/b \rceil, 1)$
2.  Let $M_1, M_2, \ldots, M_n$ be strings so that $M_1||M_2||\cdots||M_n = M$ and
        $|M_i| = b$ for $1 \leq i < n$.
3.  $\ell_i = |M_i|$ for each $1 \leq i \leq n$
4.  Let $M_n = M_n||0^j$ where $j \geq 0$ is the smallest integer so
        that $|M_n| + j$ mod $128 = 0$
5.  Byte-reverse each 64-bit word in $M_i$ for each $1 \leq i \leq n$
6.  $a_i = (\mathsf{NH}[b/64, 64](K, M_i) \bmod 2^{126}) + (\ell_i \bmod b)2^{64}$ for each $1 \leq i \leq n$
7.  $p = k^{n+1} + a_1 k^n + a_2 k^{n-1} + \cdots + a_n k^1 \bmod (2^{127} - 1)$
8.  $p_1 = (p \mathbf{\ div\ } 2^{64}) \bmod 2^{60}$
9.  $p_2 = p \bmod 2^{60}$
10. $h = p_1 k_1 + p_2 k_2 \bmod (2^{61} - 1)$
```

Fig. 2. The hash family VHASH is ε-$A\Delta U$, when K, k, k_1, k_2 are chosen randomly from their domains, where $\varepsilon = 2^{-59.9} + (\ell/b)2^{-107}$

Characteristics. Inner-product hashing requires at least as much key as message being hashed. This makes it unsuitable for long messages. But, for short messages, implementations can be efficient using strategies already discussed for polynomial hashing. In particular, lazy modular reduction and choosing a prime modulus of the form $p = 2^a - b$ where b is small, results in good performance. For example, when $p = 2^{61} - 1$, $j < 2^{64}$ and $k < p$, the product $jk \pmod p$ can be efficiently computed as $(jk \mathbf{\ div\ } 2^{64})2^3 + (jk \bmod 2^{64})$ because $2^{64} = 2^3 \pmod p$. This is exactly the computation done by VHASH in its third stage.

3 VHASH Definition

With these component hash functions as building blocks and the three-stage hash function as a model, a hash function suitable for authenticating arbitrary messages and optimized for 64-bit architectures can be presented. For any b which is a positive multiple of 128, Figure 2 specifies the hash family $\mathsf{VHASH}[b]$ where choosing a random function h from the family is achieved by choosing K, k, k_1 and k_2 uniformly at random from their domains and letting $h(\cdot) = \mathsf{VHASH}(\cdot, K, k, k_1, k_2)$.

Theorem 1. *Let b be any positive multiple of 128 and let ℓ be any positive integer, then* $\mathsf{VHASH}[b]$ *is ε-AΔU over all binary strings up to length ℓ bits where $\varepsilon = 2^{-59.9} + (\ell/b)2^{-107}$.*

The theorem will be proven over a sequence of lemmas later in this section. With this result, VHASH can easily be embedded in a Wegman-Carter MAC which we call VMAC. Here we summarize its construction. (It will be fully specified in a separate document.) Let $p = 2^{61} - 1$ and N be some nonce space. Then for functions $f : N \to \mathbb{Z}_p$ and $h \in \mathsf{VHASH}[b]$, tag generation under VMAC is defined as $\mathsf{VMACTagGen}_{f,h}(m, n) = h(m) + f(n) \bmod p$ for message m and nonce n. We define the security of a nonce-based MAC scheme, such as VMAC, that uses tag-generation function $\mathsf{TagGen}(m, n)$ as follows. Assume an adversary knows any sequence of triples $(m_1, n_1, t_1) \dots (m_q, n_q, t_q)$ where each n_i is unique and $t_i = \mathsf{TagGen}(m_i, n_i)$ for each $1 \leq i \leq q$. The MAC scheme is α-secure if the adversary cannot produce (m, n, t) with probability exceeding α where $(m, n) \neq (m_i, n_i)$ for any i and $t = \mathsf{TagGen}(m, n)$. The following theorem follows from the theory of Wegman-Carter MACs.

Proposition 2. *Let ℓ be a positive integer, $p = 2^{61} - 1$ and N be some non-empty set. Let b be a positive multiple of 128. Let* $\mathsf{VMACTagGen}_{f,h}(m, n) = h(m) + f(n)$ *mod p for randomly chosen functions $f : N \to \mathbb{Z}_p$ and $h \in$* $\mathsf{VHASH}[b]$*. Then,* $\mathsf{VMACTagGen}_{f,h}$ *is a $(2^{-59.9} + (\ell/b)2^{-107})$-secure over messages upto ℓ bits in length.*

3.1 VHASH Analysis

The hash functions seen so far have interfaces that are incompatible with one another without some adaptation. For example, NH produces outputs with values up to $2^{128} - 1$, whereas the polynomial hash only accepts sequences of values less than $2^{127} - 1$. Similarly, the polynomial hash produces a value less than $2^{127} - 1$, but the inner-product expects a sequence of values less than $2^{61} - 1$. To address these problems, a lemma is introduced which allows out-of-range values to be brought into range with a manageable increase to the probabilities involved. Length issues must also be resolved. As presented, each hash function seen so far has a universality guarantee when hashing messages of equal length. These must be extended to provide universality guarantees over all lengths. Each stage of VHASH will now be analyzed for universality guarantee and interface.

3.2 First: A Lemma

The primary tool used to fix the problem that one hash function produces values that are outside of the domain of a second hash function is the following lemma which says that if we routinely zero any fixed bit-position of the outputs of an ε-AΔU hash function, the resulting hash function is still AΔU but with a reduced universality guarantee.

```
VHASH-128[b](M, K_1, K_2, k)
Inputs:
  $M$, a string of any length
  $K_1, K_2$, strings of length $b$ bits, where $b = 128i$ for some integer $i > 1$
  $k$, an element of $\{w2^{96} + x2^{64} + y2^{32} + z \mid w, x, y, z \in \mathbb{Z}_{2^{30}}\}$
Output:
  $h$, an integer in the range $0 \dots 2^{127} - 2$, inclusive
Algorithm:
1. $n = \max(\lceil |M|/b \rceil, 1)$
2. Let $M_1, M_2, \dots, M_n$ be strings so that $M_1 || M_2 || \cdots || M_n = M$ and
     $|M_i| = b$ for $1 \le i < n$.
3. $\ell_i = |M_i|$ for each $1 \le i \le n$
4. Let $M_n = M_n || 0^j$ where $j \ge 0$ is the smallest integer so
     that $|M_n| + j \bmod 128 = 0$
5. Byte-reverse each 64-bit word in $M_i$ for each $1 \le i \le n$
6. $a_i = (\mathsf{NH}[b/64, 64](K_1, M_i) \bmod 2^{126}) + (\ell_i \bmod b)2^{64}$ for each $1 \le i \le n$
7. $b_i = \mathsf{NH}[b/64, 64](K_2, M_i) \bmod 2^{126}$ for each $1 \le i \le n$
8. $h = k^{2n+1} + a_1 k^{2n} + b_1 k^{2n-1} + a_2 k^{2n-2} + b_2 k^{2n-3} + \cdots$
     $+ a_n k^2 + b_n k^1 \bmod (2^{127} - 1)$
```

Fig. 3. The hash family VHASH-128 is ε-$A\Delta U$, when K_1, K_2, k_1, k_2 are chosen randomly from their domains, where $\varepsilon = (\ell/b)2^{-118}$

Definitions. When x is a non-negative integer, let x_i be 1 if the binary representation of x has a 1 in the position of weight 2^i and 0 otherwise. Let $\text{Zero}_i(x)$ be the function that returns x if $x_i = 0$ and returns $x - 2^i$ if $x_i = 1$ (ie, it returns x with the 2^i position zeroed). $\mathbb{Z}_n$ is the set $\{0, 1, 2, \dots, n-1\}$. When s is a string, $|s|$ is its bitlength.

Lemma 3. *Let $H = \{h : A \to \mathbb{Z}_n\}$ be an ε-$A\Delta U$ hash family (where the operation is addition modulo n) and $H_i = \{\text{Zero}_i \circ h \mid h \in H\}$, then H_i is (3ε)-$A\Delta U$ for every i.*

Proof. Let $a \ne b$ be elements of A, and d and d' be elements of $\mathbb{Z}_n$. Because H is ε-$A\Delta U$, we know that $\Pr[h(a) - h(b) = d] \le \varepsilon$ when h is chosen randomly from H, but what is the probability $\Pr[h'(a) - h'(b) = d']$ when h' is chosen randomly from H_i? Let h be chosen randomly, and let $h' = \text{Zero}_i \circ h$ for some $0 \le i < \lg n$. Define $x = h(a)$ and $y = h(b)$. There are four possible combinations for the values of x_i and y_i: (x_i, y_i) could equal $(0,0)$, $(0,1)$, $(1,0)$ or $(1,1)$. We look at each case.

When $x_i = y_i$, then $h'(a) - h'(b) = d'$ if and only if $h(a) - h(b) = d'$. Using conditional probability we can bound the likelihood of this scenario as $\Pr[h(a) - h(b) = d' \text{ and } x_i = y_i] = \Pr[h(a) - h(b) = d'] \cdot \Pr[x_i = y_i \mid h(a) - h(b) = d'] \le \varepsilon \cdot 1$. Similarly, if (x_i, y_i) is $(0,1)$ or $(1,0)$ then $h'(a) - h'(b) = d'$ if and only if $h(a) - h(b)$

is $d' + 2^i$ or $d' - 2^i$, respectively, each of which is similarly bounded by $\varepsilon \cdot 1$. These three cases being the only ones in which $h'(a) - h'(b) = d'$, H' must be 3ε-$A\Delta U$. □

Note that a similar result is not possible for ε-AU hash families. Zeroing a bit of an ε-AU hash family can eliminate all guarantees. The identity function $f_I(x) = x$ is 0-AU, but if you zero the last bit of the output (ie, define $h = \text{Zero}_0 \circ f_I$), then $h(s||0)$ and $h(s||1)$ always collide for every s.

3.3 Stage 1 – NH

The goal of the first hashing phase (Lines 1–6 of Figure 2) is to hash arbitrary messages into much shorter representations (albeit proportional in length to their originals) in such a way that two distinct arbitrary-length messages have a low probability of hashing to the same result (so that inputs to the next hash phase are unlikely to be the same). Letting b be any positive multiple of 128, Lines 1–6 of Figure 2 defines a hash family utilizing NH. The domain of the hash family is binary strings of any length. The codomain is vectors of integers from $\mathbb{Z}_{2^{126}}$. Randomly choosing a function from the hash family is achieved by choosing a random b-bit string K. Lines 1–6 work as follows. Given string M, break M into $n = \lceil |M|/b \rceil$ blocks $M_1, M_2, \ldots, M_n$ so that each of the first $n-1$ blocks is length b and M_n is whatever is left over (Lines 1–2). If M was the empty string, then n is set to 1. Each of the blocks $M_1, \ldots, M_{n-1}$ is guaranteed to be in the domain of NH. Block M_n may not be a multiple of 128, and so not in the domain of NH which is only defined for inputs with length divisible by $2w$. Appending the fewest number of zero bits needed to make it so will bring M_n into the domain of NH (Line 4). The blocks are then each hashed independently by NH, the two most significant bits of the results are zeroed (Line 6), and the result has the modulo-b pre-zero-padding length of its corresponding block added. Finally,. The n resulting values form a vector which is the hash function's output.

Lemma 4. *Let b be any positive multiple of 128. Lines 1–6 of Figure 2 define a $(9/2^{64})$-AU hash family over binary strings of arbitrary length.*

Proof. Let b be a positive multiple of 128, K be a uniformly distributed b-bit string, and $M \neq M'$ arbitrary binary strings. Let $M = M_1, \ldots, M_m$ and $M' = M'_1, \ldots, M'_n$ be broken into blocks and let ℓ_i and ℓ'_i represent the length of M_i and M'_i as described in Lines 1–3 of Figure 2. Let M_m and M'_n be zero extended to the nearest multiple of 128 bits, if needed, as described in Line 4. The byte-reversal of Line 5 has no effect on whether $M_i = M'_i$ for any i. What is the probability that identical vectors are produced by evaluating Line 6 on $M_1, \ldots, M_m$ and $M'_1, \ldots, M'_n$? If $n \neq m$, the probability of collision is zero because the vectors produced will be different lengths. There are two other cases to examine.

If $n = m$ and $M_i \neq M'_i$ for some $1 \leq i \leq n$, then, because NH is 2^{-64}-AΔU over strings that are a multiple of $2w = 128$ bits in length (which both M_i and M'_i are guaranteed to be), the probability that $(\text{NH}(K, M_m) \textbf{ mod } 2^{126}) -$

$(\mathrm{NH}(K, M'_n) \mathbf{\ mod\ } 2^{126}) = 0$ is no more than $9/2^{64}$. The factor of nine comes from the **mod** 2^{126}, which has the affect of zeroing the top two bits of the NH output. Lemma 3 says that this causes up to a factor of nine degradation.

There is one more situation to consider: when one string is a proper prefix of the other before zero-padding, but the two strings are identical afterward. In this case, $M_m = M'_n$ because the strings are the same after padding but $\ell_m \neq \ell'_n$ because one string was a proper prefix of the other before padding. There is thus zero probability that $(\mathrm{NH}(K, M_m) \mathbf{\ mod\ } 2^{126}) + (\ell_m \mathbf{\ mod\ } b)2^{64} = (\mathrm{NH}(K, M'_n) \mathbf{\ mod\ } 2^{126}) + (\ell'_n \mathbf{\ mod\ } b)2^{64}$ because the NH hashes are guaranteed to give the same result, but two different lengths are added.

In every case, the probability that the vectors output are identical when hashing M and M' under key K and parameter b is no more than $9/2^{64}$. □

3.4 Stage 2 – Polynomial

The goal of the second hashing phase (Lines 7–9 in Figure 2) is to take the unbounded-length output of the first NH hash phase and hash it to a short fixed-length string in such a way that if two inputs to this stage differ then the probability that the outputs collide is low. Lines 7–9 define a universal hash family. The domain of the hash family is vectors of integers from $\mathbb{Z}_{2^{127}-1}$. The codomain is ordered pairs from $\mathbb{Z}_{2^{60}} \times \mathbb{Z}_{2^{60}}$. Choosing a random function from the hash family is done by choosing a random element $k \in \{w2^{96} + x2^{64} + y2^{32} + z \mid w, x, y, z \in \mathbb{Z}_{2^{30}}\}$. Line 7 is a simple polynomial evaluation hash modulo $2^{127} - 1$. Lines 8–9 utilize Lemma 3 by zeroing seven bits and then breaking in two the result to produce an output in the domain of the third hash phase. Since the first NH phase outputs sequences of values less than 2^{126}, those outputs are suitable without modification for hashing by the polynomial hash.

Lemma 5. *Let $n \geq 0$ be an integer. Lines 7–9 of Figure 2 define a $(n/2^{107})$-AU hash family over vectors of length up to n of values less than $2^{127} - 1$.*

Proof. It is known that the polynomial hash of Section 2 is universal over vectors of the same length. So, to allow vectors of varying length, let n be an integer no less than the length of the longest vector to be hashed. Then, to hash vector $m_1, m_2, \ldots, m_j$ with the polynomial hash of Section 2, first prepend $n - j$ zeros and a one to the vector, resulting in a vector $0, 0, \ldots, 0, 1, m_1, \ldots, m_j$ of length $n + 1$ elements. This preprocessing assures that all vectors hashed by the polynomial are the same length, and it assures that any pair of vectors that are different before preprocessing are also different after preprocessing. This preprocessing step extends the basic polynomial hash of Section 2 to vectors up to length n, but maintains a $((n + 1)/2^{120})$-AΔU guarantee when key k is chosen from $\{w2^{96} + x2^{64} + y2^{32} + z \mid w, x, y, z \in \mathbb{Z}_{2^{30}}\}$. Notice that Line 7 of Figure 2 produces the same result as would the preprocessed polynomial hash just described. This is because the prepended zeros have no computational effect but are used only as a conceptual device to make all vectors equal length. Thus the hash on Line 7 is also $((n + 1)/2^{120})$-AΔU. Lemma 3 tells us that zeroing seven bits as in Lines 8–9, degrades the universality guarantee by up to a factor of 3^7. To simplify the guarantee, $(3^7(n+1))/2^{120} < n/2^{107}$. $\square$

3.5 Stage 3 – Inner-Product

Line 10 of Figure 2 is a straightforward application of the inner-product hash from Section 2. It is a hash family with domain $\mathbb{Z}_{2^{61}-1} \times \mathbb{Z}_{2^{61}-1}$ and codomain $\mathbb{Z}_{2^{61}-1}$. Choosing a random function from the hash family is done by choosing a random $(k_1, k_2) \in \mathbb{Z}_{2^{61}-1} \times \mathbb{Z}_{2^{61}-1}$. The output from the second hashing phase is a pair of values less than 2^{60}, so no adjustment is needed. The following proposition needs no further proof.

Proposition 6. *Line 10 of Figure 2 defines a* $(1/(2^{61}-1))$*-AΔU hash family over* $\mathbb{Z}_{2^{61}-1} \times \mathbb{Z}_{2^{61}-1}$.

Putting it Together. Lines 1–10 of Figure 2 define VHASH as the composition of three universal hash functions. The properties of composed hash functions are well known [4,13]. If H_1 is an ε_1-AU family of hash functions with codomain A, and H_2 is an ε_2-AU family of hash functions with domain B where $A \subseteq B$, then $H = \{h_2 \circ h_1 \,|\, h_1 \in H_1, h_2 \in H_2\}$ is $(\varepsilon_1 + \varepsilon_2)$-AU. If H_2 is ε_2-AΔU, then H is $(\varepsilon_1 + \varepsilon_2)$-A$\Delta$U. This leads immediately to the result of Theorem 1.

If an application needs collision probabilities less than those of VHASH, then VHASH could be applied to given messages twice, using a different key each time. Alternatively, Figure 3 gives a hash function VHASH-128 based on the same principles as VHASH, but producing 128-bit outputs without the need for significantly more internal key than VHASH. Although no proof of correctness is given here, the arguments mirror those of VHASH. VHASH-128 is $(\ell/b)2^{-118}$-AΔU.

3.6 VHASH Performance

The performance of VHASH is influenced by many factors, the most important being how efficiently the host architecture multiplies 64-bit and adds 128-bit quantities. The Athlon 64 and recently released Intel Core 2 architectures — both 64-bit and designed for high "performance-per-watt" — are very efficient in these operations and so perform at the level described in this paper. Architectures which do not support fast 64-bit multiplication and multi-precision addition do not execute VHASH as quickly. Consider multiplication of 64-bit operands into a 128-bit result. On the Athlon 64 this can be done using a single instruction with a latency of five cycles, and VHASH hashes at a peak of 0.5 cpb. Intel's 64-bit NetBurst architecture (eg, "Nacona") can also perform the multiplication in a single instruction, but has a latency of 12 cycles, resulting in a VHASH peak of 1.4 cpb. The PowerPC 970 requires two instructions to complete a 64-bit multiplication, with a total latency of 13 cycles, and VHASH peaks at 1.0 cpb. The PowerPC version is faster than the NetBurst version due to NetBurst's horrible multiprecision addition latencies which also impact performance, but to a lesser extent than multiplication.

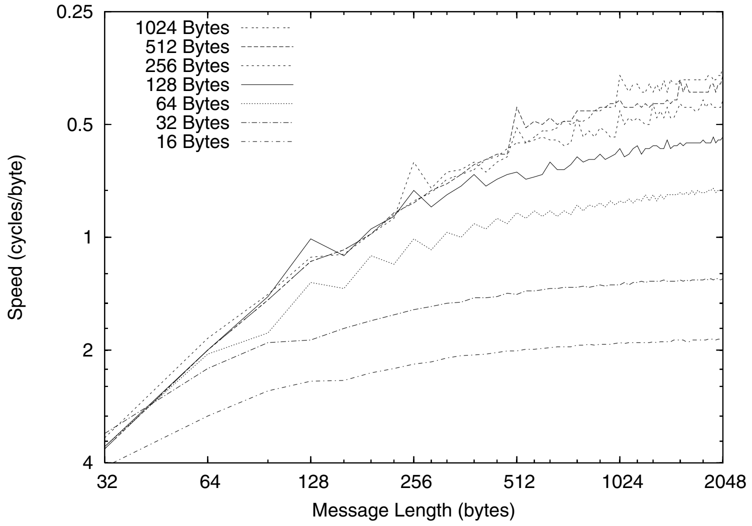


Fig. 4. Performance measured in Athlon 64 cycles per byte of message hashed for various NH key lengths and message lengths. Gains diminish greatly beyond 128 bytes.

VHASH slows down significantly on 32-bit architectures. Computing a 64-bit multiplication on a 32-bit architecture using the primary-school multiprecision multiplication algorithm requires four 32-bit multiplications and several multi-precision additions to produce a 128-bit result. On the Motorola PowerPC 7450, which has a six cycle latency per 32-bit multiplication, VHASH peaks at 5.0 cpb. On a 32-bit Intel NetBurst architecture, 32-bit multiplication latency is 11 cycles, but use of SSE vector instructions allows for a peak speed of 6.4 cpb. Clearly, VHASH benefits from architectures which multiply 64-bit registers fast, enabling exceptional VHASH performance.

Other hash functions are somewhat less variable across the mentioned architectures. UHASH and SHA1 were designed for 32-bit architectures, and so moving from 64-bit to 32-bit architectures has no inherent disadvantage. Poly1305 is multiplication based, but uses a processor's floating point unit and so is less affected by changes to general purpose register width. For all three hash functions, it is the efficiency of the processor implementation which will have the greatest impact. For example, Poly1305 has peak performance on a 64-bit PowerPC 970 of 6.6 cpb and 7.3 cpb on a slightly less efficient 32-bit PowerPC 7410. On the 64-bit Athlon 64 Poly1305 peaks at 3.1 cpb while it peaks on the much less efficient 32-bit Intel NetBurst at 5.2 cpb.

Other factors having significant affect on VHASH performance are the size of the message being hashed and the length b of the key used in the first (NH) stage of hashing. Figure 4 shows how these parameters affect performance on the Athlon 64 as measured in cycles per byte. Hashing overhead is amortized over all bytes being hashed, so as message lengths increase, overhead contributes less. Also, increasing the length of the Stage 1 NH key reduces the amount of data hashed by Stages 2 and 3. Since Stage 1 is much faster than the later stages, increasing the NH key length improves performance on longer messages.

Acknowledgements

The author wishes to thank the anonymous reviewers of SAC 2006 and FSE 2006 for their helpful reviews, especially in urging a closer look at performance on 32-bit architectures. Also, Phil Rogaway's comments and Joe Olivas's assitance in gathering timing data were timely and quite helpful. Thanks!

References

1. Afanassiev, V., Gehrmann, C., Smeets, B.: Fast message authentication using efficient polynomial evaluation. In: Biham, E. (ed.) FSE 1997. LNCS, vol. 1267, pp. 190–204. Springer, Heidelberg (1997)
2. Bernstein, D.: Stronger security bounds for Wegman-Carter-Shoup authenticators. In: Cramer, R.J.F. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 164–180. Springer, Heidelberg (2005)
3. Bernstein, D.: The Poly1305-AES message-authentication code. In: Gilbert, H., Handschuh, H. (eds.) FSE 2005. LNCS, vol. 3557, pp. 32–49. Springer, Heidelberg (2005)
4. 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)
5. Black, J., Halevi, S., Krawczyk, H., Krovetz, T., Rogaway, P.: UMAC: Fast and secure message authentication. In: Wiener, M.J. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 216–233. Springer, Heidelberg (1999)
6. Boesgaard, M., Christensen, T., Badger, Z.E.: 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)
7. Carter, L., Wegman, M.: Universal classes of hash functions. J. of Computer and System Sciences 22, 265–279 (1981)
8. Cormen, T., Leiserson, C., Rivest, R., Stein, C.: Introduction to algorithms, Section 11.3.3. MIT Press, Cambridge (2001)
9. 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)
10. Knuth, D.: The Art of Computer Programming. In: Seminumerical Algorithms, 3rd edn., vol. 2, pp. 486–489. Addison-Wesley, Reading (1998)
11. Krovetz, T., Rogaway, P.: Fast universal hashing with small keys and no preprocessing: The PolyR construction. In: Information Security and Cryptology – ICICS 2000, pp. 73–89. Springer, Heidelberg (2000)
12. 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)
13. Stinson, D.: Universal hashing and authentication codes. Designs, Codes and Cryptography 4, 369–380 (1994)
14. Wegman, M., Carter, L.: New hash functions and their use in authentication and set equality. J. of Computer and System Sciences 18, 143–154 (1979)

Some Notes on the Security of the Timed Efficient Stream Loss-Tolerant Authentication Scheme

Goce Jakimoski[⋆]

Department of Electrical and Computer Engineering
Stevens Institute of Technology, Burchard 212, Hoboken, NJ 07030, USA

Abstract. RFC4082 specifies the Timed Efficient Stream Loss-tolerant Authentication (TESLA) scheme as an Internet standard for stream authentication over lossy channels. In this paper, we show that the suggested assumptions about the security of the building blocks of TESLA are not sufficient. This can lead to implementations whose security relies on some obscure assumptions instead of the well-studied security properties of the underlying cryptographic primitives. Even worse, it can potentially lead to insecure implementations. We also provide sufficient security assumptions about the components of TESLA, and present a candidate implementation whose security is based on block ciphers resistant to related-key cryptanalysis.

Keywords: message authentication, multicast stream authentication, TESLA, cryptanalysis, block ciphers, related-key attacks.

1 Introduction

While most network applications are based on the client-server paradigm and make use of point-to-point packet delivery, many emerging applications are based on the group communications model. In particular, a packet delivery from one or more authorized sender(s) to a possibly large number of authorized receivers is required. One such class of applications is the class of multicast stream applications.

Streams of data are bit sequences of a finite, but a priori unknown, length that a sender sends to one or more recipients. They occur naturally when the buffer/memory is shorter than the message, or when real-time processing is required. Digitalized audio and video are the most common multicast stream applications. However, streams are quite common in financial applications as well. Whether it be stock quotes, customer related data or other market data feeds, the volumes of this data are growing rapidly, and the data takes the form of continuous data streams rather than finite stored data sets.

[⋆] This work was supported in part by the National Science Foundation under the grant ANI-0087641.

© Springer-Verlag Berlin Heidelberg 2007

The problems of stream authentication and stream signing have been extensively studied in the past years. Gennaro and Rohatgi [11] have proposed a stream signing scheme based on a chain of one-time signatures. A similar scheme has been presented by Zhang [28] for authentication in routing protocols. Various schemes were proposed subsequently [8,1,27,2,23,6,24] culminating with the recent adoption of TESLA as an Internet standard [19]. TESLA is also a basis for other Internet drafts (e.g., [7]), and its security and efficiency analysis can be found in [16,17,18,20].

The goal of this paper is to point out some flaws in the specification and security analysis of TESLA. Although the basic design principles of TESLA are not flawed, the suggested security assumptions about the underlying cryptographic do not provide provable security. Namely, we were able to construct examples of insecure TESLA implementations whose underlying building blocks satisfy the suggested security assumptions. We also show that provable security can be obtained by using stronger assumptions, and present an implementation whose security is based on a related-key model of a block cipher.

The outline of the paper is following. Some preliminaries are given in Section 2. In Section 3, we present examples of insecure TESLA constructions that are built using secure components. Sufficient conditions and a security proof are provided in Section 4. We propose an efficient implementation based on block ciphers in Section 5. The paper ends with the concluding remarks.

2 Background

2.1 TESLA

The Timed Efficient Stream Loss-tolerant Authentication scheme is a multicast stream authentication scheme proposed by Perrig et al [16]. Here, we briefly describe the mechanisms employed in TESLA to achieve loss-tolerance, fast transfer rates and dynamic packet rates.

The security of TESLA is based on the paradigm depicted in Figure 1. To authenticate the packet P_i of the stream, the sender first commits to the key value K_i by sending $H(K_i)$ in the packet P_{i-1}. The key K_i is only known to the sender, and it is used to compute a MAC on the packet P_i. After all recipients have received the packet P_i, the sender discloses the key value K_i in the packet P_{i+1}. The recipient verifies whether the received key value corresponds to the commitment and whether the MAC of the packet P_i computed using the received key value corresponds to the received MAC value. If both verifications are successful, the packet P_i is accepted as authentic. Note that P_i contains the commitment to the next key value K_{i+1}. To bootstrap the scheme, the first packet is signed using a digital signature scheme (e.g., RSA). If the packet P_{i-1} is lost, then the authenticity of the packet P_i and all subsequent packets cannot be verified since the commitment to the key K_i is lost. Similarly, if the packet P_{i+1} is lost, the authenticity of the packet P_i and all subsequent packets cannot be verified since the key K_i is lost.

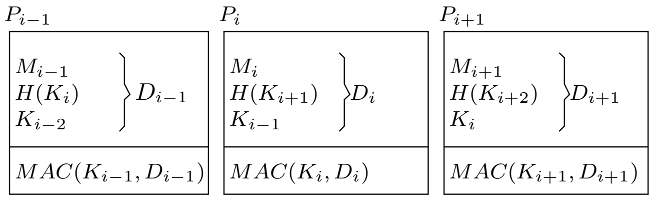


Fig. 1. The basic stream authentication scheme

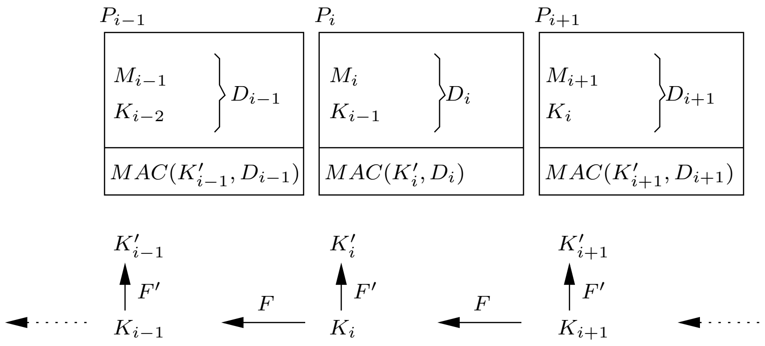


Fig. 2. TESLA Scheme II: Tolerating packet loss

Perrig et al. [16] proposed a solution to the above problem by generating the sequence of keys K_i using iterative application of a pseudo-random function to some initial value as illustrated in Figure 2. Let us denote v consecutive applications of the pseudo-random function F as $F^v(x) = F^{v-1}(F(x))$, and let $F^0(x) = x$. The sender has to pick randomly some initial key value K_n and to pre-compute n key values $K_0, \ldots, K_{n-1}$, where $K_i = F^{n-i}(K_n), i = 0, \ldots, n$. The sequence of key values is called a *key chain*. The key K'_i, which is used to authenticate the packet P_i, is derived from the corresponding key value K_i by applying the function F'. Since F is easy to compute and hard to invert, given K_i the attacker cannot compute any K_j for $j > i$. However, the recipient can compute any key value K_j from the received key value K_i, where $j < i$. Therefore, if the recipient has received a packet P_i, any subsequently received packet P_j $(j > i)$ will allow computation of $K'_i = F'(K_i)$ and verification of the authenticity of the packet P_i.

The authors suggest the function F to be implemented as $F(K_i) = f_{K_i}(0)$, where f is a target collision resistant pseudorandom function. There are no requirements imposed on the function F' in the original description of TESLA [16]. However, RFC4082 requires $F'(K_i)$ to be computed as $F'(K_i) = f'_{K_i}(1)$, where f' is a pseudorandom function. We are going to consider two cases:

– F' is an identity map. There are two main reasons why we consider this case. First, the authors make the same assumption when proving the security of TESLA in [16]. The presented proof is the only security proof of TESLA

provided by the authors. Second, if F' is an identity map or some other simple transformation, then the scheme is more efficient (i.e., we can avoid an extra PRF evaluation per packet). As shown in Section 4, the scheme can be secure even if F' is an identity map.
– $F'(K_i) = f'_{K_i}(1)$, where f' is a pseudorandom function. This is required in RFC4082.

The security of the scheme is based on the assumption that the receiver can decide whether a given packet arrived safely (i.e., before the corresponding key disclosure packet was sent by the sender). The unsafe packets are dropped. This condition severely limits the transmission rate since P_{i+1} can only be sent after every receiver has received P_i. Perrig et al [16] (TESLA Scheme III) solve this problem by disclosing the key K_i of the data packet P_i in a later packet P_{i+d}, instead of in the next packet. Another assumption made in the scheme depicted in Figure 2 is that the packet schedule is fixed or predictable, with each recipient knowing the exact sending time of each packet. This significantly restricts the flexibility of the senders. The proposed solution to this problem of dynamic packet rates is to pick the MAC key and the disclosed key in each packet only on a time interval basis. Namely, all packets sent in an interval i are authenticated using a key K_i and disclose the key K_{i-d}. This final version (TESLA Scheme IV) is the one adopted as an Internet standard. See [16,17,18,19,20] for more details.

2.2 Claimed Security of TESLA

The following theorem was given in [16].

Theorem 1. *Assume that the PRF, the MAC and the signing schemes in use are secure, and that the PRF has Target Collision Resistance property. Then, TESLA (Scheme IV) is a secure stream authentication scheme.*

To avoid complexity, the authors provide proof only for a special case when the MAC and the PRF are realized by the same function family. In their implementation, this family is the family defined by HMAC [10] when used in conjunction with MD5 [22]. However, the theorem does not require the MAC and the PRF to be realized by the same function family. We will show that the theorem does not hold in the case when the PRF and the MAC can be realized by different function families (i.e., we will disprove the theorem). Furthermore, in their proof, the authors assume that the function F' is an identity mapping. This is not the case in the RFC4082 version. Hence, their analysis does not apply to the Internet standard.

2.3 OMAC

OMAC [13] is a proven secure CBC MAC scheme that uses only one key. The evaluation of the authentication tags in OMAC is illustrated in Figure 3. The first block of the message is encrypted using a block cipher. The result is XORed with the second block and encrypted, etc. If the length of the last chunk of the message is equal to the block length n, then the last block is XORed with $L \cdot u$

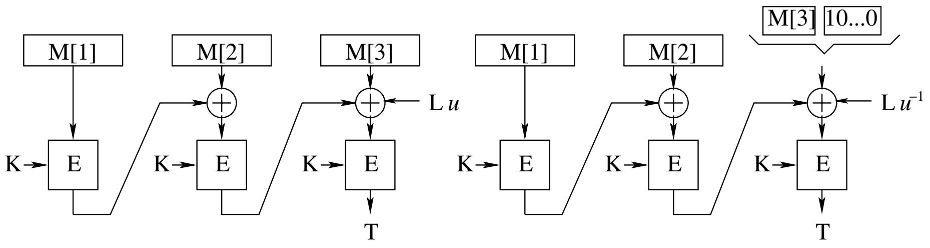


Fig. 3. One-key CBC MAC

before encryption. If the length of the last chunk of the message is less than the block length n, then 10^i padding ($i = n - 1 - |M| \bmod n$) is appended and the last block is XORed with $L \cdot u^{-1}$ before encryption. The parameter u is some known constant in GF(2^n), and $L = E_K(0^n)$ is an encryption of 0.

Let l be the key length. It was shown in [13] that if the function family $\{E_K\}_{K \in \{0,1\}^l}$ block cipher is a pseudorandom permutation family, then $\{\text{OMAC}_K\}_{K \in \{0,1\}^l}$ (the function family defined by OMAC) is a pseudorandom function family and the OMAC scheme is unforgeable.

3 Insecure TESLA Implementations Based on Secure Components

In this section, we show that the suggested assumptions about the building blocks of TESLA are not sufficient by providing examples of insecure TESLA constructions from components that satisfy those assumptions.

3.1 Permuted-Input OMAC

In order to "break" TESLA Scheme II, we introduce Permuted-input OMAC (POMAC) scheme. The scheme will be used to authenticate the packets of the stream in our insecure TESLA Scheme II implementation. It is depicted in Fig. 4. If the length of the message m is not greater than the block size n, then the authentication tag is computed as $\text{OMAC}_K(m)$. Otherwise, the message m is rotated right by n bits to derive a new message m', and the authentication tag is computed as $\text{OMAC}_K(m')$.

The unforgeability of POMAC trivially follows from the unforgeability of OMAC.

Lemma 1. *Suppose that:*

- *h is a collision resistant function (i.e., it is hard to find m_1 and $m_2 \neq m_1$ s.t. $h(m_1) = h(m_2)$), and*
- *$\{f_K\}_{K \in \{0,1\}^l}$ is a function family corresponding to an unforgeable MAC scheme.*

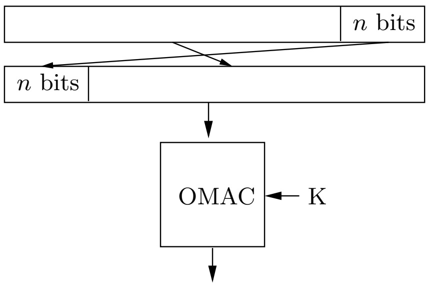


Fig. 4. Permuted-input OMAC

Then, the MAC scheme defined by the function family $\{f_K \circ h\}_{K\in\{0,1\}^l}$ is unforgeable too.

Proof. Assume that there is an adversary that can output a pair (m, a) where a is a valid authentication tag for a message m that hasn't been signed before. Since h is collision resistant, the hash value $h(m)$ must be different from the hash values of the previously signed messages. Hence, $(h(m), a)$ is a forgery for the MAC scheme defined by the function family $\{f_K\}_{K\in\{0,1\}^l}$. This contradicts our assumption that f is unforgeable. ■

Corollary 1. *If the function family $\{E_K\}_{K\in\{0,1\}^l}$ defined by the underlying block cipher is a pseudorandom permutation family, then POMAC is unforgeable.*

Proof. Follows from Lemma 1 and the facts that the initial permutation in POMAC is a bijection (i.e., collision resistant) and OMAC is unforgeable when the underlying block cipher is a pseudorandom permutation. ■

3.2 The Case When F' Is an Identity Mapping

In this section, we provide an example of an insecure TESLA construction from secure components in the case when the function F' is an identity mapping.

Suppose that the function family $\{E_K\}_{K\in\{0,1\}^n}$ is a target collision resistant pseudorandom permutation family whose members are defined on the set $\{0, 1\}^n$. Note that the length of the key is equal to the block size n. AES-128 [9] is a possible candidate. Since $\{E_K\}_{K\in\{0,1\}^n}$ is a pseudorandom permutation family, it is also a pseudorandom function family (see Proposition 3.7.3 in [12]). We will use the pseudorandom permutation E_K to generate the authentication keys as illustrated in Figure 5. The key $K_{i-1} = E_{K_i}(0^n)$ is generated by encrypting 0 using the key K_i as suggested in [16]. The MAC scheme that we use in our construction is POMAC. To encrypt the message blocks in POMAC, we use the pseudorandom permutation E_K.

The PRF and the MAC as defined above satisfy the security requirements of Theorem 1. However, the resulting stream authentication scheme is not secure. Figure 5 depicts an attack on our TESLA Scheme II example by replacing the

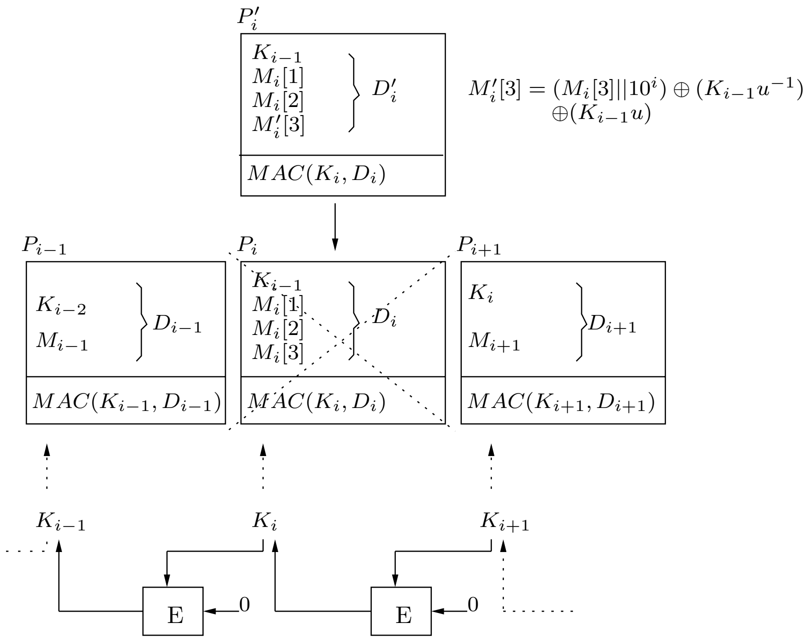


Fig. 5. Insecure TESLA implementation. MACs are computed using POMAC.

packet P_i with a packet P'_i. Without loss of generality, we assume that the message M_i consists of three chunks $M_i[1], M_i[2]$ and $M_i[3]$. The length of $M_i[1]$ and $M_i[2]$ is equal to the block length n, and the length of $M_i[3]$ is less than the block length n. This implies that $M_i[3]$ is 10^i padded and XORed with $L \cdot u^{-1}$ when computing the MAC for P_i. The forged packet P'_i is constructed by replacing $M_i[3]$ with

$$M'_i[3] = (M_i[3]||10^i) \oplus (K_{i-1} \cdot u^{-1}) \oplus (K_{i-1} \cdot u).$$

Using the equations

$$(M_i[3]||10^i) \oplus (K_{i-1} \cdot u^{-1}) = M'_i[3] \oplus (K_{i-1} \cdot u)$$

and

$$L = E_{K_i}(0^n) = K_{i-1},$$

one can easily verify that

$$\text{POMAC}(K_i, D_i) = \text{POMAC}(K_i, D'_i).$$

Note that all we need to compute P'_i is the key K_{i-1} and the message M_i. Since both the key K_{i-1} and the message M_i are disclosed in the packet P_i, we can compute P'_i before the key K_i is disclosed. Hence, we have succeeded in constructing a forgery for TESLA Scheme II.

3.3 Cryptanalysis of TESLA Scheme IV

As we mentioned earlier, the goal of upgrading TESLA Scheme II to TESLA Scheme IV was to achieve fast transfer rates and dynamic packet rates. The security of the upgraded scheme relies on the same principles as Scheme II, and the attack depicted in Figure 5 can be easily extended to the upgraded scheme. Moreover, the attack works with OMAC instead of POMAC as explained below in more detail.

There are two differences between Scheme II and Scheme IV that are relevant to our discussion. First, in Scheme IV, the same key is used to authenticate more than one packet sent to a given recipient. Second, the key K_{i-1} is revealed after the time interval i (assuming that the delay d is greater than one). However, note that the adversary can discard all but one packet in some time interval i, and then delay that packet so that the recipient gets the packet after the disclosure of K_{i-1}, but before the disclosure of K_i (i.e., the packet will be safe). Since, the adversary knows the value of K_{i-1} before handing the packet to the recipient, he can replace it with a forged one as in Figure 5.

The attack will work with OMAC instead of POMAC for the following reasons. The introduction of the POMAC scheme was motivated by the order of the message M_i and the key K_{i-1} within the packet P_i (Fig. 2). The initial permutation of POMAC swaps the message and the key so that the last block of D_i is a message block. In TESLA Scheme IV, the format of the packets is $P_j = \langle M_j, i, K_{i-d}, MAC(K'_i, M_j)\rangle$, where i is the interval during which the packet P_j was sent. Note that the MACs are computed over the messages M_j only, and the attack would work when OMAC instead of POMAC is used to compute the MACs. Hence, our analysis shows not only that the assumptions about the security properties of the building blocks of TESLA are not sufficient, but also that it is not unrealistic to expect that TESLA Scheme IV might be implemented insecurely.

3.4 The Case When F' Is Implemented Using a PRF

RFC4082 requires the function F to be implemented as $F(K) = f_K(0)$ (Section 3.2 of [19]), and F' to be implemented as $F'(K) = f'_K(1)$ (Section 3.4 of [19]), where f and f' are pseudorandom functions.

Although it seems that the scheme is secure when f and f' are identical[1], the RFC does not require f and f' to be identical. On the contrary, the use of different symbols to denote them suggests that they can be different. In this case, the new TESLA Scheme II still suffers from the flaw discussed in Section 3.2. Namely, we can view f' as a part of the key scheduling algorithm of the underlying block cipher. The function F of the insecure TESLA construction is now implemented as $F(K_i) = E_{f'_{K_i}(1)}(0)$ (see Figure 6). It is clear that $K_{i-1} = F(K_i)$ leaks the encryption of zero since $K_{i-1} = E_{K'_i}(0)$, and we can mount the same attack.

In addition to the old flaw, the modification of the scheme introduces a new one. Consider the following "naive" implementation. The function F is implemented

[1] The reader should be aware that there is no security proof provided for this case.

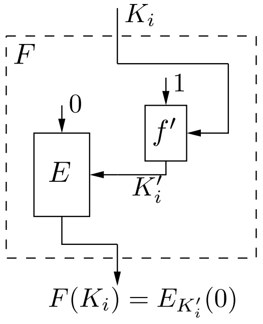


Fig. 6. The function F leaks the encryption of zero $E_{K'_i}(0)$

as $F(K_i) = f_{K_i}(0)$, where f is a target collision resistant pseudorandom function family. The function F' is implemented as $F'(K_i) = f'_{K_i}(1)$, where $f'_{K_i}(x) = f_{K_i}(x-1)$. One can easily show that f' is a pseudorandom function. It is not hard to verify that the commitment $F(K_i)$ discloses the authentication key K'_i: $F(K_i) = f_{K_i}(0) = f_{K_i}(1-1) = f'_{K_i}(1) = K'_i$. Although this implementation is very unlikely, it demonstrates the threat of exploiting the knowledge of the commitment $F(K_i)$ to compute the authentication key K'_i.

3.5 Cryptanalysis of the RFC4082 TESLA Version

The analysis presented in Section 3.4 can be extended to the TESLA version described in RFC4082. We use the same arguments as in Section 3.3. The safe packet test only checks whether a packet authenticated using a key K_i was received before the disclosure of the key K_i. Hence, the adversary can delay the packet until the key K_{i-1} is disclosed, and then replace it with a forged one. The aforementioned security flaws cannot be patched by simply modifying the safe packet test so that the receiver checks whether the packet was received before the disclosure of the key value K_{i-1}. In this case, the adversary might be able to use $K_{i-2} = F(F(K_i))$ or some previous key value to mount an attack.

4 Sufficient Assumptions About the Components of TESLA

The attacks on the insecure implementations that were presented in Section 3 are based on the following observation. The security of the MAC scheme that is used to authenticate the packets is proven in a setting where the adversary has access to a signing oracle and a verifying oracle. In the case of TESLA, we have a different setting. Now, the adversary has access to an additional oracle that computes the commitment $F(K)$ to the secret key K which is used by the MAC scheme. The adversary can exploit the knowledge of $F(K)$ to construct a forgery.

It is clear from the discussion above that we need to make an additional assumption about the function F and the MAC scheme. Namely, the MAC scheme must remain secure even when the commitment of the secret key used by the MAC scheme is revealed.

Definition 1. *A MAC scheme is* known F-commitment unforgeable *if there is no efficient adversary that given a commitment $F(K)$ of the secret key that is in use can break the MAC scheme with non-negligible probability.*

An example, which demonstrates that one can achieve known F-commitment unforgeability, is provided in Section 5.2.

We also make the following minor modification of TESLA Scheme II. Each time a stream is authenticated, the sender selects a unique number N_s (e.g., using a counter) which is securely communicated to the recipients. The number N_s is included as part of the authenticated data in each packet of the stream including the bootstrap packet. So, we assume that the format of the messages is $M_i = \langle N_s, i, C_i \rangle$, where C_i is the actual chunk of the stream [2] [3].

The following theorem holds for the security of the slightly modified TESLA Scheme II.

Theorem 2. *Suppose that:*

1. *the digital signature scheme, which is used to bootstrap TESLA, is unforgeable,*
2. *the function $F(K) = f_K(0)$, where f is a pseudorandom function, is collision resistant,*
3. *the MAC scheme, which is used to authenticate the chunks of the stream, is known F-commitment unforgeable, and*
4. *F' is an identity mapping.*

Then, TESLA Scheme II is a secure multicast stream authentication scheme.

The proof is given in Appendix A.

Note that F' is an identity mapping, while in RFC4082, F' is realized using a pseudorandom function. Hence, the scheme that is analyzed here is somewhat more efficient than the Internet standard.

The requirement for collision resistance of the function F can be slightly weakened. Assuming that there is a bound on the number of packets within a stream, it is not hard to show that TESLA Scheme II is secure when the function F is collision resistant in the following sense: Given a randomly selected value K and a bound $L \geq 1$, it is hard to find K' and a positive integer $l \leq L$ such that $F^l(K) = F(K')$ and $F^{l-1}(K) \neq K'$. A function that satisfies the aforementioned property is said to be *bounded iteration collision resistant.*

[2] TESLA does not provide ordering of the packets that are authenticated using the same key. We use sequence numbers to prevent malicious reordering of the packets.

[3] To reduce the communication overhead one can communicate N_s only once, and then just use it to compute the signature and the MACs.

5 A Candidate Implementation of TESLA

In this section, we propose an implementation that uses block ciphers to realize the different components of TESLA.

5.1 CKDA-PRPs

When cryptanalyzed, block ciphers are not considered secure unless they are resistant to related-key attacks [4]. A theoretical treatment of block ciphers resistant to related-key attacks was given in [3], where it was shown that under some restrictions one can achieve resistance to related-key attacks. We are going to use a model of a more specific case: the adversary can query oracles that use keys whose difference was chosen by the adversary (e.g., related-key differential cryptanalysis [15,14]).

We define a CKDA secure (i.e., secure against Chosen Key Difference Attacks) pseudorandom permutation family as a pseudorandom permutation family such that one cannot tell apart a pair of permutations randomly selected from the family and a pair of permutations from the family whose index (key) difference is $c \neq 0$, where c is selected by the adversary. A CKD test is a Turing machine A with access to four oracles $\mathcal{E}_1$, $\mathcal{D}_1$, $\mathcal{E}_2$ and $\mathcal{D}_2$. A selects a non-zero l-bit string c. The oracle $\mathcal{E}_1$ is selected to be a random permutation E_K from the permutation family $\{E_K\}_{K\in\{0,1\}^l}$, and the oracle $\mathcal{D}_1$ is selected to be its inverse. According to a secret random bit b, the oracle $\mathcal{E}_2$ is selected to be either the permutation $E_{K\oplus c}$ or a random permutation $E_{K\oplus r}$, where c is the public non-zero constant and r is a random bit string of length l (i.e., $K \oplus r$ is random and not related to K). The oracle $\mathcal{D}_2$ computes the inverse of $\mathcal{E}_2$. The algorithm A outputs 0 or 1. The advantage of the CKD test is defined as

$$\mathrm{Adv}_A((E_K, E_{K\oplus c}), (E_K, E_{K\oplus r})) = \frac{1}{2}(E[A^C] - E[A^R])$$

where $E[A^C]$ (resp., $E[A^R]$) is the probability that A will output 1 when the difference between the secret keys is a known non-zero constant (resp., random l-bit string).

Definition 2. *The pseudorandom permutation family $\{E_K\}_{K\in\{0,1\}^l}$ is a $[t, q, \epsilon]$-*secure CKDA pseudorandom permutation family (or $[t, q, \epsilon]$-secure CKDA-PRP) *if there is no CKD test that runs in at most t time, sends at most q queries to the oracles and has at least ϵ advantage.*

5.2 TESLA Implementation Via CKDA-PRPs

The following theorem provides a function F and a MAC scheme such that the MAC scheme is known F-commitment unforgeable.

Theorem 3. *Let the function family $\{E_K\}_{K\in\{0,1\}^n}$ corresponding to the block cipher used by OMAC be CKDA secure. Let $F : \{0,1\}^n \to \{0,1\}^n$ be defined as $F(K) = E_{K\oplus c}(0)$, where $c = 0^{n-1}1$. Then, OMAC is a known F-commitment unforgeable MAC scheme.*

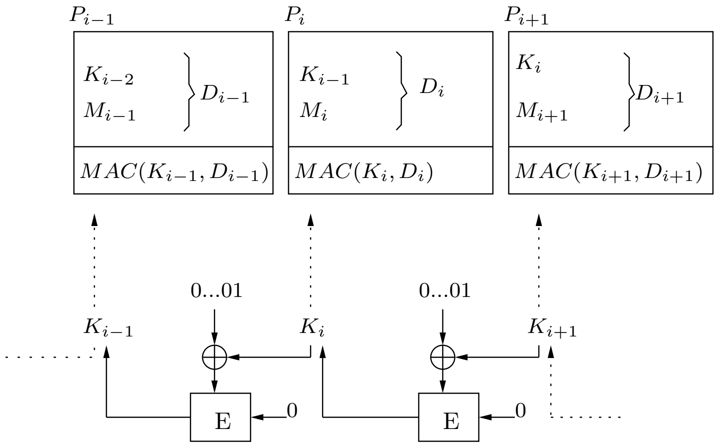


Fig. 7. TESLA implementation using a block cipher resistant to related-key cryptanalysis

Proof. An adversary A_1 that given a commitment $F(K')$ to some randomly selected key K' can break OMAC with probability ϵ can be easily converted into an adversary A_2 that can break OMAC with the same probability. In particular, A_2 can randomly select the key value K' and submit the commitment $F(K')$ to A_1. A_1's output will be A_2's output. Since OMAC is unforgeable, there is no adversary that can break OMAC with significant probability given a commitment to a randomly select key.

Now, assume that there is an adversary A_3 that can break OMAC given the commitment $F(K)$ to the secret key K that is in use. We can construct a CKD test as follows. We run the adversary A_3 and answer its queries by querying the oracles $\mathcal{E}_1$ and $\mathcal{E}_2$. If A_3 manages to produce a forgery we output 1, otherwise we output 0. Obviously, the advantage of the CKD test will be significant since the probability $E[A^C]$ is significant (OMAC is not known F-commitment unforgeable) and the probability $E[A^R]$ is small (OMAC is unforgeable). ■

The implementation that we propose here is depicted in Figure 7. It is similar to the insecure implementation shown in Figure 5. The only difference is that the key value K_{i-1} is derived by encrypting zero using the key $K_i \oplus 0^{n-1}1$ instead of the key K_i. A similar secure variant of the insecure implementation can be obtained by using a function F' that derives the key K_i' by flipping the last bit of K_i instead of using an identity map. We must note that the security of the proposed scheme is also based on the assumption that $h(x) = E_x(0)$ is collision resistant (one of the assumptions made in Theorem 2). While there are some constructions and possibility results regarding hash functions based on block ciphers [21,5], we are not aware of any results regarding the collision resistance of $h(x) = E_x(0)$ where E is some widely used cipher with relatively large block size (e.g., AES).

6 Conclusion

We have shown that the assumptions about the components of TESLA are not sufficient and can potentially lead to insecure implementations. We also provided sufficient conditions for the security of the scheme and proposed an implementation based on block ciphers.

References

1. Anderson, R., Bergadano, F., Crispo, B., Lee, J., Manifavas, C., Needham, R.: A New Family of Authentication Protocols. ACM Operating Systems Review 32(4), 9–20 (1998)
2. Bergadano, F., Cavagnino, D., Crispo, B.: Chained Stream Authentication. In: Proceedings of Selected Areas in Cryptography 2000, pp. 142–155 (2000)
3. Bellare, M., Kohno, T.: A Theoretical Treatment of Related-Key Attacks: RKA-PRPs, RKA-PRFs, and Applications. In: Biham, E. (ed.) Advances in Cryptology – EUROCRPYT 2003. LNCS, vol. 2656, pp. 491–506. Springer, Heidelberg (2003)
4. Biham, E.: New Types of Cryptanalytic Attacks Using Related Keys. Journal of Cryptology 7(4), 229–246 (1994)
5. Black, J., Rogaway, P., Shrimpton, T.: Black-Box Analysis of the Block-Cipher-Based Hash-Function Constructions from PGV. In: Yung, M. (ed.) CRYPTO 2002. LNCS, vol. 2442, pp. 225–320. Springer, Heidelberg (2002)
6. Canneti, R., Garay, J., Itkis, G., Micciancio, D., Naor, M., Pinkas, B.: Multicast security: A taxonomy and some efficient constructions. In: Infocom '99 (1999)
7. Carrara, E., Baugher, M.: The Use of TESLA in SRTP. Internet draft, `http://ietfreport.isoc.org/ids-wg-msec.html`
8. Cheung, S.: An Efficient Message Authentication Scheme for Link State Routing. In: Proceedings of the 13th Annual Computer Security Application Conference (1997)
9. FIPS PUB 197, The Advanced Encryption Standard
10. FIPS PUB 198, The Keyed-Hash Message Authentication Code (HMAC)
11. Gennaro, R., Rohatgi, P.: How to Sign Digital Streams. In: Kaliski Jr., B.S. (ed.) CRYPTO 1997. LNCS, vol. 1294, pp. 180–197. Springer, Heidelberg (1997)
12. Goldreich, O.: Foundations of Cryptography. Cambridge University Press, Cambridge (2001)
13. Iwata, T., Kurosawa, K.: OMAC: One-Key CBC MAC. In: Johansson, T. (ed.) FSE 2003. LNCS, vol. 2887, pp. 129–153. Springer, Heidelberg (2003)
14. Jakimoski, G., Desmedt, Y.: Related-key Differential Cryptanalysis of 192-bit Key AES Variants. In: Matsui, M., Zuccherato, R.J. (eds.) SAC 2003. LNCS, vol. 3006, pp. 208–221. Springer, Heidelberg (2004)
15. Kelsey, J., Schneier, B., Wagner, D.: Related-key Cryptanalysis of 3-WAY, Biham-DES, CAST, DES-X, NewDES, RC2 and TEA. In: Proceedings of ICICS'97, pp. 233–246. Springer, Heidelberg (1997)
16. Perrig, A., Canneti, R., Tygar, J.D., Song, D.: Efficient Authentication and Signing of Multicast Streams Over Lossy Channels. In: Proceedings of the IEEE Security and Privacy Symposium (2000)
17. Perrig, A., Canneti, R., Song, D., Tygar, J.D.: Efficient and Secure Source Authentication for Multicast. In: Proceedings of the Network and Distributed System Security Symposium (2001)

18. Perrig, A., Canneti, R., Tygar, J.D., Song, D.: The TESLA Broadcast Authentication Protocol. RSA CryptoBytes 5(2) (2002)
19. Perrig, A., Song, D., Canneti, R., Tygar, J.D., Briscoe, B.: Timed Efficient Stream Loss-Tolerant Authentication (TESLA): Multicast Source Authentication Transform Introduction. Internet Request for Comments, RFC 4082 (June, 2005)
20. Perrig, A., Tygar, J.D.: Secure Broadcast Communication in Wired and Wireless Networks. Kluwer Academic Publishers, Dordrecht (2002)
21. Preneel, B., Govaerts, R., Vandewalle, J.: Hash functions based on block ciphers: A synthetic approach. In: Stinson, D.R. (ed.) CRYPTO 1993. LNCS, vol. 773, Springer, Heidelberg (1994)
22. Rivest, R.L.: The MD5 message digest algorithm. Internet Request for Comments, RFC 1321 (April 1992)
23. Rohatgi, P.: A compact and fast hybrid signature scheme for multicast packet authentication. In: 6th ACM Conference on Computer and Communications Security, November 1999 (1999)
24. Syverson, P.F., Stubblebine, S.G., Goldschlag, D.M.: Unlinkable serial transactions. In: FC 1997. LNCS, vol. 1318, Springer, Heidelberg (1997)
25. Wang, X., Lai, X., Feng, D., Chen, H., Yu, X.: Cryptanalysis for Hash Functions MD4 and RIPEMD. In: Cramer, R.J.F. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 1–18. Springer, Heidelberg (2005)
26. Wang, X., Yu, H.: How to Break MD5 and Other Hash Functions. In: Cramer, R.J.F. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 19–35. Springer, Heidelberg (2005)
27. Wong, C.K., Lam, S.S.: Digital Signatures for Flows and Multicasts. In: Proceedings of IEEE ICNP '98 (1998)
28. Zhang, K.: Efficient Protocols for Signing Routing Messages. In: Proceedings of the Symposium on Network and Distributed System Security (1998)

A Proof of Theorem 2

Assume that the adversary can break the stream authentication scheme. In other words, the adversary in cooperation with some of the recipients can trick another recipient u to accept a forged packet of the stream as valid.

Let i be the smallest integer such that the contents D'_i is accepted as valid by the recipient u when the original contents D_i is different from D'_i. There are three possible events:

Event 1. If i is zero, then the adversary has managed to forge the bootstrap packet which was signed using a digital signature scheme.

Event 2. If i is greater than zero and the key that u used to verify the validity of D'_i is equal to the original key K_i, then the adversary has managed to produce a forgery for the message authentication scheme due to the uniqueness of $\langle N_s, i\rangle$.

Event 3. If i is greater than zero and the key K^f_i that u used to verify the validity of D'_i is different than the original key K_i, then the adversary has managed to find a collision for the function F. Let $K^f_i, K^f_{i-1} = F(K^f_i), \dots$ be a key chain derived from K^f_i, and let $K_i, K_{i-1} = F(K_i), \dots$ be a key chain derived from K_i. The user u verified the validity of the key value K_i^f by checking whether $F^l(K_i^f)$ is equal to some previously authenticated key value K_{i-l}^f. Since i is the smallest index of a packet whose contents D_i' is different from the original contents D_i, the received key value K_{i-l}^f must be equal to the original key value $K_{i-l} = F^l(K_i)$. Hence, there is an index $i - l \leq j < i$ s.t. $K_{j+1} \neq K_{j+1}'$ and $F(K_{j+1}) = K_j = K_j' = F(K_{j+1}')$.

Given an efficient adversary A_{SA} that breaks the stream authentication scheme with significant probability, we will construct an adversary A_{S} for the signature scheme, an adversary A_{MAC} for the MAC scheme and an adversary A_{F} for the function F, and show that at least one of these adversaries has significant success probability. All three adversaries simulate the network using sets of read and write tapes for the users and for the adversary. They differ in the following aspects:

1. The adversary for the signature scheme answers the stream signing queries by randomly selecting initial key values, computing the key chains and using the signing oracle for the bootstrap packets. Whenever A_{SA} manages to forge a bootstrap packet, A_{S} outputs the forged message/signature pair. Otherwise, it outputs a randomly selected message/signature pair.
2. The adversary for the MAC scheme guesses which stream will be forged and what will be the smallest index i of a forged packet within the stream. If the guess is that the stream will not be forged, then A_{MAC} answers the stream signing query by randomly selecting the initial key value. Otherwise, the adversary uses the given value $K_{i-1} = F(K_i)$ to derive the keys that will be used to authenticate the packets $P_1, \ldots P_{i-1}$, and computes the MAC for the packet P_i by submitting a query to the (MAC) signing oracle. If the adversary for the stream scheme manages to forge the i-th packet, then the adversary for the MAC scheme outputs the forged message/MAC pair. Otherwise, it outputs a randomly selected message/MAC pair.
3. The adversary for the function F answers the stream signing queries by randomly selecting initial key values, computing the key chains and using a private key when signing the bootstrap packets. In the case when Event 3 occurs, A_{F} finds and outputs a pair of key values that collide. Otherwise, it outputs two randomly selected key values.

It is easy to show that if the probabilities of Event 1 and Event 3 are significant, then the success probabilities of the corresponding adversaries A_{S} and A_{F} are significant too. To derive a relation between the probability of Event 2 and A_{MAC}, we need the following Lemma.

Lemma 2. *If f is a pseudorandom function, then there is no efficient algorithm that can distinguish between a random key value and the key value $F^l(K)$ derived from a secret random key K by $l \geq 1$ iterations of the function F.*

Proof. We can prove the Lemma by induction. If there is an algorithm that can tell apart between $F(K) = f_K(0)$ and a random key value, then we can construct an algorithm that can distinguish between the function family $\{f_K\}$ defined by

f and the random function family. Now, assume that there is no algorithm that can tell apart between the key $K_{l-1} = F^{l-1}(K)$ and a random key value. Since the function f and the key K_{l-1} are pseudorandom, the key $K_l = f_{K_{l-1}}(0)$ will be indistinguishable from a random key too. ■

Assume that n_s and L are the maximum number of streams and the maximum number of packets within a single stream respectively. The probability that A_{MAC} will guess the forged stream and the index i of the first forged packet within the stream is $\frac{1}{n_s L}$. According to Lemma 2, there is no efficient algorithm that can distinguish with significant probability between the secret key K_i used by the MAC scheme and a key that is derived from some initial key value by $l-i$ iterations of the function F. Hence, if the probability ϵ of Event 2 is significant, then the success probability of A_{MAC} will be approximately $\frac{\epsilon}{n_s L}$.

Constructing an Ideal Hash Function from Weak Ideal Compression Functions

Moses Liskov

Computer Science Department
The College of William and Mary
Williamsburg, Virginia, USA
mliskov@cs.wm.edu

Abstract. We introduce the notion of a *weak* ideal compression function, which is vulnerable to strong forms of attack, but is otherwise random. We show that such weak ideal compression functions can be used to create secure hash functions, thereby giving a design that can be used to eliminate attacks caused by undesirable properties of compression functions.

We prove that the construction we give, which we call the "zipper hash," is *ideal* in the sense that the overall hash function is indistinguishable from a random oracle when implemented with these weak ideal building blocks.

The zipper hash function is relatively simple, requiring two compression function evaluations per block of input, but it is not streamable. We also show how to create an ideal (strong) compression function from ideal weak compression functions, which can be used in the standard iterated way to make a streamable hash function.

Keywords: Hash function, compression function, Merkle-Damgård, ideal primitives, non-streamable hash functions, zipper hash.

1 Introduction

The design of hash functions is a long-studied problem that has become recently more relevant because of significant attacks against commonly-used hash functions [22,20,21,19,1]. It is much easier to create *collision functions*, which take input of a particular size and produce output of a reduced size, than a full hash function directly. It is common practice to follow the basic concept of the Merkle-Damgård construction [6,14]: composing a compression function with itself, each time incorporating a block of the message, until the entire message is processed. If f is the compression function and x is an input divisible into l blocks of the appropriate size, then

$$H(x) = f(x_l, f(x_{l-1}, \ldots, f(x_1, IV)) \ldots)$$

is the basic iterated hash function. There are two main ways in which this basic method has evolved: first of all, to handle messages of arbitrary length, a message may have to be padded so that the block size divides the length. In addition, the length of the initial message is included in the padding: this, along with fixing

© Springer-Verlag Berlin Heidelberg 2007

an IV, is called Merkle-Damgård strengthening. Second, a finalization function g is often used after all the message blocks have been processed. Among other properties, this allows the output size of the compression function to be different from the output size of the hash function.

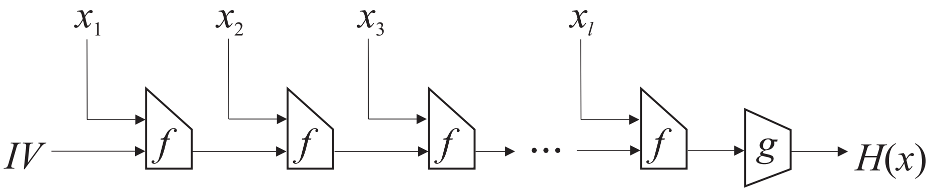


Fig. 1. The modern iterated hash function

The iterated hash function construction is elegant and natural, and is additionally attractive in that it is *streamable*, that is, a message may be hashed piece by piece with a small, finite amount of memory. Furthermore, this construction is known to be collision-resistant as long as the underlying compression function is collision-resistant [6,14]. However, there are reasons to question the iterated hash function design now.

An underlying theme in the recent high-profile attacks on hash functions has been the use of weaknesses in the compression function to build up an effective attack against the overall hash function. Furthermore, many attacks have been published recently that accomplish interesting black-box attacks against iterated hash functions once compression-function weaknesses have been found.

Here we summarize some known black-box attacks against iterated hash functions. Let n be the length of the output of a hash function H.

- **Second collision attack.** The basic attack goal here is to find a second collision on H once we have found a first collision on H. In a well-known attack, this is trivial for basic iterated hash functions: if $H(x) = H(y)$ then for all strings z, $H(x||z) = H(y||z)$ is another collision. Merkle-Damgård strengthening does not solve this problem completely, since the attack still works if $|x| = |y|$ and z contains the correct padding. [16,13]
- **Joux multicollision attack** [10]. It is easier than expected to find multicollisions: that is, a set of many distinct inputs that all hash to the same value. For a generic hash function, finding a t-way collision should require hashing an expected $2^{n\cdot(t-1)/t}$ messages. However, Joux showed that finding a t-way collision can also be done by making $(\log_2 t)2^{k/2}$ compression function queries, where k is the output size of the compression function. Essentially, the attack is to find one-block collisions for the compression function that can be chained together (by a brute force birthday attack). Once we have r such collisions, we can generate a 2^r-way collision by choosing one input for each colliding pair.
- **Fixed-point attack** [12,7]. The goal here is to come up with a second preimage for one of a set of known messages. If the target set is of size 2^t,

it is easy to see that a second preimage can be found in a generic attack in time 2^{n-t}. This attack improves upon this by finding *expandable messages* based on fixed points for the collision function. Among other examples, the compression function in any Davies-Meyer block cipher-based hash function (such as the SHA family as well as MD4 and MD5) is susceptible to fixed-point attacks[12]. This allows an attack where, after hashing 2^t mesage *blocks*, a second preimage can be found in time $t2^{n/2} + 1 + 2^{n-t+1}$. Fixed points are used to circumvent Merkle-Damgård strengthening; with fixed points, one can build "expandable messages," which let us recover a second preimage of the correct length.
– **The "herding" attack** [11]. This is an attack against the use of a hash function for commitments. The idea is to find a 2^t-way collision at a value $H(x)$, and then find a preimage of a commitment $H(x)$ that starts with an arbitrary z by trying random values y until $H(z||y)$ is one of the 2^t-way collisions.

In order to combat attacks like the Joux attack and the Kelsey-Kohno herding attack, Lucks proposed that the internal state of an iterated hash function should be larger than the output, thus preventing the usefulness of finding compression function collisions by brute force [13]. Lucks proposed *double-pipe hash* as a way to implement this, using two parallel compression function computations per block of message, in order to increase the size of the internal state. Lucks proved that, assuming the underlying compression function was ideal (i.e., a random oracle), the double-pipe hash compression function yields a collision-resistant hash function.

At the core of Lucks' paper, however, was an even more important idea: that we should attempt to design hash functions that remain secure even when the compression functions on which they are based can be attacked.

We seek to improve on the work of Lucks in two ways. First, following the work of Coron, Dodis, Malimaud, and Puniya [4], we will prove that our construction is not only collision-resistant, but in fact indistinguishable from a random oracle, assuming the building blocks are ideal. Coron et al. show that the basic Merkle-Damgård construction is not ideal in the sense that even with an ideal compression function, it is impossible to prove that the hash function is indistinguishable from a random oracle. However, with an ideal finalization function (among other alternate modifications), iterated hash functions can be shown to be indistiguishable from a random oracle when implemented with ideal components. Assuming individual components to be ideal has been established as a reasonable model for the analysis of hash functions for some time [2]. The work of Coron et al. has set a higher standard for hash functions analyzed on the basis of ideal primitives, and we aspire to that standard.

Second, Lucks only attempts to make a hash function resilient to brute-force collision attacks against the compression function. It would be better to make a hash function resilient to actual *flaws* in the compression function as well. Therefore, we will weaken our assumptions about the underlying compression function as much as possible. We will still consider an ideal form of a compression function, but we will explicitly allow attacks against it, in order to model a weak but minimally secure compression function.

1.1 Our Results

In this paper, we formalize the notion of a weak ideal compression function, and show that such compression functions can be used to make stronger ideal primitives. Namely, we give a construction we call the "zipper hash" that makes an ideal hash function from weak ideal compression functions. The zipper hash is a very simple and elegant design; it requires $2l$ compression function evaluations for an l-block input. (Additionally, this concept of a weak ideal primitive may be of independent interest.)

Then, we go on to use weak ideal compression functions to make an ideal compression function. This construction is based on the zipper hash, and requires four compression function evaluations to run. We show that the Lucks double-pipe compression function is *not* an ideal construction, but offer a simple modification of it that is ideal. Thus, the compression function we consider comparable requires eight underlying compression function evaluations per block of input.

Finally, we analyze the efficiency of our schemes. We go on to make a case for considering non-streamable hash functions like our zipper hash in practice. We note that streamable hash functions (0with constant-size state) always follow the essential Merkle-Damgård structure, so to avoid general attacks against iterated hash functions, one must consider non-streamable hash functions.

2 Notation and Definitions

2.1 Hash Functions and Compression Functions

Before we explore these issues, we must give a basic introduction to the concept of hash functions and compression functions. An n-bit *hash function family* is a family of functions $H : \mathcal{K} \times \{0,1\}^* \to \{0,1\}^n$ where $\mathcal{K}$ represents the set of "keys" from which one is chosen at random. Note that hash functions must be defined as families: any specific hash function $H : \{0,1\}^* \to \{0,1\}^n$ cannot be *totally* collision-resistant, because a collision $H(x) = H(x')$ exists, and the algorithm that merely outputs (x, x') would always find it. Thus, we imagine that the hash function we use is randomly drawn from a larger family, and the "key" represents the individual member of the family. Note that we do not think of the key as secret: indeed, once the representative is chosen, the key will be known to all.

Compression functions must also be defined in terms of families. An (m, k)-bit *compression function family* is a function $f : \mathcal{K}_f \times \{0,1\}^m \times \{0,1\}^k \to \{0,1\}^k$. Again, here, $\mathcal{K}_f$ represents the set of keys for the compression function.

2.2 Ideal Hash Functions and Compression Functions

Typically, an ideal n-bit hash function is thought of as a random function $H : \{0,1\}^* \to \{0,1\}^n$. Here, there is no notion of key; the idea of choosing a random key for the hash function is abstracted away, represented as part of the randomness in the oracle.

An ideal (m, k)-bit compression function, similarly, is a random function $f : \{0,1\}^m \times \{0,1\}^k \to \{0,1\}^k$.

2.3 Ideal Weak Compression Functions

In our construction we do not want to go so far as to assume that the compression functions are random oracles, as this would imply that they are collision resistant, and immune to all forms of attack. Instead, we will model our ideal compression function as a random oracle with additional *attack oracles* that provide results of successful attacks, and yet *still* give answers consistent with a random oracle.

This can be implemented in a variety of ways, depending on what the attack oracle does. We imagine that there is an oracle for the compression function f, so that on a new query (x, y), a random output value z is returned. The following list describes the attack oracles for a variety of compression function security levels.

- **Ideal compression function.** No attack oracle, only the f oracle.
- **Collision-tractable compression function:** On invoking the attack oracle with no input, the oracle returns random values (x, x', y, y', z) such that $f(x, y) = z = f(x', y')$ where $(x, y) \neq (x', y')$.[1]
- **Second preimage-tractable compression function:** On invoking the attack oracle on input (x, y), the oracle returns a random pair of values (x', y') such that $f(x', y') = f(x, y)$.
- **Preimage-tractable compression function:** On invoking the attack oracle on input z, the oracle returns a random pair of values (x, y) such that $f(x, y) = z$.
- **Partially-specified preimage-tractable compression function:** On invoking the attack oracle on input (x, z), the oracle returns a random value y such that $f(x, y) = z$.
- **Two-way partially-specified preimage-tractable compression function:** There are two attack oracles. On querying the first (called f^{-1}) on input (x, z), the oracle returns a random value y such that $f(x, y) = z$. On querying the second (called f^*) on input (y, z), the oracle returns a random value x such that $f(x, y) = z$.

This last form of ideal compression function we will name for convenience a *weak ideal compression function.* It should be clear that we can implement any form of compression function higher on the list with a weak ideal compression function (for instance, to implement the attack oracle for a preimage-tractable compression function, on input z, we pick a random x and query our first attack oracle on (x, z) to obtain y, then return (x, y)).

In fact, this form of weak compression function is susceptible to every form of (black-box) attack we are aware of.[2] An ideal weak compression function cannot

[1] That is, x, x', y, y', and z are generated at random; if known values of f do not prohibit the property $f(x, y) = z = f(x', y')$, then those outputs are given, otherwise new ones are selected until known values of f do not cause a problem. Once the attack oracle returns a query, it affects how f will respond to (x, y) or (x', y').

[2] Of course, we cannot capture non-black-box attacks when we try to view our primitives as ideal.

be used simply in an iterated way to make a hash function. For instance, if the padding function appends padding that depends only on the length of the input, we can find a collision by creating a random m-bit message x, computing $z = f(x, IV)$, and then querying the attack oracle $f^*(IV, z)$ to get a random x' such that $f(x', IV) = z$. Then, since the padding changes x and x' in the same way (because they are the same length), $H(x)$ and $H(x')$ will be the same, as they collide after one block, and the remaining blocks are the same.

Nonetheless, there is cryptographic strength implied in this notion of an ideal weak compression function, because despite the attacks we explicitly allow against it, we still imagine that the results of such attacks will be random and out of the control of the adversary.

Note that we are being quite generous with our attack oracles here. For an actual compression function, there is no guarantee that (for instance) a y such that $f(x, y) = z$ even exists, let alone many such y.[3]

2.4 Ideal Hash Functions and Compression Functions Based on Weak Ideal Compression Functions

Following Coron et al. [4], and paraphrasing closely from their paper, we will use the following methodology to prove that our constructions are sound. Let C be a Turing machine with access to an oracle: C will represent the *construction* and its oracle(s) will represent the ideal primitive the construction is made from.

Let $\mathit{\Gamma}$ represent the oracle(s) for the underlying ideal primitive(s), and let $\mathit{\Delta}$ represent the oracle(s) for the ideal version of the primitive we try to construct with C.

We say that C is (t_A, t_S, q, ϵ)-*indifferentiable* from $\mathit{\Delta}$ if there is a simulator S such that for all distinguishers A,

$$|Pr[A^{C,\mathit{\Gamma}} = 1] - Pr[A^{\mathit{\Delta},S} = 1]| < \epsilon,$$

where (1) S answers as many different types of oracle queries as $\mathit{\Gamma}$ provides, and S has oracle access to $\mathit{\Delta}$ and runs in time at most t_S, and (2) A runs in time at most t_A and makes at most q queries of its various oracles. We say that C is computationally indifferentiable from $\mathit{\Delta}$ if for all security parameters α it holds that C is $(t_A(\alpha), t_S(\alpha), q(\alpha), \epsilon(\alpha))$-indifferentiable from $\mathit{\Delta}$, where t_A and t_S are polynomial in α, where $q(\alpha) \leq t_A(\alpha)$, and where ϵ is negligible in α. We say that C is statistically indifferentiable if for all security parameters α it holds that C is $(t_A(\alpha), t_S(\alpha), q(\alpha), \epsilon(\alpha))$-indifferentiable from $\mathit{\Delta}$, where t_S and q are polynomial in α, and where ϵ is negligible in α.[4]

[3] It may be more reasonable to think of our ideal compression function as a random quasigroup: that is, for every (x, z) there is a unique random y such that $f(x, y) = z$, and similarly, for every (y, z) there is a unique random x. However, we proceed under the more general attack oracle.

[4] In other words, we no longer restrict the running time of the adversary, but we still restrict the number of queries.

3 The Zipper Hash Construction

The zipper hash is a general hash function construction. To build an n-bit hash function, we need two independent (m, k)-bit compression functions f_0 and f_1, as well as a padding function P, an initialization vector IV, and a finalization function $g : \{0,1\}^k \rightarrow \{0,1\}^n$. On input x, P is guaranteed to return a value such that $x||P(x)$ is a string that can be broken down into m-bit blocks, and for all $x \neq x'$, $x||P(x) \neq x'||P(x')$. Given all these pieces, the zipper hash function works as follows:

1. Let $x_1, \ldots x_l$ be m-bit strings such that $x_1|| \ldots ||x_l = x||P(x)$.
2. H_1 is computed as $f_0(x_1, IV)$, and $H_2, \ldots, H_l$ are computed iteratively as $H_i = f_0(x_i, H_{i-1})$.
3. H'_1 is computed as $f_1(x_l, H_l)$, and $H'_2, \ldots, H'_l$ are computed iteratively as $H'_i = f_1(x_{l-i+1}, H'_{i-1})$.
4. Output $H(x) = g(H'_l)$.

This construction is called the zipper hash as its structure is reminscent of a zipper. See figure 2. Note that although we require two independent compression functions, we can implement two independent weak ideal compression functions with a single one; see Appendix A.

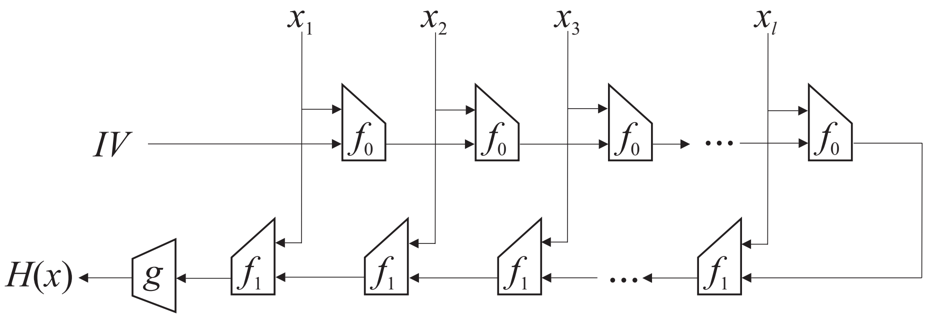


Fig. 2. The zipper hash function

4 Security

Let C be the Turing machine that implements the zipper hash. We will prove the following theorem:

Theorem 1. *C is statistically indifferentiable from an ideal hash function Δ, using two ideal weak compression functions represented by Γ, where g is the identity function.*

This will prove that the zipper hash, with g being the identity function, is indistinguishable from a random oracle. If g is not the identity function, then the overall zipper hash will be indistinguishable from $g \circ \Delta$. If, for instance, g has the property that it produces a random output on a random input, this will also be an ideal hash.

To briefly sketch the proof, the simulator answers oracle queries for the weak ideal compression functions randomly, except when a query is the last one needed to compute the hash function on some value, in which case the simulator assumes that the query was in the forward direction for the last compression function evaluation, and queries Δ and gives this value. It is nontrivial to show that the simulator can always determine when a query amounts to the last one needed to compute the hash function, but with careful record-keeping, we can do it in polynomial time.

We will then make the assumption that no unexpected coincidences occur: that is, for instance, if (u, v) is given as a query to an oracle of Γ, that the randomly generated answer w is not equal to any w that has been involved in a query before, nor is it equal to IV. We describe an event Bad, the event that this assumption fails. We then prove (1) that if Bad never happens, the simulator will simulate Γ perfectly, and (2) that Bad only happens with negligible probability over the course of an attack.

4.1 Record Keeping

In order to simulate Γ (the weak ideal compression functions) with access only to Δ, we use the natural approach: we answer queries to Γ's oracles randomly as long as it follows the constraints: (1) for each (x, y) pair, there is only one value z such that $f_0(x, y) = z$, and only one value z' such that $f_1(x, y) = z'$, and (2) for any l m-bit values $x_1, \ldots, x_l$, f_0 and f_1 have to be such that $\Delta(x_1 || \ldots || x_l) = C(\Gamma)$.

Meeting the first constraint is easy; we simply do the following on each query. When we receive a query $f(x, y)$[5], we check to see if we have defined an answer $z = f(x, y)$; if so, we return z, and if not, we generate a random z and note that $z = f(x, y)$, and return z. When we receive an attack query $f^{-1}(x, z)$, we pick a random y until we find one such that we have not defined an answer $z' = f(x, y)$ for $z \neq z'$, and return that y, and note that $z = f(x, y)$; we do similarly for an attack query on $f^*(y, z)$.

However, the most difficult part of record keeping is that we must be aware of when a query imposes a constraint based on Δ. In order to do this, we will attempt to keep track of all "partial chains." A partial chain is a sequence of x values $x_1, \ldots, x_l$, and two y-values y, y' such that $f_1(x_1, f_1(x_2, \ldots, f_1(x_l, f_0(x_l, \ldots, f_0(x_1, y') \ldots)) \ldots)) = y$. If a partial chain is such that $y' = IV$ then y must be equal to $\Delta(x_1 || \ldots || x_l)$. However, it may be computationally infeasible to keep track of *all* partial chains that arise. Instead, we will keep track of only those that arise in expected ways, and we will prove later that we will actually find all partial chains as long as no unexpected coincidences occur.

[5] In this proof, when we refer to an f query, we mean either an f_0 or f_1 query. We use this convention similarly when referring to f^* or f^{-1} queries.

For ease of notation, when we discover a partial chain, we will make a note of it, which we denote $\mathsf{Chain}(x, y', y)$. Effectively, this note means that if the initialization vector were y', then $H(x)$ would output y.

Forward queries. We show how to keep track of this for one type of query at a time, starting with forward queries. Without loss of generality, we assume that the query is on a new input pair (x, y). If the query is an f_0 query, we will not attempt to find whether any partial chains have been formed. For f_1 queries, we will check if any partial chains have been formed using this query at the end. If so, we check if any of these partial chains are formed starting at IV, and if so, we use Δ to find the value we should set to be $f_1(x, y)$. If not, we pick $f_1(x, y)$ at random. If a query forms two or more distinct partial chains starting at IV, the simulator gives up and halts. If the simulator doesn't halt, it will make notes of all partial chains that have been formed with the current query at the end.

If the query is $f_0(x, y)$ then we can check if this completes a single-block partial chain. If there is a y' such that $f_0(x, y') = y$ then the value we return will form the chain $\mathsf{Chain}(x, y', f_1(x, y))$. If there is an x' and a y'' such that $\mathsf{Chain}(x', y'', y)$ and also there is a y' such that $f_0(x, y') = y''$ then the value we return will form the chain $\mathsf{Chain}(x||x', y', f_1(x, y))$.

Backward queries. Next, we consider "backward" queries, that is, a query $f^{-1}(x, z)$. Similarly to forward queries, if the query is an f_1^{-1} query, we will not attempt to find whether any partial chains have been formed. For f_0^{-1} queries, however, we will check if any partial chains have been formed using this query at the beginning. If so, it may be that a partial chain has been formed starting at IV, but we can do nothing to set the appropriate value to one matching Δ in this case: it is too late, and the simulator will halt. However, this will not happen unless an unexpected coincidence occurs. Thus, once we have found all partial chains that will be formed from the current query, we pick a random answer to it and note the chains that are formed.

The result of a query $f_0^{-1}(x, z)$ will form a single-block chain if it is already known that $f_1(x, z) = y$ for some value y. In this case, we may note $\mathsf{Chain}(x, f_0^{-1}(x, z), y)$. The result of $f_0^{-1}(x, z)$ will form a longer chain if it is already noted that $\mathsf{Chain}(x', z, y')$ for some y', and also $f_1(x, y') = y$ is known for some y, in which case we may note $\mathsf{Chain}(x||x', f_0^{-1}(x, z), y)$.

Squeeze queries. Finally, we consider "squeeze" queries, that is, a query $f^*(y, z)$. Though squeeze queries may form chains, we do not check for them. If a chain is accidentally formed through a squeeze query, the simulator's behavior may become bad later, but this only happes if an unexpected coincidence occurs.

4.2 The Bad Event

We will prove that our simulator fools the adversary by proving that the distribution of the adversary's output in the real system (where S is not involved) is identical to the distribution of the adversary's output in the ideal system, conditioned on a certain "bad" event not happening. The bad event Bad represents the event that a previously-used value is generated as the random answer to a later query. To be precise, let us imagine that (x_i, y_i, z_i) are all the triples of values such that $f_0(x_i, y_i) = z_i$ has been established in a query, and that (x'_i, y'_i, z'_i) are all the triples of values such that $f_1(x'_i, y'_i) = z'_i$ has been previously established. Then Bad occurs on the next query if:

1. The latest query is an $f(x, y)$ query that returns a value z equal to y_i, z_i, y'_i or z'_i for some i, or $z = IV$.
2. The latest query is an $f^{-1}(x, z)$ query that returns a value y equal to y_i, z_i, y'_i, or z'_i for some i, or $y = IV$.
3. The latest query is an $f^*(y, z)$ query that returns some value x equal to x_i or x'_i for some i.

Lemma 1. *If* Bad *does not happen when we simulate, the simulator will not halt during a query.*

Recall that the simulator will only halt in one situation: if a forward f_1 query completes more than one partial chain that start at IV. Specifically, this happens when a forward query $f_1(x, y)$ is such that for some $x' \neq x''$ and for some y'_0 and y'_1, we know $\mathsf{Chain}(x', y'_0, y)$ and $\mathsf{Chain}(x'', y'_1, y)$, and $f_0(x, IV) = y'_0$ and $f_0(x, IV) = y'_1$. Therefore we can conclude that $y'_0 = y'_1$. In order for this to happen, we must have noted both $\mathsf{Chain}(x', y', y)$ and $\mathsf{Chain}(x'', y', y)$ for some $x' \neq x''$.

Remark 1. If we note $\mathsf{Chain}(x, y', y)$ then, when we note it, either y' or y is a newly-generated random query answer. This is clear from our description of S above.

Remark 2. First, we prove that if there is some pair of notes $\mathsf{Chain}(x_0, y', y)$ and $\mathsf{Chain}(x_1, y', y)$ where the first block of x_0 is not the same as the first block of x_1, then Bad must have happened. Assume, without loss of generality, that $\mathsf{Chain}(x_0, y', y)$ was not noted later than $\mathsf{Chain}(x_1, y', y)$. Because of the way we notice chains, we note $\mathsf{Chain}(x_1, y', y)$ only when computing either a forward or backward query with x as the input value, where x is the first block of x_1. Since x is not the first block of x_0, we do not note $\mathsf{Chain}(x_0, y', y)$ at this time, so it must have been noted previously. However, because of remark 1, when we note $\mathsf{Chain}(x_1, y', y)$ either y' or y must be a newly-generated random query answer, so it can only be equal to the previously-known value of y if Bad occurs on this query.

Remark 3. Next, we note that if x_0 and x_1 consist of at least one block, and there is some y such that $\mathsf{Chain}(x||x_0, y', y)$ and $\mathsf{Chain}(x||x_1, y', y)$, where x is a single block, then either (1) there is some w and some w' such that $\mathsf{Chain}(x_0, w', w)$ and $\mathsf{Chain}(x_1, w', w)$ are already known, or (2) Bad has happened.

Assuming that both $\mathsf{Chain}(x||x_0, y', y)$ and $\mathsf{Chain}(x||x_1, y', y)$ were discovered simultaneously (if not, the previous argument shows that Bad happened), there are two cases:

- If both were discovered on a forward query $f_1(x, w)$, it must have been that both $\mathsf{Chain}(x_0, w', w)$ was known, and that $w' = f_0(x, y')$ for some w and w'. Furthermore, it must also be true that $\mathsf{Chain}(x_1, w'', w)$ was known, and that $w'' = f_0(x, y')$. But then, $w'' = f_0(x, y') = w'$, so the first condition holds.
- If both were discovered on a backward query $f_0^{-1}(x, w')$, then it must have been that $\mathsf{Chain}(x_0, w', w)$ was known for some w, and that $f_1(x, w) = y$. We must also have noted $\mathsf{Chain}(x_1, w', w'')$ for some w'' such that $f_1(x, w'') = y$. If $w = w''$ then the first condition holds. If not, then from remark 1, whichever of $f_1(x, w) = y$, $f_1(x, w'') = y$, $\mathsf{Chain}(x_0, w', w)$, or $\mathsf{Chain}(x_1, w', w'')$ was discovered last would have triggered Bad.

Remark 4. If there is some note $\mathsf{Chain}(x, y', y)$ and $\mathsf{Chain}(x||x_1, y', y)$ where x is a single block, then Bad has happened. Again, we may assume that $\mathsf{Chain}(x, y', y)$ and $\mathsf{Chain}(x||x_1, y', y)$ were discovered simultaneously. There are two cases:

- If both were discovered on a forward query $f_1(x, y'')$, then it must have been known in advance be that $f_0(x, y') = y''$, and that $\mathsf{Chain}(x_1, y'', y'')$. However, $\mathsf{Chain}(x_1, y'', y'')$ is impossible unless Bad happens, in view of remark 1.
- Similarly, if both were discovered on a backward query $f_0^{-1}(x, z)$, then it must have been known in advance that $f_1(x, z) = y$ and that $\mathsf{Chain}(x_1, z, z)$, which again guarantees that Bad has happened.

By remarks 2, 3, and 4, if $\mathsf{Chain}(x, y', y)$ and $\mathsf{Chain}(x', y', y)$ are known for $x \neq x'$ then Bad must have happened: if x is not a prefix of x' of x' or vice versa, we can descend by remark 2, getting similar properties, until the first blocks of x and x' are unequal. If x is a prefix of x' or vice versa, we can descend by remark 2 until we fall in to the case covered by remark 3. Therefore, the simulator will never halt prematurely unless Bad has happened.

Lemma 2. *If a query is ever made to S that would complete a partial chain, we note it unless* Bad *happens.*

Suppose a query is made to S that would complete a partial chain. There are three cases to consider:

Case i: A partial chain is completed on a forward query. If the link determined by $f(x, y)$ is used anywhere other than at the end, it can only be used there if the value generated for $f(x, y)$ triggers the Bad event. If the link determined by $f(x, y)$ only completes chains by adding on to the end, it must be a query to f_1, and then there are two cases: either the partial chain is one block long, which we explicitly check for, or the partial chain is longer, in which case, a shorter, compatible partial chain is already known. In either case, we note the newly completed partial chain.

Case ii: A partial chain is completed on a backward query $f^{-1}(x, z)$. Similarly, if the result of this query is used anywhere other than at the beginning, it can only be used there if the result triggers the Bad event. Again, if the result can be used at the beginning, it must be a f_0^{-1} query, and our algorithm for the simulator is correct.

Case iii: A partial chain is completed on a "squeeze" query $f^*(y, z)$. In this case, the chain could only be completed if something is already known about f_0 or f_1 on input x where x is the result of this query. If this were the case, the result of this query would trigger the Bad event.

Lemma 3. *If a query is ever made to S that would complete a partial chain starting at IV, we note it, and respond correctly, unless* Bad *happens.*

The proof of this lemma is very similar to the proof of lemma 3. Note that by lemma 3, if a query is made to S that completes *any* partial chain, and Bad has not happened, we note it. Therefore, we need only consider two cases:

Case i: The partial chain $\mathsf{Chain}(x, IV, y)$ is noted on a forward query to f_1. In this case, we obtain y by querying $\Delta(x)$, so our answer is correct.

Case ii: The partial chain $\mathsf{Chain}(x, IV, y)$ is noted on a backward query to f_0^{-1}. If this is the case, Bad must have happened, because this can only happen if the result of the final f_0^{-1} query was IV.

Lemma 4. *The probability that* Bad *happens is negligible.*

Note that initially, before any queries are made, Bad has not happened. If Bad has not happened after the first q queries, then the probability that it happens on the $q + 1$st query is at most $(2q + 1) \cdot max(2^{-m}, 2^{-k})$. This is because there are at most $(2q + 1)$ answers (all the previous y and z values, plus IV) that would make Bad happen, out of 2^m or 2^k possible random answers, depending on the type of query. Therefore, if the adversary makes a total of q queries, the probability that Bad happens is at most $\Omega(q^2 2^{-r})$, where $r = min(m, k)$.

We note that the running time of S is polynomial in the number of queries, but is independent from the running time of the adversary. This completes the proof of Theorem 1.

4.3 Security Against Standard Attacks

In this section we discuss the applicability of our security proof to the standard attacks against hash functions. What we have proven, essentially, is that an adversary with a limited number of queries cannot distinguish between the zipper hash implemented with weak ideal compression function and a random oracle. Specifically, if the number of queries the adversary can make is significantly less than $2^{min(m,k)/2}$, the Bad event remains extremely unlikely, and the proof is successful.

Provided the adversary makes fewer than this many queries, the only attacks an adversary could succeed in are attacks that could be performed against an ideal hash function. Hence, so long as this query limit is respected, the adversary should not be able to find collisions, preimages, second preimages, et cetera. However, our proof does not imply that the adversary cannot perform these attacks more efficiently on the zipper hash than on an ideal hash function if the adversary exceeds this query limit. For instance, to find a preimage of an ideal hash function takes $O(2^k)$ queries, where k is the output size, whereas we cannot guarantee security against that many queries. As another example, our construction does not provide security against multicollisions: in fact, it fits a known framework in which an extension of the Joux attack is possible [15,9].

Nonetheless, keep in mind that the queries the adversary is allowed to make in attacking the zipper hash include attack queries, which are modeled as if they are trivial, but may in fact require significant effort.

5 Zipper Hash-Based Compression Function

The most natural criticism of the zipper hash in practice is that it is no longer streamable, as iterated hash functions are. However, we can easily use the zipper hash construction to create an ideal compression function rather than a full ideal hash function, which will allow us to use one of the modified iterated constructions of Coron et al. [4] and create a streamable, ideal hash function from weak ideal compression functions.

Now that we have proven that the zipper hash is indifferentiable from a random oracle, if we assume that we have an (m, m)-bit underlying compression function, we can make an (m, m)-bit ideal compression function very simply: let $f(x, y) = H(x||y)$. That is, we use a full zipper hash computation on the two-block message $x||y$ as our compression function. By a simple restriction on our theorem, this is indifferentiable from a random oracle from $\{0,1\}^m \times \{0,1\}^m \rightarrow \{0,1\}^m$, and is therefore an ideal compression function.

5.1 Amortizing Streamability vs. Efficiency

The full zipper hash requires 2 underlying compression function queries per input block. If we use the zipper hash in the natural way as a compression function, the resulting iterated construction (after double-piping) requires 4 underlying compression function queries per input block. However, we can trade streamability for efficiency here, by using the zipper hash function on more blocks of input at once.

For instance, we can make a $(3m, m)$-bit compression function by computing $f(x_1||x_2||x_3, y) = H(x_1||x_2||x_3||y)$. This requires 8 queries for 3 input blocks, which is a significant savings compared to 12 queries for 3 input blocks. However, by having the compression function require more input, we are sacrificing streamability: we must now buffer 3 input blocks instead of one before we can apply the compression function. In general, if we use b blocks at once, our efficiency will be $\frac{2(b+1)}{b} = 2 + \frac{2}{b}$ queries per input block.

6 Efficiency

The zipper hash requires $2l$ compression function evaluations on an input of l blocks. There is one additional drawback in that the zipper hash is not *streamable*: we have to scan the message twice, so in principle, we cannot compute the zipper hash in fixed memory unless for some reason it is feasible to access the input a second time. This is an especially significant point as it is often desired that limited devices such as smart cards be able to compute hash functions with limited available memory. However, there are some points in favor of this approach anyway:

- In applications on non-limited devices, streamability is not mission-critical. It may be worthwhile to consider a non-streamable hash function like the zipper hash if it has attractive theoretical properties.
- The zipper hash can be implemented using existing machinery: essentially all that is required is two traditional Merkle-Damgård hash function evaluations.
- Theoretically, it is possible that weakly secure compression functions could be designed that may be more efficient than strong ones. If this is the case, then such compression functions used in the zipper hash constructions may actually yield a *more* efficient hash function.
- The zipper hash may remain secure even if the compression function is vulnerable to attack.
- Finally, *any* streamable hash function is essentially an iterated hash function based on a compression function. Therefore, some black-box attacks are known that apply to any streamable hash function. If such attacks are undesirable, it may be necessary to adopt a non-streamable approach.

This last point needs some explanation. We prove that all streamable hash functions (that is, all hash functions with constant-size state) are in fact iterated hash functions in Appendix B. The gist of the argument is that if a hash function can be streamed, this means it can be computed with a fixed amount of state, with a fixed maximum amount of memory into which the input is provided. Whatever method is used to combine the input with the current state to arrive at the next state can be thought of as the compression function. The initial state can be thought of as the initialization vector, and whatever method is used, once all input has been processed, to determine the output can be thought of as the finalization function.

7 Conclusion

This paper is new in two ways. First of all, this is the first paper that we are aware of to foray into positive results for non-streamable hash function design. Second, as far as we know, this is the first paper to explicitly model weakly-secure primitives as ideal primitives with relevant attack oracles available. Here are some open problems we consider worth investigating:

– Are there attacks against the generic zipper hash design that are better than brute force?
– What other non-streamable hash function designs are possible, and what properties do they have? In particular, are there benefits to making three or more passes in a zipper-like construction?
– Is there a weaker version of an ideal compression function? If so, can we use it to build secure hash functions?
– Can this notion of a weak ideal primitive be used elsewhere?
– Can we make better constructions, or prove stronger security results, by representing our compression functions as ideal random quasigroups?

Acknowledgements

We would like to sincerely thank Bart Preneel [17] and Stefan Lucks [13] for their Asiacrypt 2005 presentations which inspired this research. We would also like to thank those with whom we had useful conversations concerning this project: Ron Rivest, Stefan Lucks, Stuart Haber, Zulfikar Ramzan, Silvio Micali, Susan Hohenberger, and Barbara Liskov. We would also like to thank the program committee for their comments.

References

1. Biham, E., Chen, R., Joux, A., Carribault, P., Lemuet, C., Jalby, W.: Collisions of SHA-0 and reduced SHA-1. In: Cramer [5], pp. 36–57
2. Black, J., Rogaway, P., Shrimpton, T.: Black-box analysis of the block-cipher-based hash-function constructions from PGV. In: Yung, M. (ed.) CRYPTO 2002. LNCS, vol. 2442, pp. 320–335. Springer, Heidelberg (2002)
3. Brassard, G. (ed.): CRYPTO 1989. LNCS, vol. 435. Springer, Heidelberg (1990)
4. Coron, J., Dodis, Y., Malinaud, C., Punyia, P.: Merkle-Damgård revisited:how to construct a hash function. In: Shoup [18], pp. 430–448
5. Cramer, R. (ed.): EUROCRYPT 2005. LNCS, vol. 3494, pp. 22–26. Springer, Heidelberg (2005)
6. Damgård, I.: A design principle for hash functions. In: Brassard [3], pp. 416–427
7. Dean, R.: Formal aspects of mobile code security. Ph.D. Dissertation, Princeton University (1999)
8. Franklin, M. (ed.): CRYPTO 2004. LNCS, vol. 3152. Springer, Heidelberg (2004)
9. Hoch, J., Shamir, A.: Breaking the ICE - finding multicollisions in iterated concatenated and expanded (ICE) hash functions. In: Robshaw, M. (ed.) FSE 2006. LNCS, vol. 4047, Springer, Heidelberg (2006)
10. Joux, A.: Multicollisions in iterated hash functions, application to cascaded constructions. In: Franklin [8], pp. 306–316
11. Kelsey, J., Kohno, T.: Herding hash functions and the Nostradamus attack. Available on eprint: article 2005/281 (2005)

12. Kelsey, J., Schneier, B.: Second preimages on n-bit hash functions for much less than 2^n work. In: Cramer [5], pp. 474–490
13. Lucks, S.: A failure-friendly design principle for hash functions. In: Roy, B. (ed.) ASIACRYPT 2005. LNCS, vol. 3788, pp. 474–494. Springer, Heidelberg (2005)
14. Merkle, R.C.: A certified digital signature. In: Brassard [3], pp. 218–238
15. Nandi, M., Stinson, D.R.: Multicollision attacks on a class of hash functions. Available on IACR eprint archive, paper 2006-2055 (2006)
16. Preneel, B.: Analysis and design of cryptographic hash functions. Ph. D. thesis, updated version (2003)
17. Preneel, B.: Hash functions: past, present and future. In: Roy, B. (ed.) ASIACRYPT 2005. LNCS, vol. 3788, Springer, Heidelberg (2005)
18. Shoup, V. (ed.): CRYPTO 2005. LNCS, vol. 3621. Springer, Heidelberg (2005)
19. Wang, X., Lai, X., Feng, D., Chen, H., Yu, X.: Cryptanalysis of the hash functions MD4 and RIPEMD. In: Cramer [5], pp. 1–18
20. Wang, X., Yin, Y.L., Yu, H.: Finding collisions in the full SHA-1. In: Shoup [18], pp. 17–36
21. Wang, X., Yu, H.: How to break MD5 and other hash functions. In: Cramer [5], pp. 19–35
22. Wang, X., Yu, H., Yin, Y.L.: Efficient collision search attacks on SHA-0. In: Shoup [18], pp. 1–16

A Simulating Two Compression Functions

In the zipper hash construction, we require two independent compression functions. However, in practice, it is more sensible to use only one. If we have a single $(m+1,k)$-bit compression function f, we can define $f_0(x,y) = f(0||x,y)$ and $f_1(x,y) = f(1||x,y)$. This idea is natural, but we must prove that it is secure in an ideal sense.

Theorem 2. *If C is the TM that computes f_0 and f_1 given the oracle Γ for a weak ideal compression function f, then C is (perfectly) indifferentiable from Δ, where Δ is the oracle for two independent random weak ideal compression functions.*

Proof. First of all, we must describe how C answers attack queries. If C is given an attack query $f_b^{-1}(x,z)$, C makes an attack query $f^{-1}(b||x,z)$ to obtain the answer y. If, on the other hand, C is given an attack query $f_b^{-1}(y,z)$, C makes an attack query $f^{-1}(y,z)$ to obtain an answer $b'||x$ where b' is the first bit. If $b' = b$, C returns x; if not, C makes an identical attack query, and repeats until it obtains an answer that does begin with b.

To simulate the oracle for the single weak ideal compression function is very easy. To answer a forward query $f(b||x,y)$, we simply return the result of the forward query $f_b(x,y)$. To answer an attack query $f^{-1}(b||x,z)$, we return the result of the attack query $f_b^{-1}(x,z)$. Finally, to answer an attack query $f^{-1}(y,z)$, we pick a random bit b and return the result of $f_b^{-1}(y,z)$.

The only effective difference between the simulation and the construction is that the construction may have made superfluous queries to $f^{-1}(y,z)$. However, for all adversaries, the queries of the adversary are independent of the superfluous queries, and even if the adversary makes a query that is later restricted by one of those superfluous queries, the result is still random and distributed properly.

Another application of this is to show how to make the Lucks double-pipe construction ideally secure.

Lucks' construction of a double-pipe compression function can be summed up as follows: if f is a $(2m, n)$-bit compression function, then
$f'(x, y_1||y_2) = f(x, y_1||y_2)||f(x, y_2||y_1)$ is a $(2m, 2n)$-bit compression function.

It is easy to see that this construction is not ideal: $f'(x, y||y) = z||z$ for some z, whereas this is unlikely to be the case for an ideal compression function f'. This flaw is easily avoided, however, by the following modification. Assume that f_0 and f_1 are two *independent* $(2m, n)$-bit compression functions, and let

$$f'(x, y_1||y_2) = f_0(x, y_1||y_2)||f_1(x, y_2||y_1).$$

Theorem 3. *f' is computationally indifferentiable from an ideal compression function.*

Proof. It is easy to see how to simulate the f_0 and f_1 random oracles in the presence of a single random oracle for f': to calculate $f_0(x, y)$, for instance, we query $f'(x, y)$, and split the result into two halves, the first of which is $f_0(x, y)$, and the second of which is recorded as $f_1(x, y^{fl})$ where y^{fl} flips the first and second halves of y.

Naturally, we can apply Theorem 2 to show that f_0 and f_1 can be simulated with a single compression function. We do not even need the full strength of Theorem 2, since in this case the compression functions are meant to be ideal, rather than weak ideal.

B Universality of Merkle-Damgård

In this section, we prove that any streamable hash function can be viewed as an iterated hash function with a single compression function f, a fixed initialization vector IV, and a finalization function g.

In order to consider a hash function "streamable," it must be the case that H can be computed by an algorithm M in such a way that (1) M has a fixed-size state, and (2) M receives its input in pieces, each piece being no larger than some maximum size m. If (1) does not hold, then potentially, M simply stores the entire input and computes the hash function only at the end. Furthermore, we assume that the function operates directly on the input message, that is, we assume that any padding is computed as part of the hash function. If this is the case, we can view H as an iterated hash function as follows.

Let k be such that any state of M can be written as a k-bit string. Let IV be the k-bit string corresponding to the starting state of M, appended with the empty string ϵ.

Let f be the $(m, k+m+1)$-bit compression function that works as follows. On input (x, y), let the state s be the state corresponding to the first k bits of y, and let x' be the string encoded in the remaining $m+1$ bits. Based on s, we determine the amount $m' \leq m$ of input that M expects in state s. Let s' be the state after M is run in state s with input the first m' bits of $x'||x$. If $|x'||x|$ is large enough that it contains the entire next block to process, we repeat this, processing another chunk of the input. If not, we let s_{out} be the state M ends in, and let x_{out} be whatever part of $x'||x$ has not been used, and output the $k+m+1$-bit string representation $\langle s_{out}, x_{out} \rangle$.

Let g be as follows. On input x, let s be the state corresponding to the first k bits of x, and let x' be the string encoded in the remaining bits. Run M, from state s, on input x' with the signal that no more input remains, and output the output of M.

It should be clear that the iterated hash function with compression function f, finalization function g, and initialization vector IV simply runs M on the input to the hash function. This shows that the streamable hash function H is in fact an iterated hash function.

B.1 Universal Vulnerability of Iterated Hash Functions

Since all hash functions that can reasonably be thought of as streamable are in fact iterated hash functions, black-box attacks against iterated hash functions are actually universal attacks against streamable hash functions. For instance, the second collision attack described in the introduction applies to *all* streamable hash functions. Also, the Joux attack applies to all streamable hash functions. The efficiency of the Joux attack, as well as its extensions, depends on the size of the internal state (k+m+1, with the construction above); if this size is large enough, the Joux attack is irrelevant, as Lucks has shown. However, note that a streamable hash function may be realizable as an iterated hash function more efficiently in terms of internal state size than the general construction above.

Provably Good Codes for Hash Function Design

Charanjit S. Jutla[1] and Anindya C. Patthak[2,⋆]

[1] IBM Thomas J. Watson Research Center
csjutla@watson.ibm.com
[2] University of Texas at Austin
anindya@cs.utexas.edu

Abstract. We develop a new technique to lower bound the minimum distance of quasi-cyclic codes with large dimension by reducing the problem to lower bounding the minimum distance of a few significantly smaller dimensional codes. Using this technique, we prove that a code which is similar to the SHA-1 message expansion code has minimum distance at least 82, and that too in just the last 64 of the 80 expanded words. Further the minimum weight in the last 60 words (last 48 words) is at least 75 (52 respectively). We expect our technique to be helpful in designing future practical collision-resistant hash functions. We also use the technique to find the minimum weight of the SHA-1 code (25 in the last 60 words), which was an open problem.

Keywords: linear codes, minimum distance, collision-resistant hash functions, SHA-1.

1 Introduction

Recall the SHA-1 message expansion code which is a binary linear code of dimension 512: the 512 information bits are packed into 16 32-bit words $\langle W_0, \cdots, W_{15}\rangle$, and 64 additional words are generated by the recurrence:

$$W_i = (W_{i-3} \oplus W_{i-8} \oplus W_{i-14} \oplus W_{i-16}) <<< 1 \quad \text{for } i = 16, \cdots, 79 \qquad (1)$$

The 80 words $\langle W_0, \cdots, W_{79}\rangle$ can be seen as constituting a code-word in a linear code over $\mathbb{F}_2$ with the above parity check equations. Unfortunately, this code has a minimum distance or weight of no more than 44. Further, the weight restricted to the last 60 words is only 25. This has been exploited in [21] to give a differential attack on SHA-1 with complexity 2^{69} hash operations. Recently, the complexity has further been improved to 2^{63} hash operations [19].

The code for SHA-0, which is same as (1) but without the rotation (see [14], has an even worse minimum weight. The small minimum weight of these codes is an integral part of the attack strategies on these hash functions (see [22,23,3,2,1,20,21]). The question naturally arises as to why codes with better

⋆ This work was done while the author was visiting IBM T.J. Watson Research Center, N.Y.

© Springer-Verlag Berlin Heidelberg 2007

minimum weight were not employed, even though the coding theory literature [17] is rife with codes with proven good minimum weight. However, as we point out later in section 2, none of them comes close to being as efficient to implement in software (i.e., do not have an efficient encoder) as the code (1) above. One is then led to ask if codes more complex than (1), but still easy to implement, could be shown to have a better minimum distance. Surprisingly, it was not even known how to lower bound the minimum weight of the above SHA-1 code, even though it is related to codes such as Hadamard code [17] (we address this relationship in section 2).

The purpose of this paper is three-fold. First, to introduce a novel technique for lower bounding efficient-to-implement codes such as given by (1). Second, to use this technique to lower bound this particular code (which was an open problem). Third, to show how one can design efficient-to-implement codes with a much better minimum distance, and to actually give such a code. We expect our technique to be helpful in designing future practical collision-resistant hash functions.

Before we describe our technique, we mention the specific code we analyze, as this specific example will help in understanding the complexity of the problem and the intricacy of the technique. The code, $\mathcal{C}$, we consider is a 80×32 length binary code of dimension 16×32, given by the following recurrence relation (or parity check equations): Let $V_i \overset{def}{=} W_{i-3} \oplus W_{i-8} \oplus W_{i-14} \oplus W_{i-16}$.

$$W_i \overset{def}{=} \begin{cases} V_i \oplus ((W_{i-1} \oplus W_{i-2} \oplus W_{i-15}) \lll 1) & \text{if } 16 \le i < 36 \\ V_i \oplus ((W_{i-1} \oplus W_{i-2} \oplus W_{i-15} \oplus W_{i-20}) \lll 1) & \text{if } 36 \le i \le 79 \end{cases} \tag{2}$$

We will show that this code has minimum distance 82, and that too in just the last 64 words (contrast this with SHA-1 which has minimum weight at most 30 [21], and 192 for the highly inefficient Reed Solomon code described in Section 2). Of course, since the dimension of this code is 16×32, a brute force search of $2^{16\times 32}$ is infeasible. Further, it is known that computing minimum weight of an arbitrary linear code is NP-hard (see [18]), and that approximating within a constant factor is NP-hard under randomized reduction (see [5]).

Still, there is additional structure in the above codes (i.e. (1) and (2)), and we intend to exploit that. Note that a codeword of the above codes can be seen as an 80×32 matrix, with each column representing the codeword projected on a particular bit position. Further, the above codes have the property that they are closed under column rotations. Such codes are called *quasi-cyclic codes* in the literature, and have been studied extensively (see [13,4,8,9]). As for estimating the minimum weight of such codes by algorithmic means, the presently known techniques are computationally infeasible [4,8].

Our novel technique reduces the problem of lower bounding the minimum weight of a $k \times n$ dimensional quasi-cyclic code to a function of the minimum weight of a few $k \times n'$ dimensional codes, where n' is much smaller than n. We now briefly explain the main idea of our technique, using the above example code given by (2). For any codeword represented as a 80×32 matrix, note that either (a) there are no all-zero columns in the codeword, in which case we would like to show that on average there are a few (three, e.g.) non-zero bits in each column, or (b) there is a zero column in the codeword, in which case we would like to show that the code projected on a few columns (say, $m << n$) has a large minimum distance.

Unfortunately, there are two major hurdles in this plan (related to case (b)). Consider the first non-zero column next to a zero column (either to the left or the right). It turns out that the code projected on that column is not expected to be any better than the code for SHA-0, and hence we do not expect a minimum weight of more than 15-20 for that column. Thus, we would need m to be about five to get a minimum weight of 75, in which case the dimension of the projected code is still too large, i.e. 16×5. Further, there are *pathological cases* (and which cannot be avoided) where the code projected on a column yields a minimum weight as low as 1. Thus, we may be forced to consider m to be much larger than five. The novelty of our approach lies in tackling these two major hurdles. We show that the minimum weight of the sub-code in case (b) can be lower bounded by a function of the minimum weight of a few codes (some of which are subspaces), each of dimension at most 16×3. A "lazy" brute force search with early-stopping then yields a lower bound of 82.

Other Contributions: We also use the techniques developed to give a lower bound of 25 (in the last 60 words) on the minimum weight of the codewords of SHA-1 (this was an open problem). A codeword of weight smaller than 25, could potentially lead to an even more drastic attack on SHA-1. As for further advancement of our techniques, we also prove that the minimum weight of our example code (2) is at least 75 (52) when restricted to the last 60 (48 resp.) words. We will give detailed proof of this in the full version of the paper. Note that front truncations are not equivalent to back truncations for this code.

Organization: The rest of the paper is organized as follows: In section 2 we review limitations of known algebraic techniques. In section 3 we give an informal description of the proof technique and the intuition behind why certain codes are easier to analyze. In section 4 we give a detailed proof of a lower bound on the code given by (2). In the Conclusion section, we describe applications of our methods to designing hash functions. In Appendix A, we give detailed algebraic proofs of some lemmas in section 4.

2 Limitations of Purely Algebraic Techniques

We first investigate the SHA-0 code restricted to a single column, which is a length 80 binary code of dimension 16, given by the binary parity check equations:

$$a_i = a_{i-3} \oplus a_{i-8} \oplus a_{i-14} \oplus a_{i-16} \quad \text{for } i \;=\; 16, \cdots, 79 \tag{3}$$

Consider the polynomial $h(X) = X^{16} + X^{13} + X^8 + X^2 + 1$, which is known to be a primitive polynomial, as the smallest n such that $h(X)$ divides $X^n - 1$ is $2^{16}-1$. Hence, if the above code was extended up to length $2^{16}-1$, it would be the code generated by the LFSR (Linear Feedback Shift Register) given by primitive polynomial $h(X)$. However, such codes are well known [24,7] to be a subcode of first-order Reed Muller codes (also known as Hadamard codes) with one digit dropped. Such codes have an extremely good minimum distance of $2^{15} - 1$, or fractional distance 1/2. Unfortunately, nothing useful can be said about this code truncated to just the first 80 bits, based purely on known algebraic methods. In fact, any such code (i.e. using any degree 16 primitive polynomial) has a minimum weight of at most 26, i.e. a fractional distance of less than 1/3 (as can be checked by a computer).

The lack of purely algebraic techniques to lower bound even this single column code emphasizes the difficulty of analyzing the more complex codes such as SHA-1 and that given by Equation (2). Of course, if $h(X)$ above was not primitive, and divided $X^{80} - 1$, then we would get a cyclic code of length 80. Such codes can be analyzed much more easily, and it is not too difficult to see that the best cyclic code gives a minimum distance of only 8. However, there are non-cyclic linear codes known of minimum distance 31, though they are really difficult to encode. One could also consider cyclic codes of length 85, which have a much better minimum distance and then truncate them. However, the analysis does not extend to codes which do column mixing like SHA-1.

Instead of quasi-cyclic codes as SHA-1 or Equation (2), one could consider cyclic codes of length 80×32, or of an appropriate length. First note that a random code will give minimum distance roughly 475 for a code with rate 1/4 and length 64×32 (follows from the Gilbert-Varshamov bound). Of course, finding such a code is infeasible. Alternatively, one can try a Reed Solomon code over $\mathbb{F}_{2^8}$ of length $2^8 - 1$ (bytes), and dimension 64 (bytes). Such a code has distance $256 - 64 = 192$ (over bytes). However, the encoder for this code requires multiplication by various elements in $\mathbb{F}_{2^8}$, and is not at all suitable for software implementations. A binary cyclic code of dimension 16×32 would also be extremely cumbersome to implement. Similar considerations rule out known good quasi-cyclic codes.

3 Intuition Behind the Code

Let us start by examining why the message expansion code in SHA-1 given by Equation (1) is not satisfactory (observed independently in [11] and [10]). We can rewrite Equation (1) as follows:

$$\forall i, 0 \leq i \leq 63, \quad W_i = W_{i+2} \oplus W_{i+8} \oplus W_{i+13} \oplus (W_{i+16} >>> 1), \tag{4}$$

where "$>>> 1$" denotes a one bit rotation to the right. The above clearly shows that a difference created in the last 16 words propagates to only up to 4 different bit positions.

One way to remedy this situation is to let $W_i = (W_{i+2} >>> 1) \oplus W_{i+8} \oplus W_{i+13} \oplus (W_{i+16} >>> 1)$. Now Equation (1) becomes $W_i = (W_{i-3} \oplus W_{i-8} \oplus W_{i-16}) <<< 1 \oplus W_{i-14}$. Thus, whether you consider the evaluation in the *forward direction* or in the *reverse direction*, the spread of differences to the neighboring columns (i.e. neighboring bits) is more frequent. However, it is not enough to just have a good intuition about the code, but one also needs to prove a good lower bound on the minimum weight of such codes.

The strategy we use to prove lower bounds on such codes is to divide the proof into two main cases. We argue that either there are no zero columns in a codeword (a column in the codeword is the codeword projected on a particular bit position) or starting from an all zero column, the first neighboring non-zero column is actually a codeword in a good code, and so on.

Elaborating on the first case, i.e., when there are no zero columns, if every column has at least 3 bits ON, we are done. So, assume that there is some column which has 1 or 2 bits ON. Thus, there are $(64 \times 63)/2 + 64$ choices for picking these bits in the column. Having picked these bits, the neighboring column is completely specified by at most 16 bits in that column. Now the two columns together either have weight 6, in which case we are maintaining an average of 3 per column, or the weight of these two columns is at most 5. Thus, our search is quite restricted. We continue in this fashion, noting that the code has to be designed carefully so as to satisfy a property as in Claim 3.

As for the second case, we consider a contiguous band of zero columns, bordered on both sides with non-zero columns (we prove that they cannot be same; in fact we prove by a rank argument that there must be at least four consecutive non-zero columns). We have to assure that when a column is zero, and the neighboring column is non-zero (whether to the right or left), the resulting code for the neighboring column is a good code, i.e., with a good minimum weight. Note that this is important since we may possibly have at most 5-6 non-zero columns. Therefore it is desired that the disturbance propagates fast across columns. Unfortunately, this is impossible for the codes we are considering so far.

Consider a SHA-1 like code, with dimension 16×32, and which is invariant under column rotations. Moreover, suppose that the code is of the form

$$W_i = \sum_{t=1}^{16} a_t W_{i-t} + \left(\left(\sum_{t=1}^{16} b_t W_{i-t} \right) <<< 1 \right), \qquad (5)$$

where $a_1, \cdots, a_{16}, b_1, \cdots, b_{16}$ are boolean. If a_{16} and b_{16} are equal, then there is a codeword which is zero everywhere, except for W_0 which is the all one 32-bit word. Thus for the sake of the argument, assume that $b_{16} = 0$ and $a_{16} = 1$. However in this case, suppose $t' < 16$ is the largest t such that $b_{t'}$ is non-zero. First note that if a column, say C^j, is zero, then in the column to its right, say C^{j-1}, C_k^{j-1} (for $k = 0$ to $15 - t'$) can take any value (i.e., are free variables), and the rest of the column C^{j-1} can be all zero. Further, the propagation to columns C^{j-2}, C^{j-3} etc. can be rather weak.

A similar situation arises when the code is evaluated in the backward direction. The trick is to keep the above free variables few in number, so that the subspace of such *pathological cases* is of a relatively small dimension. This small dimension is absolutely necessary to keep the exhaustive search over this space tractable. One way to get rid of these pathological free variables is to include a term like W_{i-20}, as we do in our code. This in fact gets rid of all the pathological variables in the forward direction and thereby yields a fast expansion. In the backward direction at least one pathological free variable per column remains, and we must search over such subspaces.

4 A Lower Bound on the Minimum Distance

In this section we will prove a lower bound on the code described in the introduction. As mentioned earlier, this is a general technique for reducing the problem to smaller dimensional codes. However, if the reduction is to codes with dimensions too large, then a brute force search may not be feasible. On the other hand, if the reduction is to codes which have really low minimum weight, then we will not obtain a good bound.

We will see in Claim 7 and Claim 8 (in Appendix A) that if the polynomials describing the parity check equations (5) have a certain algebraic property, namely that the polynomial corresponding to coefficients a_t is irreducible, and does not divide the polynomial corresponding to coefficients b_t, then some key reduced codes have low dimensions. Although, these are not necessary conditions, they make a good choice. Similarly, if the coefficients b_1 and b_{15} are both one, then the number of pathological variables per column is small.

We will prove a lower bound on the minimum weight of the code given by Equation (2), but projected on the last 64 words. Clearly, the same bound holds for the full 80 words. The reason we focus on the last 64 words is because the recent attacks on hash functions have shown that the weight of the code in early words (the information words, and a few following words) is mostly immaterial (see "message modification technique" in [21]), and hence the weight in the latter words decides the complexity of the attack.

Since we will be arguing about the weight of this code in the last 64 words, we instead consider the following code $\mathcal{C}64$: Let $W_0, \cdots, W_{15}$ be the message blocks. Let $V_i \overset{def}{=} W_{i-3} \oplus W_{i-8} \oplus W_{i-14} \oplus W_{i-16}$.

$$W_i \overset{def}{=} \begin{cases} V_i \oplus ((W_{i-1} \oplus W_{i-2} \oplus W_{i-15}) <<< 1) & \text{if } 16 \leq i < 20 \\ V_i \oplus ((W_{i-1} \oplus W_{i-2} \oplus W_{i-15} \oplus W_{i-20}) <<< 1) & \text{if } 20 \leq i \leq 63 \end{cases} \tag{6}$$

We first prove that this is indeed sufficient.

Lemma 1. *If the code $\mathcal{C}64$ described above has minimum weight at least w, then $\mathcal{C}$ has minimum weight at least w in its last 64 words.*

Proof. Consider any nonzero codeword in $\mathcal{C}$, say $U = \langle U_0, \cdots, U_{79}\rangle$. Denote $X = \langle U_0, \cdots, U_{15}\rangle$ and $Y = \langle U_{16} \cdots, U_{31}\rangle$ and $Z = \langle U_{32} \cdots, U_{79}\rangle$. Therefore $U = \langle X, Y, Z\rangle$. From Equation 2 observe that the code $\mathcal{C}$ is completely determined by specifying any consecutive 16 word block provided the block starts anywhere in 0 to 20, since the rest can then be obtained by solving the recurrence relation. We therefore choose to specify $Y = \langle U_{16}, \cdots, U_{31}\rangle$, that is we treat Y as the message symbols. Note that a fixed choice of Y also fixes X and Z. Following this observation it is now clear that $\langle Y, Z\rangle$ is a codeword in $\mathcal{C}64$.

Assume that the minimum weight of $\mathcal{C}64$ is d. Then we need to show that any non-zero codeword in $\mathcal{C}$ has weight at least d in its last 64 words. This follows provided a non-zero X implies a non-zero Y. However, if Y is zero then X is zero, as X is a linear function of Y.

Therefore the minimum weight of $\mathcal{C}64$ is exactly the minimum weight of code $\mathcal{C}$ in its last 64 words. ■

Next we prove a lower bound on the minimum distance of $\mathcal{C}64$. We break down the proof into several sub-cases. In each sub-case, we argue often following an exhaustive search over a small space that the minimum weight of the code is at least 82. We mention that a naive algorithm may require to search a space as large as $2^{32\times 16}$ which is clearly not feasible. Therefore the novelty in our approach lies in a careful sub-division of the problem into a small number of tractable cases.

Theorem 2. *The code $\mathcal{C}64$ as defined by Equation 6 has minimum distance at least 82.*

Proof. It is easy to see that the code $\mathcal{C}64$ is a quasi-cyclic code by noting that it is invariant under a 64 bit cyclic shift. From now onwards, we view the codewords of $\mathcal{C}64$ as a matrix that has 32 columns where each column is 64-bit long. The quasi-cyclic property then just mean that the code is invariant under column rotations. Unless otherwise specified, *the arithmetic in the superscript will be modulo 32.*

Now consider any non-zero codeword. Since the code is linear, it suffices to prove that it has weight at least 82. We break down the proof into two main cases depending upon whether or not a codeword has zero columns.

1. (**All Columns Non-Zero Case:**) Consider any such codeword. Also, consider any non-zero column, w.l.o.g., let it be C^0. Denote the columns, to the left of it by $C^1, C^2, \cdots, C^{31}$. Note that all C^i's are non-zero. In this case the following claim holds.

   Claim 3. *For any non-zero column C^j, there exists $k, 0 \leq k \leq 7$ such that the combined weight of columns $C^j, C^{j+1}, \cdots, C^{j+k}$ is at least $3 \cdot (k+1)$.*

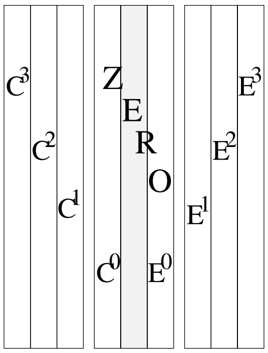
   Proof. This is easily verified by a computer program. We mention that for $k \leq 6$, an average of 3 cannot be assured (see Appendix B for an example). ■

   Next we create a partition of the 32 columns into several groups. We pick a non-zero column C^j. Now following Claim 3, there exists $(k+1)$-columns $(0 \leq k \leq 7)$ such that the average weight of each column is at least 3. Consider the smallest k that achieves this. Then put these $(k+1)$ columns $C^j, C^{j+1}, \cdots, C^{j+k}$ into a group. Call these columns good columns and the group a good group. We then choose C^{k+j+1} and form another group. We continue like this till no more good groups can be created. The remaining columns are then grouped together. Call this group a bad group. Note that the bad group has average weight at least 1. Now let e be the size of this bad group. Then we have $(32-e)$ good columns. Also following Claim 3, e

could be at most 7. Therefore the total weight of the codeword is at least $3 \cdot (32 - e) + e = 96 - 2 \cdot e \geq 82$.

2. (**At least One column Zero Case:**) Assume that there is at least one zero column. W.l.o.g. let C^0 be a zero column such that the column to the left of it is non-zero (note that such a column always exists since we are considering a non-zero codeword). Denote the columns to the left of C^0 by $C^1, C^2, \cdots$ (see figure).

   Also, going towards the right of C^0, denote the first non-zero column by E^1 and thereafter $E^2, E^3, \cdots$. Denote the column to the left of E^1 by E^0. (Note that it may be possible that C^0 and E^0 are the same column.) We argue that a few columns to the left and right of a band of zero columns must contribute a total weight of at least 82.



   It will be immaterial in our analysis below if there are some non-zero columns between C^0 and E^0. All we require in our analysis is that C^0 and E^0 are zero.

   Next consider $C^1, C^2, \cdots$. How soon can the sequence yield a zero column, i.e., what is the smallest value of j such that $C^j = E^0$? In order to answer this question, first note that since C^0 is everywhere zero, C^1 is essentially generated by the code whose parity check equations over $\mathbb{F}_2$ are given as follows: Denote $C^1 = \langle y_0, \cdots, y_{63} \rangle$. Then

$$\forall i, 16 \leq i \leq 63, \quad 0 = y_i + y_{i-3} + y_{i-8} + y_{i-14} + y_{i-16}. \tag{7}$$

Similarly for a fixed C^1, the column C^2 is generated by the code whose parity check equations over $\mathbb{F}_2$ are given as follows: Denote $C^2 = \langle x_0, \cdots, x_{63} \rangle$. Let $u_i \overset{def}{=} x_i + x_{i-3} + x_{i-8} + x_{i-14} + x_{i-16}$.

$$0 = \begin{cases} u_i + y_{i-1} + y_{i-2} + y_{i-15} & \text{for } 16 \leq i \leq 19 \\ u_i + y_{i-1} + y_{i-2} + y_{i-15} + y_{i-20} & \text{for } 20 \leq i \leq 63 \end{cases} \tag{8}$$

On the other hand E^1 is generated by the code whose parity check equations over $\mathbb{F}_2$ are given as follows: Denote $E^1 = \langle w_0, \cdots, w_{63} \rangle$. Then

$$0 = \begin{cases} w_{i-1} + w_{i-2} + w_{i-15} & \text{for } 16 \leq i \leq 19 \\ w_{i-1} + w_{i-2} + w_{i-15} + w_{i-20} & \text{for } 20 \leq i \leq 63 \end{cases} \tag{9}$$

Similarly for a fixed E^1, the column E^2 is generated by the code whose parity check equations over $\mathbb{F}_2$ are given as follows: Denote $E^2 = \langle z_0, \cdots, z_{63} \rangle$. Let $v_i \overset{def}{=} w_i + w_{i-3} + w_{i-8} + w_{i-14} + w_{i-16}$. Then

$$0 = \begin{cases} v_i + z_{i-1} + z_{i-2} + z_{i-15} & \text{for } 16 \leq i \leq 19 \\ v_i + z_{i-1} + z_{i-2} + z_{i-15} + z_{i-20} & \text{for } 20 \leq i \leq 63 \end{cases} \tag{10}$$

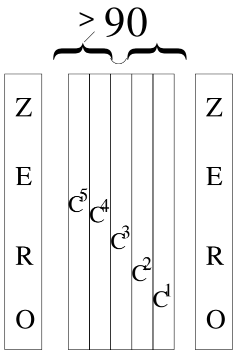
The following claim shows that at least four consecutive columns have to be non-zero.

Claim 4. *If C^0 is everywhere zero, and C^1 is non-zero, then C^2, C^3 and C^4 are all non-zero.*

Proof. Suppose for a j it is the case that $C^j = E^1$, i.e., C^{j+1} is all zero. Then a homogeneous system of linear equations over $\mathbb{F}_2$ can be set up. Consider the $64 \times j$ variables in column C^1 through C^j. There are 48 equations for each of the columns C^1 through C^j. Also, there are 48 more equations for C^{j+1}. It is well known that such a system can have a non-trivial solution if and only if the rank of the co-efficient matrix is strictly smaller than the number of variables. It can easily be verified by a computer program that for $j = 1, 2, 3$, the system has full rank, that is exactly $64 \times j$. This can also be proved algebraically for $j = 1, 2$. We give a simple algebraic proof in the appendix (see Appendix A). ■

This proof also highlights that for the rank to be full the recurrence relation must satisfy nice properties. Ranks of all linear systems considered in this paper have been computed using Gaussian elimination. We now divide the proof into two cases.

(a) (**Number Of Consecutive Non-Zero Columns is at most Five):** By the claim above, we can safely assume that we have at least four consecutive non-zero columns. Also, if we assume $C^4 = E^1$, then the number of nontrivial solutions can be at most $2^{16} - 1$ (since the co-rank or nullity of the matrix is 16, as verified by implementing a Gaussian elimination program). Similarly, assuming $C^5 = E^1$, the number of nontrivial solutions can be at most $2^{32} - 1$. We do an exhaustive search to conclude that the minimum weight in the latter case is at least 90. (Note that this latter case alone is sufficient.)



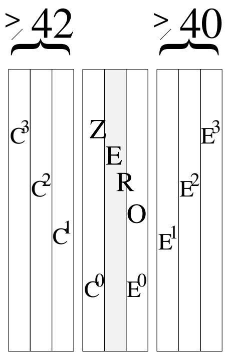
Case 2(a)

(b) (**Number Of Consecutive Non-Zero Columns is at least Six**): If case 1 and case 2(a) do not hold then, the only case that remains to be considered is the one where at least six consecutive columns are non-zero. Note that C^1, C^2, C^3 are then distinct from E^1, E^2, E^3. We use a computer program to verify that in this case the combined weight of C^1, C^2 and C^3 is at least 42. However the same cannot be said of E^1, E^2, E^3, and we have to do a more detailed analysis.

Now recall Equation 9, the constraints induced on E^1. A quick observation reveals that its free variables are the first 15 bits and the very last bit. Depending on the values taken by E^1s first 15 bits we sub-divide our proof into two cases:

i. (**Non-Pathological Case:**) Not all of the first 15 bits of E^1 are zero.

This is the simpler case. In this case, the recurrence induces a good expansion. By an exhaustive search we obtain that in this case the combined weight of E^1, E^2 and E^3 is at least 40. Since the combined weight of C^1, C^2 and C^3 is at least 42, and that C^i, E^i are all distinct, together they establish this case.



Case 2(b)i

ii. (**Pathological Case:**) Here we assume that the first 15 variables of E^1 are all zero. This is the most subtle and difficult case. Going back to Equation 9, we note that in this case it must hold that $w_{63} = 1$ and for all $0 \leq i \leq 62, w_i = 0$. We call such w *pathological*.

Now consider Equation 10. We can have two cases here.

In the first case, assume that the first 15 variables of z are zero. In that case, it must hold that $z_{62} = 1$. (Plugging in $i = 16$ to 62 in Equation 10 will yield $z_j = 0$ for all $15 \leq j \leq 61$ since $w_i = 0$ for these values.) Also note that z_{63} is free. In this case, we also call z pathological. In fact this may continue along the diagonal i.e., $E^3, E^4, \cdots$ may be pathological. If that happens then it is easy to show that the first non-zero bits of E^3 will be its 61^{st} bit, that of E^4 will be 60^{th} bit and so on. Also each column will have a free variable in its 63^{rd} bit.

In the second case, we assume that not all of its first 15 variables are zero. We call such z's to be non-pathological.

We now sub-divide into many small cases depending primarily on the number of pathological columns (and thus on the number of free variables).

A. (**# Pathological Columns** ≤ 8) We break this case into two sub-cases. That each of these sub-cases holds has been verified using a computer program.

(I). 6^{th} **and earlier non-pathological columns are non-zero:** In this case, we verify that the combined weight of the pathological columns and the first three non-pathological columns to the right of the pathological columns is at least 40. This ensures that in this case the minimum weight is at least 82.

We mention that the search space has dimension

$$\# \text{ of Pathological vars} + \# \text{ of Non-Pathological Cols.} \times 16,$$

which is at most 40 in this case.

We next consider the case where the non-pathological columns are same as one of C^1, C^2 or C^3.

(II). **6th or earlier non-pathological column is identically zero:** Firstly note that it suffices to check the case where the 6th non-pathological column is identically zero (that is $E^3 = C^3$), since other cases do fall in this case. Now we consider the parity check equations induced on the pathological columns and the six non-pathological columns. Note that C^1 satisfies Equation 7 and that E^1 satisfies Equation 9. Also note that in between columns satisfy equations similar to Equations 8 and 10. These equations then set up a homogeneous system of linear equations whose nullity can be verified (by a computer program) to be at most 40.

Let the number of pathological columns be p and the number of non-pathological columns be n. Specifically then the nullity of the system can then be shown to be exactly (see Appendix A Claim 9) : $p + 64 \times n - 48 \times (n + 1) = p + 16 \cdot n - 48$, which is at most 40 in this case. We do an exhaustive search over the null space to establish that the min-weight is at least 82.

B. ($8 <$ # **Pathological Columns** ≤ 16): We also break this case into two sub-cases. That each of these sub-cases holds has been verified using a computer program.

(I). 5th **and earlier non-pathological columns are non-zero:** In this case, we verify that the combined weight of the pathological columns and the first two non-pathological columns to the right of the pathological columns is at least 40. This ensures that in this case the minimum weight is at least 82.

Therefore the case that remains to be considered is the one where the non-pathological columns are same as one of C^2 or C^3 which leads us to the next case.

(II). **5th or earlier non-pathological column is identically zero:** Firstly, note that it suffices to check the case when the 5th non-pathological column is identically zero (that is $E^2 = C^3$), since other cases do fall in this case. As in the 2nd subcase of the previous case (i.e., Case 2(b)(ii)(A)(II)), we verify that the min-weight is at least 82.

C. ($16 <$ **Pathological Columns** ≤ 28**):** First of all, notice that 28 columns is enough, since by our assumption there is at least one zero column and three non-pathological column (i.e., C^1, C^2, C^3). Now, we also break this case into two sub-cases. That each of these sub-cases holds has been verified using a computer program.

(I). 4th **and earlier non-pathological columns are non-zero:** In this case, we verify that the combined weight of the pathological columns and the first non-pathological column to the right of the pathological columns is at least 40. This ensures that in this case the minimum weight is at least 82.

Therefore the case that remains to be considered is the one where the 1st non-pathological column is the same as C^3.

(II). **4th non-pathological column is identically zero:** As in the 2nd sub-case of the previous case (or Case 2(b)(ii)(A)(II)), we verify that the min-weight is at least 82.

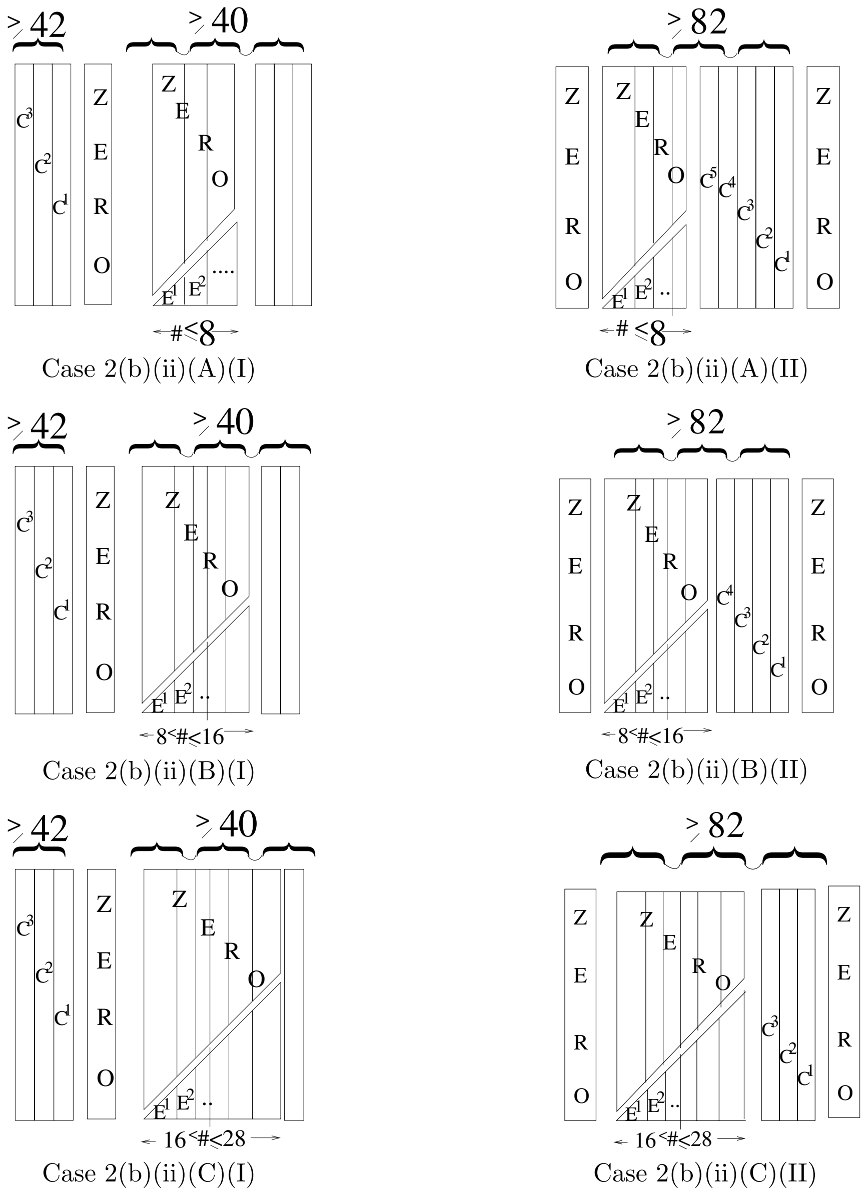


Fig. 1. Illustrations of various cases in the proof of Theorem 2

We remark that the minimum weight of this code can at most be 82 and therefore our result is tight (see Appendix B). Extending our approach, we can further prove the following theorem whose proof has been deferred to the full version.

Theorem 5. *The code $\mathcal{C}64$, as defined by Equation (6), has minimum weight at least 75 (and at least 52) in its last 60 words (and in its last 48 words, respectively).*

We remark that a simple variants of the above technique can be used to give a lower bound on the minimum weight of SHA-1 (of course, there are much fewer cases to consider here). Specifically we have the following theorem whose proof has been deferred to the full version of this paper (also see [6]).

Theorem 6. *SHA-1 message expansion code has minimum weight 25 in the last 60 words.*

5 Conclusion

In this paper we have shown how lower bounds on minimum weight of quasi-cyclic linear codes of dimension $m \times n$ given by parity equations of the form

$$W_i = \sum_{t=1}^{i} a_{it} W_{i-t} + \left(\left(\sum_{t=1}^{i} b_{it} W_{i-t} \right) <<< 1 \right) \qquad \text{for } i \geq n,$$

can be obtained by reducing the problem to the minimum weight of significantly smaller dimensional codes. Note that this equation is more general than Equation (5), and Equation (2) is of this form rather than the simpler Equation (5). In some cases, we obtain the exact minimum weight, including the example codes we considered. An obvious generalization is to consider three or more column mixing (the equation above has only two column mixing), which could lead to codes with even better minimum distance.

A common paradigm for designing hash functions, including MD5[12], SHA-0, SHA-1 and SHA-2[16] is the following: the 512-bit message is first expanded into N words, and then the N words are used as step keys (sometimes known as round keys) in N steps of a (non-linear) block cipher invoked on an initial vector. The output of the block cipher is the output of the compression function. As pointed out in the Introduction, one of the key ingredients of the recent differential attacks on MD5, SHA-0, and SHA-1 has been their poor message expansion (in terms of minimum weight) into the N words. We propose SHA1-IME which is SHA-1 with the original message expansion (see [15]) substituted by our improved message expansion as given in Equation 2. A preliminary evaluation has shown that this proposed compression function has at most a 5% overhead in speed over SHA-1 in a software implementation, and at most a 10% overhead in gate count in a high performance hardware implementation. However, on the positive side this proposed compression function resists all presently known attacks against SHA-1. Thus, we consider our novel technique to be an important advance in the design of collision-resistant hash functions.

References

1. Biham, E., Chen, R.: Near collisions of SHA-0. In: Franklin, M. (ed.) CRYPTO 2004. LNCS, vol. 3152, Springer, Heidelberg (2004)
2. Biham, E., Chen, R.: New results on SHA-0 and SHA-1. In: Franklin, M. (ed.) CRYPTO 2004. LNCS, vol. 3152, Springer, Heidelberg (2004)
3. Chabaud, F., Joux, A.: Differential collisions in SHA-0. In: Krawczyk, H. (ed.) CRYPTO 1998. LNCS, vol. 1462, Springer, Heidelberg (1998)
4. Chepyzhov, V.V.: New lower bounds for minimum distance of linear quasi-cyclic and almost linear cyclic codes. Problems of information Transmission 28(1) (1992)
5. Dumer, I., Micciancio, D., Sudan, M.: Hardness of approximating the minimum distance of a linear code. IEEE Transaction on Information Theory 49(1) (2003)
6. Jutla, C.S., Patthak, A.C.: A Matching Lower Bound on the Minimum Weight of SHA-1 Expansion Code. Cryptology ePrint Archive, Report 2005/266 (2005), `http://eprint.iacr.org/`
7. Kasami, T., Lin, S., Peterson, W.W.: New Generalization of the Reed-Muller Codes Part I: Primitive Codes. IEEE Transactions on Information Theory IT-14(2), 189–199 (1968)
8. Lally, K.: Quasicyclic codes of index ℓ over $\mathbb{F}_q$ Viewed as $\mathbb{F}_q[x]$-submodules of $\mathbb{F}_{q^l}[x]/\langle x^m - 1\rangle$. In: Fossorier, M.P.C., Høholdt, T., Poli, A. (eds.) Applied Algebra, Algebraic Algorithms and Error-Correcting Codes. LNCS, vol. 2643, Springer, Heidelberg (2003)
9. Ling, S., Solé, P.: Structure of quasi-clcyic codes III: Generator theory. In: IEEE Transaction on Information Theory (2005)
10. Matusiewicz, K., Pieprzyk, J.: Finding good differential patterns for attacks on SHA-1. In: International Workshop on Coding and Cryptography (2005)
11. Rijmen, V., Oswald, E.: Update on SHA-1. In: Menezes, A.J. (ed.) CT-RSA 2005. LNCS, vol. 3376, Springer, Heidelberg (2005)
12. Rivest, R.: RFC1321: The MD5 message-digest algorithm. In: Internet Activities Board (1992)
13. Townsend, R.L., Weldon, E.J.: Self-orthogonal quasi-cyclic codes. IEEE Transaction on Information Theory (1967)
14. United States Department of Commerce, National Institute of Standards and Technology, Federal Information Processing Standard Publication #180. Secure Hash Standard (1993)
15. United States Department of Commerce, National Institute of Standards and Technology, Federal Information Processing Standard Publication #180-1 (addendum to [14]). Secure Hash Standard (1995)
16. United States Department of Commerce, National Institute of Standards and Technology, Federal Information Processing Standard Publication #180-2. Secure Hash Standard (August 2002)
17. van Lint, J.H.: Introduction to Coding Theory. Springer, Heidelberg (1998)
18. Vardy, A.: The intractability of computing the minimum distance of a code. IEEE Transaction on Information Theory 43(6) (1997)
19. Wang, X., Yao, A., Yao, F.: New collision search for SHA-1. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, Springer, Heidelberg (2005)
20. Wang, X., Yu, H., Yin, Y.L.: Efficient collision search attacks in SHA-0. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, Springer, Heidelberg (2005)
21. Wang, X., Yu, H., Yin, Y.L.: Finding collisions in the full SHA-1. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, Springer, Heidelberg (2005)

22. Wang, X.Y.: The collision attack on SHA-0. In Chinese (1997)
23. Wang, X.Y.: The Improved collision attack on SHA-0. In Chinese (1997), http://www.infosec.edu.cn/
24. Zierler, N.: On a variation of the first-order reed-muller codes. In: M.I.T. Lincoln Lab., Group Report, 34-80, Lexington, Mass (October 1958)

A Rank Proofs

Claim 7. *If C^0 is zero, and C^1 is non-zero, then C^2 is non-zero.*

Proof. Assume otherwise i.e., that C^2 is zero. Consider the 48×64 dimensional parity check matrices (essentially Equations (7) and (9)) over $\mathbb{F}_2$.

$$\begin{pmatrix} 1010000001000010010000 \cdots & 000000000000000000 \\ 0101000000100001001000 \cdots & 000000000000000000 \\ \ddots \qquad\quad \cdots & \ddots \\ 0000000000000000000000 \cdots & 010100000100001001 \\ \end{pmatrix}$$

$$H_1$$

$$\begin{pmatrix} 0100000000000011000000 \cdots & 000000000000000000000 \\ 0010000000000001100000 \cdots & 000000000000000000000 \\ 0001000000000000110000 \cdots & 000000000000000000000 \\ 0000100000000000011000 \cdots & 000000000000000000000 \\ 1000010000000000001100 \cdots & 000000000000000000000 \\ 0100001000000000000110 \cdots & 000000000000000000000 \\ \ddots \qquad\quad \cdots & \ddots \\ 0000000000000000000000 \cdots & 100001000000000000110 \\ \end{pmatrix}$$

$$H_2$$

Then we need to show that $H = \begin{pmatrix} H_1 \\ H_2 \end{pmatrix}$ has full rank. For that it is enough to show that there are 64 linearly independent rows. We consider the 48 rows of H_1 and 16 additional rows, namely 5^{th} through 20^{th} rows of H_2. We reduce the problem to showing that a certain equation over polynomial ring $\mathbb{F}_2[x]$ does not have solutions in a restricted set of polynomials. We associate with the vector $c = \langle c_0, \cdots, c_{63} \rangle$ in $\mathbb{F}_2^{64}$ the polynomial $c(x) = \sum_{i=0}^{63} c_i x^i$ in $\mathbb{F}_2[x]$. Then the following polynomials can be associated with the 1^{st} and 5^{th} rows of matrix H_1 and H_2, respectively:

$$p(x) \overset{def}{=} x^{16} + x^{13} + x^8 + x^2 + 1, \text{ and } r(x) \overset{def}{=} x^{19} + x^{18} + x^5 + 1.$$

Further note that the i^{th} (note $1 \leq i \leq 48$) row of H_1 then gets associated with $x^{i-1}p(s)$. Similarly the j^{th} (note we restrict ourselves to $5 \leq j \leq 20$) row of H_2 then gets associated with $x^{j-5}r(s)$. Therefore, observe that if the 80 rows that

we are considering were dependent then we would have a non-zero solution of the following polynomial equation : $p(x)\alpha(x) + \beta(x)r(x) = 0$, with additional constraints that degree$(\alpha) \leq 47$ and degree$(\beta) \leq 15$. However, it is easy to check that $p(x)$ is irreducible, therefore if such a equation holds then it must be the case that $p(x)$ divides $r(x)$. However, it is easy to check that $p(x)$ does not divide $r(x)$, thus leading to a contradiction.
Therefore H has full rank. ■

We now strengthen the claim slightly.

Claim 8. *If C^0 is zero, and C^1 is non-zero, then both C^2, C^3 are non-zero.*

Proof. Consider the following polynomials :

$$p(x) \overset{def}{=} x^{16} + x^{13} + x^8 + x^2 + 1,$$

$$q(x) \overset{def}{=} x^{15} + x^{14} + x, \text{ and } r(x) \overset{def}{=} x^{19} + x^{18} + x^5 + 1 = x^4 \cdot q(x) + 1.$$

Let H_1 and H_2 be as above. First of all note that H_2 has full rank. (This is clear from the matrix. Otherwise, note that we would have an identity: $q(x) \cdot a(x) + r(x) \cdot b(x) = 0$, with degree$(a) \leq 3$ and degree$(b) \leq 43$. Since degree$(q \cdot a) <$ degree(r), this cannot happen.) Now we will show that the rank of the matrix

$$\begin{pmatrix} H_2 & 0 \\ H_1 & H_2 \\ 0 & H_1 \end{pmatrix}$$

is at least 128. Since H_1 has full rank, observe that $\begin{pmatrix} H_1 & H_2 \\ 0 & H_1 \end{pmatrix}$ has rank at least 96. So consider the following 92 independent rows from the above matrix, namely 5^{th} row onwards. We also argue that another additional 5^{th} through 40^{th} rows of the top H_2 are also independent. If not, then they would satisfy the following polynomial equations

$$\alpha(x)p(x) + \beta(x)r(x) = 0 \qquad (11)$$
$$x^4\beta(x)p(x) + \gamma(x)r(x) = 0 \qquad (12)$$

with restrictions degree$(\alpha) \leq 47$, degree$(\beta) \leq 43$, and degree$(\gamma) \leq 35$.

Since $p(x)$ is an irreducible polynomial, and $p(x) \nmid r(x)$, observe from Equation (11) that $p(x)|\beta(x)$. Hence, set $\beta(x) = \mu(x)p(x)$. Substituting in Equation (12) we get

$$x^4p(x)^2\mu(x) + \gamma(x)r(x) = 0.$$

Since $p(x)$ is irreducible, and $p(x) \nmid r(x)$, and $x \nmid r(x)$, it must hold that $x^4 p(x)^2|\gamma(x)$. But that is impossible, since degree$(\gamma) \leq 35 < 36 =$degree$(x^4p(x)^2)$. ■

Recall that we used E^0 to denote a column that is zero everywhere. Also, recall that the columns left to E^0 are denoted E^1, E^2 and so on. In the following claim, we will assume $3 \leq n$.

Claim 9. *Let $E^1, E^2, \cdots, E^p$ be p pathological columns. Also, let $E^{p+1}, E^{p+2}, \cdots,$ E^{p+n} be n non-pathological columns. Further assume that $E^{p+n+1} = C^0$ is everywhere zero. If the nullity of the parity check equations resulting from these columns for $p = 0$ is $16 \cdot n - 48$, then the nullity of the parity check equations resulting from these columns for any $p \leq 28$ is $p + 16 \cdot n - 48$.*

Proof. Let $N_{i,j}$, $(1 \leq i \leq n, 0 \leq j \leq 63)$ denote the entries in the non-pathological columns. Also let $P_{i,j}$, $(1 \leq i \leq p,$ for each $i,$ $64-i \geq j \leq 63)$ be the pathological variables. We will denote $N_i = \langle N_{i,0}, \cdots, N_{i,63}\rangle$ and $P_i = \langle P_{i,64-i}, \cdots, P_{i,63}\rangle$. Let $H_{1|i}$ denote the matrix H_1 restricted to the last i columns. (Note that only the last i rows will be non-zero.) Also let $H_{2|i}$ denote the matrix H_2 restricted to the last i columns. (Note that only the last $i-1$ rows will be non-zero.) Note that $\langle P_1, \cdots, P_p, N_1, \cdots, N_n\rangle$ must belong to the null space of the following matrix:

$$\mathcal{H} = \begin{pmatrix} H_{1|1} & H_{2|2} & & & & & & & \\ & H_{1|2} & H_{2|3} & & & & & & \\ & & \ddots & \ddots & & & & & \\ & & & H_{1|p-1} & H_{2|p} & & & & \\ & & & & H_{1|p} & H_2 & & & \\ & & & & & H_1 & H_2 & & \\ & & & & & & \ddots & \ddots & \\ & & & & & & & H_1 & H_2 \\ & & & & & & & & H_1 \end{pmatrix}$$

Note that when we restrict H_1 or H_2 to the last few columns, the top rows in that restricted entries may become zero. We remove such rows if the entire row in the above matrix $\mathcal{H}$ becomes everywhere zero. Note that with this modification, the following sub-matrix is already in the echelon form:

$$\mathcal{H}_1 = \left.\begin{pmatrix} H_{1|1} & H_{2|2} & & & \\ & H_{1|2} & H_{2|3} & & \\ & & \ddots & \ddots & \\ & & & H_{1|p-1} & H_{2|p} \end{pmatrix}\right\} (p-1) \text{ blocks}$$

(Observe that first block corresponding to $(H_{1|1}\ H_{2|2})$ reduces to $(1\ 10)$, and that corresponding to $(H_{1|2}\ H_{2|3})$ reduces to $\begin{pmatrix} 10\ 100 \\ 01\ 110 \end{pmatrix}$.)

Furthermore, since by assumption the following sub-matrix has full rank:

$$\mathcal{H}_2 = \left.\begin{pmatrix} H_2 & & & \\ H_1 & H_2 & & \\ \ddots & \ddots & & \\ & & H_1 & H_2 \\ & & & H_1 \end{pmatrix}\right\} (n+1) \text{ blocks}$$

the matrix $\mathcal{H}$ has full rank. Note here that in the top $48-p$ rows, $H_{1|p}$ is entirely zero. However these rows in $\mathcal{H}$ are independent since $\mathcal{H}_2$ has full rank. In the remaining rows $H_{1|p}$ is in echelon form and hence independent. Note that the number of rows i.e., number of constraints is:

$$48 \times (n+1) + \sum_{i=1}^{p-1} i = 48(n+1) + \frac{p(p-1)}{2}.$$

Also, note the number of variables i.e., columns is

$$64 \times n + \sum_{i=1}^{p} i = 64 \cdot n + \frac{p(p+1)}{2}.$$

Thus the nullity of the system is

$$64 \cdot n + \frac{p(p+1)}{2} - \left(48(n+1) + \frac{p(p-1)}{2}\right) = p + 16 \cdot n - 48.$$

This completes the proof. ∎

B Examples

We cite below an example where over 7 columns an average of 3 does not hold. Below we only give 8 columns and the columns are placed horizontally. Note that the 8 columns yield 29, whereas the first 7 columns yield only 14.

```
0000000000000000000000000000000000000000000000000000000001000000
0000000000000000000000000000000000000000000000000000000000110110
0000000000000000000000000000000000000000000000000000000000010100
0000000000000000000000000000000000000000000000000000000000001110
0000000000000000000000000000000000000000000000000000000000000100
0000000000000000000000000000000000000000000000000000000000000011
0000000000000000000000000000000000000000000000000000000000000001
1000101010000000001001000010000010000100101100000010001000010000
```

Below is a codeword in the code defined by Equation (6) with optimal minimum weight. We found the following codeword while searching for Case 2(b)(ii)(A)(II). Below we only give eight columns that includes six non-zero and two zero columns. The rests are all zero columns. Below the columns are placed horizontally.

```
0000000000000000 0000000000000000 0000000000000000 0000000000000000
0011110010011110 1000000001101001 1101001001010110 0000110010010000
1011000101000100 0010111101001000 1011100010101100 1101000000101111
1010101000111011 0010100100110010 1000000101001000 0110011000000000
0000000000000000 0000000000000000 0000000000000000 0000000000000100
0000000000000000 0000000000000000 0000000000000000 0000000000000011
0000000000000000 0000000000000000 0000000000000000 0000000000000001
0000000000000000 0000000000000000 0000000000000000 0000000000000000
```

Author Index