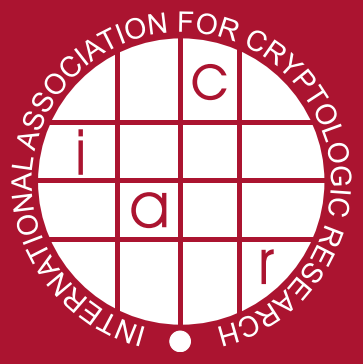
Carlos Cid
Christian Rechberger (Eds.)

LNCS 8540

Fast Software Encryption

21st International Workshop, FSE 2014
London, UK, March 3–5, 2014
Revised Selected Papers



Springer

Lecture Notes in Computer Science 8540

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

Editorial Board

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

Carlos Cid · Christian Rechberger (Eds.)

Fast Software Encryption

21st International Workshop, FSE 2014
London, UK, March 3–5, 2014
Revised Selected Papers

Springer

Editors
Carlos Cid
Royal Holloway, University of London
Egham
UK

Christian Rechberger
Technical University of Denmark
Lyngby
Denmark

ISSN 0302-9743 ISSN 1611-3349 (electronic)
Lecture Notes in Computer Science
ISBN 978-3-662-46705-3 ISBN 978-3-662-46706-0 (eBook)
DOI 10.1007/978-3-662-46706-0

Library of Congress Control Number: 2015937349

LNCS Sublibrary: SL4 – Security and Cryptology

Springer Heidelberg New York Dordrecht London

Printed on acid-free paper

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

Preface

The 21st International Workshop on Fast Software Encryption (FSE 2014) was held in London March 3–5, 2014. The workshop was organized in cooperation with the International Association for Cryptologic Research, and took place at London's Natural History Museum. The workshop had 156 registered participants, of which 31 were students.

The FSE 2014 Program Committee comprised 26 members, and counted on the support of 75 external reviewers. We received 99 valid submissions, and each submission was reviewed by at least three PC members. After more than two months of deliberation and discussions, a total of 31 papers were accepted. This has been the highest number of accepted papers for an FSE so far, driven by the rather high quality of submissions. We are very grateful to all PC members and reviewers for their effort and contribution to the selection of an outstanding program of original articles in symmetric cryptography.

Besides the 31 selected talks, the workshop program also included two invited talks: Thomas Johansson from Lund University spoke on the application of low weight polynomials in cryptography; Thomas Ristenpart from the University of Wisconsin-Madison closed the workshop with the talk "New Encryption Primitives for Uncertain Times." The workshop also featured a rump session, chaired by Dan Bernstein and Tanja Lange, with several short informal presentations.

As it is tradition, the FSE 2014 Program Committee was asked to select the best submissions to the workshop, based on their scientific quality and contribution. Two submissions received the award for best papers: "Direct Construction of Recursive MDS Diffusion Layers using Shortened BCH Codes" by Daniel Augot and Matthieu Finiasz, and "Differential-Linear Cryptanalysis Revisited" by Céline Blondeau, Gregor Leander, and Kaisa Nyberg. The two papers also received a special solicitation for submission to the *Journal of Cryptology*.

In addition to the authors, PC members, and external reviewers, several other people contributed to the success of FSE 2014: colleagues, students, and supporting staff at DTU and Royal Holloway (in particular Claire Hudson); Shai Halevi, Greg Rose and abhi shelat at the IACR; the members of the FSE Steering Committee; Anne Kramer at Springer; and staff at the Natural History Museum. We were also fortunate to count on the financial support of four sponsors (CESG, KPMG, NXP, and Visa Europe), which made it possible to hold the event in such an impressive venue. We are very grateful to you all for your support.

It was a great honor to have been in charge of the organization of FSE 2014 and to coordinate the selection of its scientific program. It gave us the opportunity to work with a number of outstanding researchers and professionals in the cryptographic community; we were very pleased with its success and greatly enjoyed it. We hope the reader also enjoys the papers in these proceedings.

January 2015

Carlos Cid
Christian Rechberger

FSE 2014

21st International Workshop on Fast Software Encryption
Natural History Museum, London, UK
March 3–5, 2014

General Chairs

Carlos Cid	Royal Holloway, University of London, UK
Christian Rechberger	Technical University of Denmark, Denmark

Program Chairs

Carlos Cid	Royal Holloway, University of London, UK
Christian Rechberger	Technical University of Denmark, Denmark

Program Committee

Martin R. Albrecht	Technical University of Denmark, Denmark
Elena Andreeva	Katholieke Universiteit Leuven, Belgium
Kazumaro Aoki	NTT, Japan
Frederik Armknecht	University of Mannheim, Germany
Daniel J. Bernstein	University of Illinois at Chicago, USA, and Technische Universiteit, Eindhoven, The Netherlands
John Black	University of Colorado at Boulder, USA
Anne Canteaut	Inria Paris-Rocquencourt, France
Joan Daemen	STMicroelectronics, Belgium
Christophe De Cannière	Google, Switzerland
Orr Dunkelman	University of Haifa, Israel
Martin Hell	Lund University, Sweden
Dmitry Khovratovich	University of Luxembourg, Luxembourg
Gregor Leander	Ruhr Universität Bochum, Germany
Subhamoy Maitra	Indian Statistical Institute, Kolkata, India
Mitsuru Matsui	Mitsubishi Electric, Japan
Florian Mendel	Technische Universtität Graz, Austria
Svetla Nikova	Katholieke Universiteit Leuven, Belgium
Elisabeth Oswald	University of Bristol, UK
Thomas Peyrin	Nanyang Technological University, Singapore
Josef Pieprzyk	Macquarie University, Australia
Martijn Stam	University of Bristol, UK
François-Xavier Standaert	Université catholique de Louvain, Belgium
Serge Vaudenay	EPFL, Switzerland
Hongbo Yu	Tsinghua University, China

External Reviewers

Hoda A. Alkhzaimi
Gilles Van Assche
Jean-Philippe Aumasson
Subhadeep Banik
Harry Bartlett
Aslı Bay
Guido Bertoni
Begul Bilgin
Andrey Bogdanov
Sonia Bogos
Christina Boura
Daniel Cabarcas
Claude Carlet
Anupam Chattopadhyay
Alexandre Duc
François Durvaux
Maria Eichlseder
Sebastian Faust
Vincent Grosso
Sourav Sen Gupta
Jialin Huang
Andreas Hülsing
Takanori Isobe
Tetsu Iwata
Guo Jian
Philipp Jovanovic
Angela Jäschke
Pierre Karpman
Elif Kavun
Nathan Keller
Stéphanie Kerckhof
Lars R. Knudsen
Matthias Krause
Stefan Kölbl
Martin M. Lauridsen
Gaëtan Leurent
Zhiqiang Liu
Atul Luykx
Daniel Martin
Bart Mennink
Vasily Mikhalev
Pawel Morawiecki
Nicky Mouha
Tomislav Nad
Mridul Nandi
Ivica Nikolic
Kaisa Nyberg
Kenny Paterson
Michaël Peeters
Ludovic Perret
Christiane Peters
Romain Poussier
Santos Merino Del Pozo
Francesco Regazzoni
Reza Reyhanitabar
Vincent Rijmen
Phillip Rogaway
Santanu Sarkar
Martin Schläffer
Peter Schwabe
Takeshi Shimoyama
Paul Stankovski
Ron Steinfeld
Petr Susil
Seth Terashima
Tyge Tiessen
Kerem Varici
Vesselin Velichkov
Huaxiong Wang
Lei Wang
Meiqin Wang
Quingju Wang
Gaven Watson
Tolga Yalcin
Yusi (James) Zhang

Contents

Designs

Direct Construction of Recursive MDS Diffusion Layers Using Shortened BCH Codes

Daniel Augot[1] and Matthieu Finiasz[2(✉)]

[1] INRIA - LIX UMR 7161 X-CNRS, Paris, France
[2] CryptoExperts, Paris, France
finiasz@gmail.com

Abstract. MDS matrices allow to build optimal linear diffusion layers in block ciphers. However, MDS matrices cannot be sparse and usually have a large description, inducing costly software/hardware implementations. Recursive MDS matrices allow to solve this problem by focusing on MDS matrices that can be computed as a power of a simple companion matrix, thus having a compact description suitable even for constrained environments. However, up to now, finding recursive MDS matrices required to perform an exhaustive search on families of companion matrices, thus limiting the size of MDS matrices one could look for. In this article we propose a new *direct* construction based on shortened BCH codes, allowing to efficiently construct such matrices for whatever parameters. Unfortunately, not all recursive MDS matrices can be obtained from BCH codes, and our algorithm is not always guaranteed to find the best matrices for a given set of parameters.

Keywords: Linear diffusion · Recursive MDS matrices · BCH codes

1 Introduction

Diffusion layers are a central part of most block cipher constructions. There are many options when designing a diffusion layer, but linear diffusion is usually a good choice as it can be efficient and is easy to analyze. The quality of a linear diffusion layer is connected to its *branch number* [3]: the minimum over all possible nonzero inputs of the sum of the Hamming weights of the input and the corresponding output of this diffusion layer. A high branch number implies that changing a single bit of the input will change the output a lot, which is exactly what one expects from a good diffusion layer. Before going into more details on how to build linear diffusion with a high branch number, let us recall some elements of coding theory.

Linear Diffusion and Coding Theory. A linear code Γ of dimension k and length n over $\mathbb{F}_q$ (denoted as an $[n,k]_q$ code) is a vectorial subspace of dimension k of $(\mathbb{F}_q)^n$. Elements of Γ are called code words. The minimal distance d of a code is the minimum over all nonzero code words $c \in \Gamma$ of the Hamming weight

C. Cid and C. Rechberger (Eds.): FSE 2014, LNCS 8540, pp. 3–17, 2015.
DOI: 10.1007/978-3-662-46706-0_1

of c. A $[n,k]_q$ code of minimal distance d will be denoted as an $[n,k,d]_q$ code. A generator matrix G of a code is any $k \times n$ matrix over $\mathbb{F}_q$ formed by a basis of the vectorial subspace Γ. We say a generator matrix is in systematic form when it contains (usually on the left-most positions) the $k \times k$ identity matrix I_k. The non-systematic part (or redundancy part) of G is the $k \times (n-k)$ matrix next to this identity matrix.

Now, suppose a linear diffusion layer of a block cipher is defined by an invertible matrix M of size $k \times k$ over $\mathbb{F}_q$, so that an input $x \in (\mathbb{F}_q)^k$ yields an output $y \in (\mathbb{F}_q)^k$ with $y = x \times M$. Then, the $k \times 2k$ generator matrix G_M having M as its non-systematic part (the matrix defined as the concatenation of the $k \times k$ identity matrix I_k and of M, as $G_M = [I_k \mid M]$) generates a $[2k,k]_q$ code Γ_M whose minimal distance is exactly the branch number of M. Indeed, a code word $c = x \times G_M$ in Γ_M is the concatenation of an input x to the diffusion layer and the corresponding output $y = x \times M$. So the Hamming weight of every code word is the sum of the Hamming weights of an input and its output.

Optimal linear diffusion can thus be obtained by using codes with the largest possible minimal distance, namely maximum distance separable (MDS) codes. A $[n,k]_q$ code is called MDS if its minimal distance is $d = n-k+1$. By extension, we will say that a matrix M is MDS when its concatenation with the identity matrix yields a generating matrix G_M of an MDS code Γ_M. In the context of diffusion where $n = 2k$ being MDS means that $d = k+1$: changing a single element in the input of the diffusion layer will change all the elements in its output.

We also recall the MDS conjecture: if there exists an $[n,k]_q$ MDS code, meaning an MDS code of length n and dimension k over $\mathbb{F}_q$, then $n \leq q+1$, except for particular cases which are not relevant to our context. All along this article we will assume that this conjecture holds [8].

Note on Vector Representation. In coding theory, vectors are usually represented as rows (with $y = x \times M$), as we have done for the moment. In cryptography, however, they are more often represented as columns (with $y = M^T \times x$). Luckily, the transposed of an MDS matrix is also MDS, so if G_M defines an MDS code, both M and M^T can be used as MDS diffusion matrices. In the rest of the article we will use the column representation, which people used to the AES and the MixColumns operation are more familiar with: the diffusion layer defined by a matrix M computes $y = M \times x$. This way, the branch number of M is the minimal distance of the code generated by $G_{M^T} = [I_k \mid M^T]$. However, in order to avoid matrix transpositions, we will rather check whether $G_M = [I_k \mid M]$ generates an MDS code or not.

Recursive MDS Matrices. MDS matrices offer optimal linear diffusion, but in general, they do not allow for a very compact description. Indeed, the non-systematic part M of an MDS generator matrix cannot contain any 0 element[1].

[1] If the non-systematic part M of an MDS generator matrix contained a 0, then the line of G_M containing this zero would have Hamming weight $\leq k$, which is in contradiction with the minimal distance of the code. More generally, for an MDS code Γ_M, for any $i \leq k$ all the $i \times i$ minors of M must be non-zero.

These matrices can never be sparse and applying such a matrix to its input requires a full matrix multiplication for the diffusion. Several different techniques have been studied to obtain *simpler* MDS matrices, a well known example being circulant matrices (or modifications of circulant matrices) as used in the AES [4] or FOX [7]. Recently a new construction has been proposed: the so-called *recursive* MDS matrices, that were for example used in Photon [5] or LED [6]. These matrices have the property that they can be expressed as a power of a companion matrix C. For example, in Photon, using the same decimal representation of elements of $\mathbb{F}_{256}$ as in [5]:

$$M = \begin{pmatrix} 1 & 2 & 1 & 4 \\ 4 & 9 & 6 & 17 \\ 17 & 38 & 24 & 66 \\ 66 & 149 & 100 & 11 \end{pmatrix} = C^4, \text{ with } C = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 2 & 1 & 4 \end{pmatrix} = \mathsf{Companion}(1, 2, 1, 4).$$

The advantage of such matrices is that they are particularly well suited for lightweight implementations: the diffusion layer can be implemented as a linear feedback shift register that is clocked 4 times (or more generally k times), using a very small number of gates in hardware implementations, or a very small amount of memory for software. The inverse of the diffusion layer also benefits from a similar structure, see Eq. (1) for a particular case.

Outline. In the next section, we will present previous methods that have been used to find recursive MDS matrices. Then, in Sect. 3, we will introduce BCH codes and shortened BCH codes, show that they too can yield recursive MDS matrices, and give a direct construction of such matrices. In Sect. 4 we will then describe an algorithm to explore all BCH codes and the MDS diffusion layers they yield for given parameters. We will conclude with a few experimental results.

2 Exhaustive Search for Recursive MDS Matrices

Exhaustive search for recursive MDS matrices can be quite straightforward:

- pick some parameters: the matrix size k and the field size $q = 2^s$,
- loop through all companion matrices C of size k over $\mathbb{F}_q$,
- for each C, computes its k-th power and check if it is MDS.

However, this technique is very expensive as there are many companion matrices (2^{ks}, which could be 2^{128} for an 128-bit cipher) and checking if a matrix is MDS is also expensive (the number of minors to compute is exponential in k). Also, it does not specially explore the most efficient matrices first. In the Photon example, the matrix uses very sparse coefficients (the field elements represented by 1, 2 and 4) to make the implementation of their operations on inputs even more efficient. Exhaustive search should focus on such matrices.

Following this idea, Sajadieh *et al.* [9] proposed to split the search in two. Their companion matrices are symbolic matrices $C(X)$ which have coefficients in the polynomial ring $\mathbb{F}_q[X]$ where X is an indeterminate, which will be substituted later by some $\mathbb{F}_2$-linear operator L of $\mathbb{F}_q$. Then their search space is reduced to symbolic companion matrices $C(X)$ whose coefficients are small degree polynomials in X (small degree polynomials will always yield a rather efficient matrix). Once $C(X)$ is raised to the power k, to get $D(X) = C(X)^k$, the matrix $D(X)$ will give an MDS matrix $D(L)$ when evaluated at a particular L, if for all $i \leq k$, all its $i \times i$ minors evaluated at L are invertible matrices (non-zero is enough in a field, but now the coefficients are $\mathbb{F}_2$-linear operators). Indeed, for a symbolic matrix $D(X)$, the minors are polynomials in X, and their evaluation at a particular linear operator L needs to be invertible matrices.

This way, for each matrix $C(X)$ explored during the search, the minors of all sizes of $D(X) = C(X)^k$ are computed: some matrices have minors equal to the null polynomial and can never be made MDS when X is substituted by a linear operator L, for the others this gives (many) algebraic polynomials in X which must not vanish when evaluated at L, for the k-th power $D(L)$ to be MDS. Then, the second phase of the search of Sajadieh *et al.* is to look for efficient operators L such that all the above minors are non zero when evaluated at L. The advantage of this technique is that it finds specially efficient recursive MDS matrices, but the computations of the minors of symbolic matrices can be pretty heavy, because of the growth of the degree of the intermediate polynomials involved. In the case of Photon, the matrix could be found as $C = \mathsf{Companion}(1, L, 1, L^2)$ where L is the multiplication by the field element represented by 2.

Continuing this idea and focusing on hardware implementation, Wu, Wang, and Wu [11] were able to find recursive MDS matrices using an impressively small number of XOR gates. They used a technique similar to Sajadieh *et al.*, first searching for symbolic matrices with a list of polynomials having to be invertible when evaluated in L, then finding an $\mathbb{F}_2$-linear operator L using a single XOR operation and with a minimal polynomial not among the list of polynomials that have to be invertible.

Then, looking for larger recursive MDS matrices, Augot and Finiasz [1] proposed to get rid of the expensive symbolic computations involved in this technique by choosing the minimal polynomial of L *before* the search of companion matrices $C(X)$. Then, all computation can be done in a finite field (modulo the chosen minimal polynomial of L), making them much faster. Of course, assuming the MDS conjecture holds, the length of the code cannot be larger than the size of the field plus one, so for an L with irreducible minimal polynomial of degree s, the field is of size $q = 2^s$, and k must verify $2k \leq 2^s + 1$. Larger MDS matrices will require an operator L with a higher degree minimal polynomial. Also, in the case where the bound given by the MDS conjecture is almost met (when $k = 2^{s-1}$), Augot and Finiasz noted that all companion matrices found had some kind of symmetry: if the k-th power of $\mathsf{Companion}(1, c_1, c_2, \ldots, c_{k-1})$ is MDS, then $c_i = c_{k-i}$ for all $1 \leq i \leq \frac{k-1}{2}$.

2.1 An Interesting Example

One of the *symmetric* MDS matrices found by Augot and Finiasz [1] for $k = 8$ and $\mathbb{F}_q = \mathbb{F}_{16}$ is

$$C = \mathsf{Companion}(1, \alpha^3, \alpha^4, \alpha^{12}, \alpha^8, \alpha^{12}, \alpha^4, \alpha^3)$$

with $\alpha^4+\alpha+1 = 0$. As we will see later, there is a strong link between companion matrices and the associated polynomial, here

$$P_C(X) = 1 + \alpha^3 X + \alpha^4 X^2 + \alpha^{12} X^3 + \alpha^8 X^4 + \alpha^{12} X^5 + \alpha^4 X^6 + \alpha^3 X^7 + X^8.$$

In this example, this polynomial factors into terms of degree two:

$$P_C(X) = (1 + \alpha^2 X + X^2)(1 + \alpha^4 X + X^2)(1 + \alpha^8 X + X^2)(1 + \alpha^9 X + X^2),$$

meaning that $P_C(X)$ is split in a degree-2 extension of $\mathbb{F}_{16}$, the field $\mathbb{F}_{256}$.

If we now consider $P_C(X)$ in $\mathbb{F}_{256}[X]$, which we can, since $\mathbb{F}_{16}$ is a subfield of $\mathbb{F}_{256}$, and look for its roots in $\mathbb{F}_{256}$, we find that there are 8 roots in $\mathbb{F}_{256}$, which, for a certain primitive 255-th root of unity $\beta \in \mathbb{F}_{256}$, are

$$[\beta^5, \beta^6, \beta^7, \beta^8, \beta^9, \beta^{10}, \beta^{11}, \beta^{12}].$$

This indicates a strong connection with BCH codes that we will now study.

3 Cyclic Codes, BCH Codes, and Shortening

Before jumping to BCH codes, we must first note a few things that are true for any cyclic code and not only BCH codes. For more details on the definition and properties of cyclic codes, the reader can refer to [8].

3.1 A Systematic Representation of Cyclic Codes

An $[n, k]_q$ code is said to be cyclic if a cyclic shift of any element of the code remains in the code. For example, the code defined by the following generator matrix G over $\mathbb{F}_2$ is cyclic:

$$G = \begin{pmatrix} 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{pmatrix}.$$

A cyclic shift to the right of the last line of G gives $(1, 0, 0, 0, 1, 0, 1)$ which is the sum of the first, third and last lines of G, thus remains in the code: G indeed generates a cyclic code.

Cyclic codes can also be defined in terms of polynomials: $(1, 0, 1, 1, 0, 0, 0)$ corresponds to $1 + X^2 + X^3$ and a cyclic shift to the right is a multiplication by X modulo $X^n - 1$. This way, cyclic codes can be seen as ideals of $\mathbb{F}_q[X]/(X^n - 1)$,

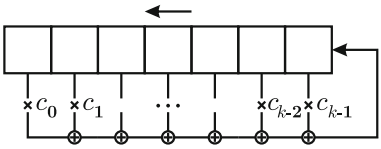


Fig. 1. An LFSR corresponding to the companion matrix C of polynomial $g(X) = X^k + c_{k-1}X^{k-1} + ... + c_0$. Clocking it k times is equivalent to applying C^k to its internal state.

meaning that each cyclic code Γ can be defined by a generator polynomial $g(X)$ such that $\Gamma =< g(X) >$ and $g(X)$ divides $X^n - 1$. Then, the code defined by $g(X)$ has dimension $k = n - \deg(g)$. In our example, $g(X) = 1 + X^2 + X^3$, which divides $X^7 - 1$, and the code is indeed of dimension 4.

Any multiple of $g(X)$ is in the code, so for any polynomial $P(X)$ of degree less than n, the polynomial $P(X) - (P(X) \bmod g(X))$ is in the code. Using this property with $P(X) = X^i$ for $i \in [\deg(g), n-1]$, we obtain an interesting systematic form for any cyclic code generator matrix:

$$G = \left(\begin{array}{c|cccc} -X^3 \bmod g(X) & 1 & 0 & 0 & 0 \\ -X^4 \bmod g(X) & 0 & 1 & 0 & 0 \\ -X^5 \bmod g(X) & 0 & 0 & 1 & 0 \\ -X^6 \bmod g(X) & 0 & 0 & 0 & 1 \end{array}\right) = \left(\begin{array}{ccc|cccc} 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 1 \end{array}\right).$$

This form is exactly what we are looking for when searching for powers of companion matrices. Indeed, if we associate the companion matrix $C = \mathsf{Companion}(c_0, \ldots, c_{k-1})$ to the polynomial $g(X) = X^k + c_{k-1}X^{k-1} + \cdots + c_0$, then the successive powers of C are (continuing with our example where $k = 3$):

$$C = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ -X^3 \bmod g(X) \end{pmatrix}, \; C^2 = \begin{pmatrix} 0 & 0 & 1 \\ -X^3 \bmod g(X) \\ -X^4 \bmod g(X) \end{pmatrix}, \; C^3 = \begin{pmatrix} -X^3 \bmod g(X) \\ -X^4 \bmod g(X) \\ -X^5 \bmod g(X) \end{pmatrix}.$$

To build recursive MDS matrices we thus simply need to build MDS cyclic codes with suitable parameters and their corresponding $g(X)$.

Note that a multiplication by a companion matrix can also be expressed in terms of LFSR. Initializing the LFSR of Fig. 1 with a vector and clocking it once corresponds to the multiplication of this vector by C. Clocking it k times corresponds to the multiplication by $M = C^k$. We will continue using the matrix representation in the rest of the paper, but most results could also be expressed in terms of LFSR.

3.2 BCH Codes and Shortened BCH Codes

In general, given a generator polynomial $g(X)$, computing the minimal distance of the associated cyclic code is a hard problem. For instance, the code generated by $g(X) = 1 + X^2 + X^3$ in the example of the previous section has minimal

distance 3, but even for such small examples it is not necessarily immediate to find the minimum distance. Nonetheless, lower bounds exist for some specific constructions. This is the case for BCH codes, as described for example in [8].

Definition 1 (BCH codes). *A BCH code over $\mathbb{F}_q$ is defined using an element β in some extension $\mathbb{F}_{q^m}$ of $\mathbb{F}_q$. First, pick integers ℓ and d and take the $(d-1)$ consecutive powers $\beta^\ell, \beta^{\ell+1}, \ldots, \beta^{\ell+d-2}$ of β, then compute $g(X) = \mathrm{lcm}(\mathrm{Min}_{\mathbb{F}_q}(\beta^\ell), \ldots, \mathrm{Min}_{\mathbb{F}_q}(\beta^{\ell+d-2}))$, where $\mathrm{Min}_{\mathbb{F}_q}(\beta^\ell)$ is the minimal polynomial of β^ℓ over $\mathbb{F}_q$.*

The cyclic code over $\mathbb{F}_q$ of length $\mathrm{ord}(\beta)$ defined by $g(X)$ is called a BCH code*, it has dimension $(\mathrm{ord}(\beta) - \deg(g))$ and has minimal distance at least d. We write this as being an $[\mathrm{ord}(\beta), \mathrm{ord}(\beta) - \deg(g), \geq d]_q$ code.*

For such a BCH code to be MDS, $g(X)$ must have degree exactly $d-1$ (for a cyclic code $\deg(g(X)) = n - k$ and for an MDS code $d = n - k + 1$, so an MDS BCH code necessarily verifies $\deg(g(X)) = d - 1$). Seeing that $g(X)$ already has $d-1$ roots over $\mathbb{F}_{q^m}$, it cannot have any other roots. This means that the powers $\beta^{\ell+j}$, $j = 0, \ldots, d-2$, must all be conjugates of each other.

The Need for Shortening. When building diffusion layers, the input and output of the diffusion generally have the same size (otherwise inversion might be a problem), so we need codes of length $2k$ and dimension k. In terms of BCH codes, this translates into using k consecutive powers of an element β of order $2k$, and having $g(X)$ of degree k. Of course, elements of even order do not exist in extensions of $\mathbb{F}_2$, so this is not possible. Instead of using full length BCH codes, we thus use *shortened* BCH codes.

Definition 2 (Shortened code). *Given a $[n, k, d]_q$ code Γ, and a set I of z indices $\{i_1, \ldots, i_z\}$, the shortened code Γ_I of C at indices from I is the set of words from Γ which are zero at positions $i_1, \ldots, i_z$, and whose zero coordinates are deleted, thus effectively shortening these words by z positions. The shortened code Γ_I has length $n - z$, dimension $\geq k - z$ and minimal distance $\geq d$.*

If Γ is MDS, then $d = n-k+1$ and Γ_I will necessarily be an $[n-z, k-z, d]_q$ MDS code, as neither the dimension nor the minimal distance can increase without breaking the Singleton bound [10].

We can thus look for $[2k + z, k + z, k + 1]_q$ BCH codes and shorten them on z positions to obtain our MDS codes. However, shortened BCH codes are no longer cyclic, so the shortening has to be done in a way that conserves the recursive structure. This is easy to achieve by using the previous systematic representation and shortening on the last positions. Starting from $g(X)$ of degree k, which divides $X^{2k+z} - 1$, we get a generating matrix:

$$G = \left(\begin{array}{c|c} X^k \bmod g(X) & 1\ 0\ 0\ 0 \\ X^{k+1} \bmod g(X) & 0\ 1\ 0\ 0 \\ \cdots & \cdots \\ X^{2k+z-1} \bmod g(X) & \underbrace{0\ 0\ 0\ 1}_{\text{size } k+z} \end{array}\right).$$

Shortening the code on the z last positions will maintain the systematic form and simply remove the z last lines to obtain:

$$G_I = \left(\begin{array}{c|cccc} X^k \bmod g(X) & 1 & 0 & 0 & 0 \\ X^{k+1} \bmod g(X) & 0 & 1 & 0 & 0 \\ \cdots & & \cdots & & \\ X^{2k-1} \bmod g(X) & 0 & 0 & 0 & 1 \end{array}\right) \underbrace{\qquad\qquad}_{\text{size } k}.$$

As said above, when G generates an MDS code, then G_I also generates an MDS code, and this is (up to a permutation of the two $k \times k$ blocks, that will not affect the MDS property) exactly what we are looking for: a recursive MDS matrix defined by the companion matrix associated to the polynomial $g(X)$.

3.3 Direct Construction of Recursive MDS Matrices

From this result, in the case where $q = 2^s$, we can deduce a direct construction of recursive MDS matrices based on MDS BCH codes that were already described in [8], Chap. 11, Sect. 5. We first pick a β of order $q+1$. As $q+1$ divides q^2-1, β is always in $\mathbb{F}_{q^2}$, the degree-2 extension of $\mathbb{F}_q$. Then, apart from $\beta^0 = 1$, all powers of β have minimal polynomials of degree 2: since β is of order $q+1$, each β^i has a conjugate $\beta^{qi} = \beta^{-i}$ which is the second root of $\mathrm{Min}_{\mathbb{F}_q}(\beta^i)$. From there, it is easy to build a $[q+1, q+1-k, k+1]_q$ MDS BCH code for any value of $k \leq \frac{q}{2}$.

- If k is even, we need to select k consecutive powers of β that are conjugates by pairs: if β^i is selected, β^{q+1-i} is selected too. We thus select all the powers β^i with $i \in [\frac{q-k}{2}+1, \frac{q+k}{2}]$, grouped around $\frac{q+1}{2}$.
- If k is odd, we need to select β^0 as well. We thus select all the powers β^i with $i \in [-\frac{k-1}{2}, \frac{k-1}{2}]$, grouped around 0.

In both cases, we get a polynomial $g(X)$ of degree k defining an MDS BCH code of length $q+1$. We can then shorten this code on $z = (q+1-2k)$ positions and obtain the $[2k, k, k+1]_q$ MDS code we were looking for. The non-systematic part of the generator matrix of this code is the k-th power of the companion matrix defined by $g(X)$.

Also, as the conjugate of β^i is its inverse, $g(X)$ enjoys the same symmetry as the example of Sect. 2.1: $X^k g(X^{-1}) = g(X)$. This explains the symmetry observed in [1]. Furthermore, the companion matrix associated to $g(X)$ thus has at most $\frac{k}{2}$ different coefficients and can be implemented with at most $\frac{k}{2}$ multiplications.

Finally, by cycling over all β of order $q+1$, in the case where $2k = q$ we were able to recover with this direct construction all the solutions found in [1] through exhaustive search. We conjecture that when $2k = q$, the only recursive MDS matrices that exist come from these shortened BCH codes.

4 An Algorithm to Find All MDS BCH Codes

We have seen that shortened BCH codes allow to directly build recursive MDS matrices. However, when building a block cipher, the designer usually has some parameters in mind (say, a diffusion layer on k symbols of s bits each) and wants the *best* diffusion layer matching these parameters. Our direct construction gives good solutions, but cannot guarantee they are the best. So the designer needs an algorithm that will enumerate all possible matrices and let him pick the most suitable one. For this, we will consider BCH codes where β is a $(2k+z)$-th root of unity and not only a $(2k+1)$-th root of unity as in the direct construction. First, there are a few constraints to consider.

Field Multiplication or $\mathbb{F}_2$-linearity? The designer has to choose the type of linearity he wants for his diffusion layer. If he wants (standard) linearity over $\mathbb{F}_{2^s}$, then the BCH code has to be built over $\mathbb{F}_{2^s}$ (or a subfield of $\mathbb{F}_{2^s}$, but the construction is the same). However, as in the Sajadieh *et al.* [9] or the Wu *et al.* [11] constructions, he could choose to use an $\mathbb{F}_2$-linear operator L. Assuming L has an irreducible minimal polynomial of degree $s' \leq s$ (see [1] for how to deal with non-irreducible minimal polynomials), then he needs to build a BCH code over $\mathbb{F}_{2^{s'}}$. This choice is up to the designer but does not change anything to the rest of the algorithm, so we will assume $s' = s$.

The MDS Conjecture. Our shortened BCH construction starts by building an MDS code of length $2k+z$ over $\mathbb{F}_{2^s}$. The MDS conjecture tells us that $2k+z \leq 2^s+1$ must hold. When $k = 2^{s-1}$, $z = 1$ is the only choice. In general, we can choose any $z \in [1, 2^s+1-2k]$, so the algorithm will need to try all these possibilities.

Minimal Polynomials of Roots of Unity. The β involved in the BCH construction is a $(2k+z)$-th root of unity, and $g(X)$ is formed as the product of minimal polynomials of powers of β. First, $(2k+z)$-th roots of unity must exist, meaning $2k+z$ must be odd (or more generally coprime with q when q is not 2^s). Then, when factorizing $X^{2k+z}-1$, the minimal polynomials of the β^i are factors of this decomposition, and $g(X)$ is the product of some of these factors. It must thus be possible to obtain a polynomial of degree k this way. This is not always possible: for example, $X^{23}-1$ decomposes over $\mathbb{F}_{2^8}$ in a factor of degree 1 and two factors of degree 11 and very few values of k can be obtained. However, this last condition is rather complex to integrate in an algorithm and it will be easier to simply not take it into account.

4.1 A Simple Algorithm

For given parameters k and $q = 2^s$ we propose to use Algorithm 1 (Fig. 2) to enumerate all possible recursive MDS matrices coming from shortened BCH codes. This algorithm explores all code lengths from $2k+1$ to $q+1$, meaning that the number of shortened columns can be much larger than the final code we are

aiming for. Instead of computing minimal polynomials and their least common multiple as in the definition of BCH codes we directly compute $\prod_{j=0}^{k-1}(X-\beta^{\ell+j})$ and check if it is in $\mathbb{F}_q[X]$. This allows the algorithm to be more efficient and also makes upper bounding its complexity much easier. The following lemma shows that the two formulations are equivalent.

Lemma 1. *A BCH code over $\mathbb{F}_q$ defined by the $d-1$ roots $[\beta^\ell, ..., \beta^{\ell+d-2}]$ is MDS, if and only if $P(X) = \prod_{j=0}^{d-2}(X-\beta^{\ell+j})$ is in $\mathbb{F}_q[X]$. In this case, $g(X) = \mathrm{lcm}\big(\mathrm{Min}_{\mathbb{F}_q}(\beta^\ell), ..., \mathrm{Min}_{\mathbb{F}_q}(\beta^{\ell+d-2})\big)$ is equal to $P(X)$.*

Proof. We have seen that a BCH code is MDS if and only if $g(X)$ is of degree $d-1$ exactly. Also, $g(X)$ is always a multiple of $P(X)$.

First, assume we have an MDS BCH code. Then $g(X)$ is of degree $d-1$ and is a multiple of $P(X)$ which is also of degree $d-1$. So, up to a scalar factor, $g(X) = P(X)$ and $P(X) \in \mathbb{F}_q[X]$.

Conversly, assume we have a BCH code such that $P(X) \in \mathbb{F}_q[X]$. Then, for any $j \in [0, d-2]$, $P(X)$ is a polynomial in $\mathbb{F}_q[X]$ having $\beta^{\ell+j}$ as a root, so $P(X)$ is a multiple of $\mathrm{Min}_{\mathbb{F}_q}(\beta^{\ell+j})$. Therefore, $g(X)$ divides $P(X)$ and, as $P(X)$ also divides $g(X)$, we have $g(X) = P(X)$. $g(X)$ thus has degree $d-1$ and the code is MDS. □

4.2 Complexity

The previous algorithm simply tests all possible candidates without trying to be smart about which could be eliminated faster. It also finds each solution several times (typically for β and β^{-1}), and finds some *equivalent* solutions (applying $\alpha \mapsto \alpha^2$ on all coefficients of the polynomial preserves the MDS property, so each equivalence class is found s times).

The product at line 7 does not have to be fully recomputed for each value of ℓ. It can be computed once for $\ell = 0$, then one division by $(X-\beta^\ell)$ and one multiplication by $(X-\beta^{\ell+k})$ are enough to update it at each iteration. This update costs $O(k)$ operations in the extension of $\mathbb{F}_q$ containing α. The whole loop on ℓ can thus be executed in $O((2k+z)k)$ operations in the extension field.

The number of β for which the loop has to be done is Euler's phi function $\varphi(2k+z)$ which is smaller than $(2k+z)$, itself smaller than q, and there are $\frac{q-2k}{2}+1$ values of z to test. This gives an overall complexity of $O(q^2k(q-2k))$ operations in an extension of $\mathbb{F}_q$. This extension is of degree at most $2k+z$, so operations are at most on $q\log q$ bits in this extension and cost at most $O(q^2(\log q)^2)$. This gives an upper bound on the total complexity of $O\big(q^4k(q-2k)(\log q)^2\big)$ binary operations, a degree-6 polynomial in k and q. This is a quite expensive, but as we will see in the next section, this algorithms runs fast enough for most practical parameters. It should also be possible to accelerate this algorithm using more elaborate computer algebra techniques.

Algorithm 1. Search for Recursive MDS Matrices

Input: parameters k and s
Output: a set $\mathcal{S}$ of polynomials yielding MDS matrices

$q \leftarrow 2^s$
$\mathcal{S} \leftarrow \emptyset$
for $z \leftarrow 1$ **to** $(q + 1 - 2k)$, **with** z *odd* **do**
  $\alpha \leftarrow$ primitive $(2k + z)$-th root of unity of $\mathbb{F}_q$
  forall the $\beta = \alpha^i$ **such that** $\mathrm{ord}(\beta) = 2k + z$ **do**
    for $\ell \leftarrow 0$ **to** $(2k + z - 2)$ **do**
      $g(X) \leftarrow \prod_{j=0}^{k-1}(X - \beta^{\ell+j})$
      if $g(X) \in \mathbb{F}_q[X]$ **then** *(we test if g has all its coefficients in $\mathbb{F}_q$)*
        $\mathcal{S} \leftarrow \mathcal{S} \cup \{g(X)\}$
      end
    end
  end
end
return $\mathcal{S}$

Fig. 2. Algorithm searching for MDS BCH codes

5 Experimental Results

We implemented Algorithm 1 in Magma [2] (see the code in Appendix A) and ran it for various parameters.

5.1 The Extremal Case: $2k = 2^s$

First, we ran the algorithm for parameters on the bound given by the MDS conjecture, that is, when $2k = 2^s$. These are the parameters that were studied by Augot and Finiasz in [1]. It took their algorithm 80 days of CPU time to perform the exhaustive search with parameters $k = 16$ and $s = 5$ and find the same 10 solutions that our new algorithm finds in a few milliseconds. The timings and number of solutions we obtained are summarized in Table 1. We were also able to find much larger MDS diffusion layers. For example, we could deal with $k = 128$ elements of $s = 8$ bits, which maybe is probably too large to be practical, even with a recursive structure and the nice symmetry. Below are the logs in base α (with $\alpha^8 + \alpha^4 + \alpha^3 + \alpha^2 + 1 = 0$) of the last line of the companion matrix of an example of such 1024-bit diffusion:

$$[0, 83, 25, 136, 62, 8, 73, 112, 253, 110, 246, 156, 53, 1, 41, 73, 5, 93, 190, 253, 149,$$
$$98, 125, 124, 149, 94, 100, 41, 37, 183, 81, 6, 242, 74, 252, 104, 57, 117, 55, 224,$$
$$153, 130, 77, 156, 192, 176, 52, 133, 218, 59, 158, 18, 228, 89, 218, 126, 146,$$
$$210, 217, 18, 84, 209, 30, 123, \mathbf{97}, 123, \ldots \text{ [symmetric] } \ldots, 83]$$

Table 1. Experimental results for parameters on the bound given by MDS conjecture. The value "Diff. bits" is the size in bits of the corresponding diffusion layer. The number of solutions is given as both the raw number and the number of distinct equivalence classes.

k	s	Diff. bits	Solutions		Time
			Num.	Classes	
4	3	12	3	1	<0.01 s
8	4	32	8	2	<0.01 s
16	5	80	10	2	<0.01 s
32	6	192	24	4	∼0.02 s
64	7	448	42	6	∼0.07 s
128	8	1024	128	16	∼0.52 s
256	9	2304	162	18	∼1.71 s

Table 2. Experimental results for other interesting parameters. The Reg. solutions refer to regular solutions where the constant term of the polynomial is 1.

k	s	Diff. bits	Solutions		Time
			Num.	Reg.	
4	4	16	68	12	∼0.02 s
4	8	32	20180	252	∼37 s
8	8	64	20120	248	∼44 s
16	8	128	19984	240	∼55 s
32	8	256	19168	224	∼80 s

5.2 The General Case

We also ran some computations for other interesting parameters, typically for values of k and s that are both powers of 2 as it is often the case in block ciphers. The results we obtained are summarized in Table 2. Note that for these solutions the number of shortened positions is sometime huge: for $k = 4$ and $s = 8$ one can start from a $[257, 253, 5]_{256}$ BCH code and shorten it on 249 positions to obtain a $[8, 4, 5]_{256}$ code. We counted both the total number of solutions we found and the number of regular solutions where the constant term of the polynomial is 1. Regular solutions are particularly interesting as the diffusion and its inverse share the same coefficients:

$$\mathsf{Companion}(1, c_1, \ldots, c_{k-1})^{-1} = \begin{pmatrix} 0 & 1 & & 0 \\ & & \ddots & \\ 0 & 0 & & 1 \\ 1 & c_1 & & c_{k-1} \end{pmatrix}^{-1} = \begin{pmatrix} c_1 & & c_{k-1} & 1 \\ 1 & & 0 & 0 \\ & \ddots & & \\ 0 & & 1 & 0 \end{pmatrix}. \quad (1)$$

In the case of symmetric solutions (like those from Sect. 3.3), encryption and decryption can even use the exact same circuit by simply reversing the order of the input and output symbols. Here are some examples of what we found:

- for parameters $k = 4$ and $s = 4$, with α such that $\alpha^4 + \alpha + 1 = 0$, the matrices $\mathsf{Companion}(1, \alpha^3, \alpha, \alpha^3)^4$ and $\mathsf{Companion}(\alpha^3 + \alpha, 1, \alpha, \alpha^3)^4$ are MDS.
- for parameters $k = 4$ and $s = 8$, with α such that $\alpha^8 + \alpha^4 + \alpha^3 + \alpha^2 + 1 = 0$, the matrices $\mathsf{Companion}(1, \alpha^3, \alpha^{-1}, \alpha^3)^4$, $\mathsf{Companion}(1, \alpha^3 + \alpha^2, \alpha^3, \alpha^3 + \alpha^2)^4$, and $\mathsf{Companion}(\alpha + 1, 1, \alpha^{202} + 1, \alpha^{202})^4$ are MDS.

The reader might note the absence of larger fields in Table 2. One could for example want to obtain a 128-bit diffusion layer using $k = 8$ symbols of $s = 16$ bits. However, going through all the possible values of z and ℓ takes too long with $q = 2^{16}$. Our algorithm is too naive, and an algorithm enumerating the divisors of $X^{2k+z} - 1$ of degree k and checking if they correspond to BCH codes could be faster in this case. Otherwise, it is always possible to use the direct construction given in Sect. 3.3.

5.3 Further Work

As we have seen, for most parameters, this algorithm runs fast enough to find all recursive MDS matrices coming from BCH codes. However, not all recursive MDS matrices come from a BCH code.

- First, there are other classes of cyclic codes that are MDS and could be shortened in a similar way. Any such class of codes can directly be plugged into our algorithm, searching for polynomials $g(X)$ having another structure than roots that are consecutive powers of β.
- Then, there also are cyclic codes which are not MDS, but become MDS once they are shortened. These will be much harder to track as they do not have to obey the MDS conjecture and can have a much larger length before shortening.

For this reason, we are not always able (yet) to find the most efficient matrices with our algorithm. For example, the matrix used in Photon corresponds to a cyclic code of length $2^{24} - 1$ over $\mathbb{F}_{2^8}$ which is not MDS. We know that this code has minimum distance 3, and its distance grows to 5 when shortened from the length $2^{24} - 1$ to the length 8.

However, for some parameters, our algorithm is able to find very nice solutions. For $k = 4$ and α verifying $\alpha^5 + \alpha^2 + 1 = 0$ (a primitive element of $\mathbb{F}_{2^5}$, or an $\mathbb{F}_2$-linear operator with this minimal polynomial), the matrix $\mathsf{Companion}(1, \alpha, \alpha^{-1}, \alpha)$ found by Algorithm 1 yields an MDS diffusion layer. This is especially nice because it is possible to build simple $\mathbb{F}_2$-linear operators that also have a simple inverse, and this solution is symmetric meaning the inverse diffusion can use the same circuit as the diffusion itself.

6 Conclusion

The main result of this article is the understanding that recursive MDS matrices can be obtained directly from shortened MDS cyclic codes. From this, we derive both a direct construction and a very simple algorithm, based on the enumeration of BCH codes, that allows to efficiently find recursive MDS matrices for any diffusion and symbol sizes. These constructions do not always find all existing recursive MDS matrices and can thus miss some interesting solutions. As part of our future works, we will continue to investigate this problem, trying to understand what properties the other solutions have and how we can extend our algorithm to find them all. A first step is to elucidate the Photon matrix in terms of cyclic codes which are not BCH codes, hopefully finding a direct construction of this matrix. However, in the same way as computing the minimal distance of a cyclic code is difficult, it might turn out that finding all recursive MDS matrices of a given size is a hard problem.

References

1. Augot, D., Finiasz, M.: Exhaustive search for small dimension recursive MDS diffusion layers for block ciphers and hash functions. In: 2013 IEEE International Symposium on Information Theory Proceedings (ISIT), pp. 1551–1555. IEEE (2013)
2. Bosma, W., Cannon, J., Playoust, C.: The Magma algebra system. I. The user language. J. Symb. Comput. **24**(3–4), 235–265 (1997)
3. Daemen, J.: Cipher and hash function design, strategies based on linear and differential cryptanalysis, Ph.D. thesis. K.U. Leuven (1995)
4. Daemen, J., Rijmen, V.: The Design of Rijndael. Information Security and Cryptography. Springer, Heidelberg (2002)
5. Guo, J., Peyrin, T., Poschmann, A.: The PHOTON family of lightweight hash functions. In: Rogaway, P. (ed.) CRYPTO 2011. LNCS, vol. 6841, pp. 222–239. Springer, Heidelberg (2011)
6. Guo, J., Peyrin, T., Poschmann, A., Robshaw, M.: The LED block cipher. In: Preneel, B., Takagi, T. (eds.) CHES 2011. LNCS, vol. 6917, pp. 326–341. Springer, Heidelberg (2011)
7. Junod, P., Vaudenay, S.: FOX: a new family of block ciphers. In: Handschuh, H., Hasan, M.A. (eds.) SAC 2004. LNCS, vol. 3357, pp. 114–129. Springer, Heidelberg (2004)
8. MacWilliams, F.J., Sloane, N.J.A.: The Theory of Error Correcting Codes. North-Holland Mathematical Library. North-Holland, Amsterdam (1978)
9. Sajadieh, M., Dakhilalian, M., Mala, H., Sepehrdad, P.: Recursive diffusion layers for block ciphers and hash functions. In: Canteaut, A. (ed.) FSE 2012. LNCS, vol. 7549, pp. 385–401. Springer, Heidelberg (2012)
10. Singleton, R.: Maximum distance q-nary codes. IEEE Trans. Inf. Theor. **10**(2), 116–118 (1964)
11. Wu, S., Wang, M., Wu, W.: Recursive diffusion layers for (lightweight) block ciphers and hash functions. In: Knudsen, L.R., Wu, H. (eds.) SAC 2012. LNCS, vol. 7707, pp. 355–371. Springer, Heidelberg (2013)

A Magma Code

Here is the Magma code for Algorithm 1. Simply run `BCH(k,s)` to get the set of all polynomials of degree k over $\mathbb{F}_{2^s}$ that yield MDS diffusion layers on ks bits. Of course, these polynomials have to be written as companion matrices which then have to be raised to the power k to obtain the final MDS matrices.

```
BCH := function(k,s)
  q := 2^s;
  F := GF(q);
  P := PolynomialRing(F);
  S := { };
  for z:=1 to q+1-2*k by 2 do
    a := RootOfUnity(2*k+z, F);
    Pext<X> := PolynomialRing(Parent(a));
    for i:=0 to 2*k+z-1 do
      b := a^i;
      if Order(b) eq (2*k+z) then
        g := &*[(X-b^l): l in [-1..k-2]];
        for l in [0..2*k+z-2] do
          g := (g*(X-b^(l+k-1))) div (X-b^(l-1));
          if IsCoercible(P,g) then
            Include(~S, P!g);
          end if;
        end for;
      end if;
    end for;
  end for;
  return S;
end function;
```

LS-Designs: Bitslice Encryption for Efficient Masked Software Implementations

Vincent Grosso[1], Gaëtan Leurent[1,2], François-Xavier Standaert[1(✉)], and Kerem Varıcı[1]

[1] ICTEAM/ELEN/Crypto Group, Université catholique de Louvain, Louvain-la-Neuve, Belgium
fstandae@uclouvain.be
[2] Inria, EPI SECRET, Rocquencourt, France

Abstract. Side-channel analysis is an important issue for the security of embedded cryptographic devices, and masking is one of the most investigated solutions to mitigate such attacks. In this context, efficient masking has recently been considered as a possible criteria for new block cipher designs. Previous proposals in this direction were applicable to different types of masking schemes (e.g. Boolean and polynomial). In this paper, we study possible optimizations when specializing the designs to Boolean masking. For this purpose, we first observe that bitslice ciphers have interesting properties for improving both the efficiency and the regularity of masked software implementations. Next we specify a family of block ciphers (denoted as LS-designs) that can systematically take advantage of bitslicing in a principled manner. Eventually, we evaluate both the security and performance of such designs and two of their instances, confirming excellent properties for physically secure applications.

1 Introduction

Lightweight cryptography has been an active research area over the last 10 years. Many innovative ciphers have been proposed in order to optimize various performance criteria. Recently, resistance against side-channel attacks has been considered as an additional optimization goal for low-cost ciphers, as exemplified by the algorithms PICARO [41] and `Zorro` [20]. Both proposals aim at leading to efficient *masked* implementations (i.e. where all the computations are performed on shared secrets). Starting from the observation that the performance overheads in such implementations primarily come from non-linear operations, Piret et al. [41] first investigated how to reduce their amount by considering non-bijective S-boxes. Next, Gérard et al. [20] further exploited "irregular" SPN structures, i.e. where not all the state goes through the (bijective again[1]) S-boxes in each round. Both examples lead to performance gains over the AES Rijndael, which become more significant as the number of shares increases.

[1] Motivated by the recent results in [51], showing that non-bijective S-boxes lead to easily exploitable targets for generic (non-profiled) Differential Power Analysis.

C. Cid and C. Rechberger (Eds.): FSE 2014, LNCS 8540, pp. 18–37, 2015.
DOI: 10.1007/978-3-662-46706-0_2

These previous works lead to several useful observations regarding the relation between masking and the linear/non-linear operations used in block ciphers. In this paper, we aim to complement them by focusing on two important scopes for further research they left open. First from the performance point-of-view, both PICARO and `Zorro` minimize the number of field multiplications per encrypted plaintext. This is a natural direction as it leads to improvements applicable to both Boolean [45] and polynomial [43] masking schemes. Yet, further specialization to Boolean masking could potentially lead to additional gains. For example, the S-boxes of lightweight ciphers PRESENT [7] and NOEKEON [15] require three multiplications in $GF(16)$, which makes them less suitable than `Zorro` and PICARO for polynomial masking. But they have efficient representations minimizing the number of AND gates which could be exploited in Boolean masked implementations. Next from the security point-of-view, both designs are based on somewhat unusual Feistel/SPN structures (in order to deal with non-bijective S-boxes in [41], and to minimize the number of S-boxes per round in [20]). So another (quite pragmatic) open question is whether we can design ciphers for efficient masking based on more standard techniques (e.g. directly exploiting the wide-trail strategy [16] as other lightweight algorithms).

In this context, we base our investigations on two additional observations. First, Boolean masking is particularly efficient when applied to operations that are linear over $GF(2)$ (since such operations can be performed independently on each share). As a result, and in contrast with many existing block ciphers, it appears interesting to have linear diffusion layers implemented as look-up tables, since they can be straightforwardly exploited for any number of shares[2]. Second, since our focus is on software implementations, we also have a strong incentive for simple and regular designs, where computations are always performed on well aligned data. For example, manipulating bits and bytes such as in PRESENT raises additional challenges for the implementers (to guarantee that the bit manipulations do not leak more information than the byte ones). These observations combine into the conclusion that a bitslice cipher with look-up table-based diffusion layers and non-linear S-boxes with efficient gate-level representation seems an excellent candidate for efficient Boolean masked software implementations.

Following, our contributions are threefold. First, we separately analyze S-boxes and linear layers meeting the previous objectives, and compare a number of constructions from the cryptanalytic and efficient masking point-of-view. Interestingly, this part of our study confirms the previous observation that if side-channel resistance via masking is added as a block cipher design criteria, the balance between linear and non-linear operations has to be changed towards more linear ones. Such investigations open a large space of possible ciphers that we define as LS-designs (essentially made of a combination of look-up table-based L-boxes and bitslice S-boxes). We then argue that such designs have interesting properties for efficient masking. For this purpose and for concreteness, we specify two instances of 128-bit block ciphers and analyze their security against a number of standard cryptanalytic techniques. Doing so, we paid attention to make

[2] Masking non-linear look-up tables has a cost that is quadratic in this number [11].

our studies as generic as possible (i.e. leading to conclusions for LS-designs rather than for their instances). We also considered the impact of choosing involutive or non-involutive components, and show that the first ones mainly lead the security guarantees provided by the wide-trail strategy to be tighter. Eventually, we compare the performances of our two exemplary instances with the ones of the AES, PICARO, `Zorro` and NOEKEON on an 8-bit microcontroller (and argue that they also behave well on desktop CPUs with SIMD units). Overall, these results confirm the interest of bitslice ciphers in the context of physically secure implementations, as hinted in [14]. While the instances of designs we suggest are not yet optimal (because of the hardness of finding optimal L- and S-boxes) and do require more analysis, their performances are comparable to (or slightly better than) the ones of NOEKEON, which is known to have an extremely compact gate-level representation [29]. Therefore, we believe LS-designs formalize an interesting family of ciphers combining excellent performances (also for unprotected implementations), strong security guaranties, regularity/simplicity and efficient masking, i.e. properties that generally benefit to resistance against side-channel attacks and are also appealing for more general applications.

2 Design Rationale

2.1 Bitslice S-Boxes

In this first subsection, we analyze various S-boxes having an efficient bitslice representation. Our comparisons will consider various sizes (namely 4-bit, 8-bit and 16-bit), in order to study their tradeoff with the different diffusion layers in the next subsection. They will also take into account both standard cryptographic properties (such as the non-linearity, differential profile and algebraic degree of which the definitions are recalled in Appendix A) and masking efficiency considerations. For this purpose, and following the techniques in [26], we will simply consider the number of AND and XOR gates needed for each S-box. As already mentioned, the cost of XOR gates is linear in the number of shares in the masking scheme, while it is quadratic for AND gates (which will therefore count as a more important criterion in our evaluations). Note that since we are only interested in non-linear S-boxes, each Boolean function defining them has to be linearly independent of the other ones. As a result, we need at least the same number of AND gates as the output size of the S-box to reach this goal.

Why Bitslicing? In Appendix B, we recall the method to perform secure non-linear operations first proposed in [26] and generalized to extension fields in [45]. From Algorithm 2, it is easy to see that the difference of performances between Boolean and polynomial masking can (at least partially) be explained by the implementation efficiency of the underlying non-linear operation. For example, an AND gate can usually be performed in a single clock cycle on most computing devices. By contrast, a field multiplication generally requires the use of log/alog tables (if large fields are considered) which will typically account for 20 to 40 clock cycles in embedded microcontrollers [22]. These numbers can be improved

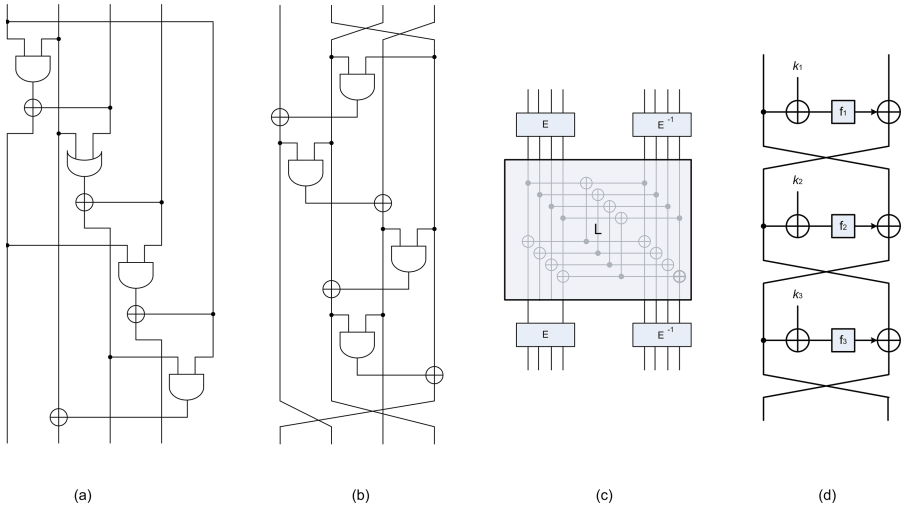


Fig. 1. (a) Non involutive 4-bit S-box with optimal bitslice representation. (b) Involutive 4-bit S-box with optimal bitslice representation. (c) Construction of 8-bit S-boxes from 4-bit ones as in the Whirlpool hash function [44]. (d) Construction of $2s$-bit S-boxes from s-bit ones as in the MISTY block cipher [36].

when small fields are considered, e.g. multiplication can tabulated if elements are represented with less than 4 bits, but even in this case the non-linear operations will require 3 to 5 cycles. As a result, masking at the gate-level in a bitslice manner as we investigate next should bring performance improvements.

We now present a couple of S-boxes with efficient gate-level representation. The main challenge is that the enumeration of S-boxes is rapidly out of reach as their size increases. Besides, finding the best gate-level description of a large S-box is also a hard problem. As a result, we will start from the 4-bit case for which exhaustive analysis is possible, and then take advantage of heuristics from the block cipher literature, in order to turn these 4-bit S-box into larger ones.

4-Bit S-Boxes. An exhaustive search of optimal bitslice S-boxes can be found in [48]. Its main result is that the so-called "Class 13" is the best option for this purpose, and can be implemented with 4 non-linear gates and a total of 9 instructions, with the best differential and linear probabilities that can be reached. It is represented in Fig. 1(a). Its only limitation is that it is not involutive. We ran a similar exhaustive search with a slightly larger instruction set (including nor gates, nand gates, and more copy instructions) while restricting the number of non-linear gates to 4. As a result, we could find an involutive S-box with similar properties as the Class 13 one: it is represented in Fig. 1(b). Note that the use of Toffoli gates (defined in [47]) allows to see this S-box as a generalized Feistel network, which explains the involution property.

Table 1. Comparison of our different S-box proposals.

	# AND	# XOR	size	Invol.	deg(S)	$\Pr_{\text{diff}}$	$\Pr_{\text{lin}}$
NOEKEON	4	7	4	Yes	3	2^{-2}	2^{-1}
Class 13 [48]	4	4	4	No	3	2^{-2}	2^{-1}
Figure 1(b)	4	4	4	Yes	3	2^{-2}	2^{-1}
AES [9]	32	83	8	No	7	2^{-6}	2^{-3}
Whirlpool + Class 13	16	41	8	No	6	$2^{-4.68}$	2^{-2}
Whirlpool + Fig. 1(b)	16	42	8	No	6	$2^{-4.68}$	2^{-2}
MISTY + Class13	12	24	8	Yes	6	2^{-4}	2^{-2}
MISTY + Fig. 1(b)	12	24	8	Yes	5	2^{-4}	2^{-2}
MISTY + 3/5-bit S-boxes	11	25	8	No	5	2^{-4}	2^{-2}
MISTY^2 + Class13	36	96	16	Yes	13	2^{-8}	2^{-4}

From 4-Bit S-Boxes to Larger Ones. Various constructions can be considered for this purpose, ranging from ad hoc (scaling 4-bit S-boxes to 8-bit ones) to generic solutions (scaling s-bit S-boxes to $2s$-bit ones). As in [20], we analyzed a couple of natural candidates and report on the most interesting results.

In the first (ad hoc) case, we considered a proposal coming from the Whirlpool hash function (which only requires four 4-bit S-boxes) and generalized it by considering a linear layer between the two levels of S-boxes (see Fig. 1(c)). Since our goal is to minimize the number of AND gates, such a solution was better than proposals with six 4-bit S-boxes as in the KHAZAD block cipher [1].

Alternatively, we looked at Feistel networks such as used in the MISTY block cipher [36] and represented in Fig. 1(d). A minimum of three rounds were considered in order to avoid trivial weaknesses with respect to linear and differential cryptanalyses. One advantage of such a construction is that it directly gives rise to involutive components. Besides, it has been shown that three rounds of such a network allow squaring the linear and differential probabilities of the round function, on average over the keys. Note however that the impact of this averaging only appears for 16-bit (or larger) S-boxes (i.e. when applying the MISTY structure twice, recursively), because the impact of the linear hull effect only becomes significant from this size on [40]. The exhaustive search over all the 16-bit S-boxes having the structure of Fig. 1(d) was too computationally intensive and we only report on the best candidate we found. Note that for 8-bit S-boxes, we additionally investigated unbalanced Feistel networks built from 3- and 5-bit ones (as also proposed with the MISTY cipher), which provided a slightly improved non-involutive candidate (with one less AND gate).

The results of our different S-box searches are summarized in Table 1. For comparison purposes, we also reported the same metrics for the NOEKEON S-box, and the bitslice representation of the AES S-box proposed in [9].

2.2 Table-Based Diffusion Layers

In a bitslice implementation of a block cipher, the same register i holds the i-th bit of several S-box inputs/outputs. In this context, we are interested in linear

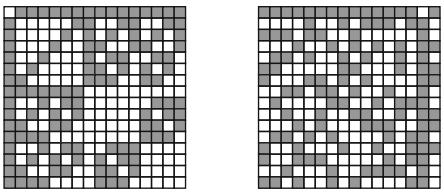
diffusion boxes (next denoted as L-boxes) that mix bits inside these registers and can be applied to them in parallel. From an implementation point of view, computing an L-box will just correspond to a table access. Yet, and compared to the usual case of non-linear tables, we benefit from more flexibility. Namely, since the table is linear, it can be decomposed into several smaller tables. This can be useful in order to only store tables that are adapted to the memory characteristics of the target platform. For instance, a 16-bit L-box can be implemented as four 8-bit to 8-bit look-up tables and two XORs. Our comparisons of diffusion layers will mainly be based on the branch number. Thanks to this number, we can directly evaluate the linear and differential properties of any design based on a combination of S-boxes and L-boxes according to the wide-trail strategy. We define the branch number of an L-box L as:

$$\mathcal{B}(\mathsf{L}) = \min_{x \neq 0} \big(|x| + |\mathsf{L}(x)|\big).$$

It follows that the linear branch number and the differential branch number of our diffusion layer is also $\mathcal{B}(\mathsf{L})$. This guarantees that any non-trivial trail in two consecutive cipher rounds will have at least $\mathcal{B}(\mathsf{L})$ active S-boxes (see [16, Theorem 1]). Note that an l-bit L-box with branch-number b is equivalent to a binary linear code with parameters $[2l, l, b]$ (length $2l$, dimension l, distance b). Therefore, we can use results from coding theory to design our L-boxes, such as [21].

8-bit L-boxes. The highest branch number possible for an 8-bit L-box is 5. We ran an exhaustive search and we found $2^{25.2}$ candidates with such a branch number, including 33 involutions. They activate at least 5/16 of the S-boxes.

16-bit L-boxes. The highest branch number possible for a 16-bit L-box is 8. It is not feasible to run an exhaustive search, but there are several known codes with parameters [16, 8, 8]. In particular, it is possible to build a quite structured 16-bit involution with branch number 8 from a systematic generator of the Reed-Muller code $RM(2, 5)$, as represented in the left part of the figure below (a non-involutive candidate is given on the right part of the same figure). This kind of L-boxes activates at least one fourth of the S-boxes over two rounds.



32-bit L-boxes. For a 32-bit L-box, the optimal branch number is not known. The best known code gives a branch number of 12, and the known upper bound shows that it is impossible to reach a branch number higher than 16. Therefore the best known option will only activate 12/64 of the S-boxes.

We show a comparison of the best known L-box diffusion layer and the AES diffusion layer in Table 2. This shows that L-box diffusion layers do not have

Table 2. Comparison of linear layers.

# S-boxes		Active S-boxes
8-bit L-box	8	5/16 (31.25 %)
16-bit L-box	16	8/32 (25 %)
32-bit L-box	32	12/64 (18.75 %)
AES linear layer	16	25/64 (39.06 %)

security bounds as good as AES-like diffusion layers (mainly because they are obtained over two rounds rather than four), but the ability to use a bitslice implementation is an important advantage for side-channel resistance.

Alternatively if the state is considered as a vector of elements over a larger field (the S-box outputs), the diffusion layer can be written as a binary matrix. This approach has been used previously, *e.g.* in the design of ARIA [33].

2.3 Which S-Box with Which L-Box?

The composition of s-bit S-boxes and l-bit L-boxes directly gives rise to various candidate $n = l \times s$-bit ciphers. In this subsection, we illustrate the tradeoffs resulting from these choices in a 64-bit case (with 8-bit and 16-bit L-boxes).

4-bit S-box and 16-bit L-box. Using the components described previously, the best involutive S-box requires 4 linear operations and 4 non-linear ones, and achieves $\Pr_{\text{diff}} = 2^{-2}$ and $\Pr_{\text{lin}} = 2^{-1}$. Therefore, we need at least 32 active S-boxes to have a secure cipher. Since we have 8 active S-boxes every 2 rounds using a 16-bit L-box, this corresponds to at least 8 rounds. In an 8-bit CPU, it would require 64 non-linear operations, 128 XORs and 128 table look-ups.

8-bit S-box and 8-bit L-box. Using the components described previously, the best involutive S-box requires 24 linear operations and 12 non-linear operations, and achieves $\Pr_{\text{diff}} = 2^{-4}$ and $\Pr_{\text{lin}} = 2^{-2}$. Therefore, we need at least 16 active S-boxes to have a secure cipher. Since we have 5 active S-boxes every 2 rounds using a 8-bit L-box, this corresponds to roughly 6 rounds. In an 8-bit CPU, it would require 72 non-linear operations, 144 XORs and 48 table look-ups.

Interestingly, we can see that the first option requires a total of 320 elementary operations, to be compared with only 264 ones for the second one. By contrast, the first option has a reduced number of non-linear operations, which will gradually dominate if a masked implementation with large number of shares is considered. While somewhat specific, this example confirms the trend already observed in [20] that the ratio between the amount of linear and non-linear operations increases in block ciphers that are easier to mask. Besides, it also shows that a small L-box can activate a larger proportion of the S-boxes, but these larger S-boxes are generally more expensive (if they are selected to have good cryptographic properties). In this 64-bit comparison, the two effects are of similar magnitude, but the conclusion is of course dependent on the block cipher size and on the knowledge we have about large bitslice S-boxes and L-boxes.

Algorithm 1. LS-design with l-bit L-boxes and s-bit S-boxes ($n = l \cdot s$)

$x \leftarrow P \oplus K$; ▷ x is a $s \times l$ bits matrix
for $0 \leq r < N_r$ **do**
  for $0 \leq i < l$ **do** ▷ S-box Layer
    $x[i, \star] = \mathsf{S}[x[i, \star]]$;
  end for
  for $0 \leq j < s$ **do** ▷ L-box Layer
    $x[\star, j] = \mathsf{L}[x[\star, j]]$;
  end for
  $x \leftarrow x \oplus K \oplus C(r)$; ▷ Key addition and round constant
end for
return x

3 LS-designs Specifications

Following the previous section, we can define LS-designs as the family of block ciphers specified in Algorithm 1. The description directly suggests simplicity and regularity as one important advantage of such ciphers: instances can be characterized by selecting a bitslice S-box S, an L-box L, a number of rounds N_r and constants $C(r)$. In the next sections, we will consider two 128-bit instances of LS-designs in order to illustrate their security against cryptanalysis and good implementation properties. The bit-size was chosen both because of the observations in [49] and in order to be comparable with NOEKEON (which is among the best ciphers published so far for efficient bitslice representation).

Involutive instance (Robin). We take the S-box denoted as "MISTY + Class13" from Table 1 and the involutive L-box in Sect. 2.2. The cipher has 16 rounds and constants are computed as $[\mathsf{L}(i)\ 0\ \ldots\ 0]$ with i the round index.

Non-involutive instance (Fantomas). We take the S-box denoted as "MISTY + 3/5-bit S-boxes" from Table 1 and the non-involutive L-box in Sect. 2.2. The cipher has 12 rounds and uses the same constants as Robin.

4 Security Evaluation

We now investigate the security properties of LS-designs, trying to extract general conclusions that apply to our family of ciphers in the first place. For concreteness, we will also consider more specific claims related to the aforementioned instances. In this respect, we note that Robin and Fantomas were specified with slightly different goals. Namely, the first one aims to have security margins similar to NOEKEON, while the second was mainly defined in order to illustrate the impact of choosing involutive components with respect to the efficiency limits that can be expected with LS-designs. Note that we aim for single-key security in both cases (i.e. exclude related-key and chosen key attacks from our claims). In particular, there is a simple related-differential with a single active S-box per round if the state difference can be corrected using a key difference.

4.1 Security Against Linear and Differential Cryptanalysis

As explained in Sect. 2.2, the structure of the cipher gives a simple upper bound on the maximum probability of differential characteristics, and the maximum bias of linear trails. Any two-round trail activates at least $\mathcal{B}(L)$ S-boxes, and this gives the following bounds for any $2r$-round trail:

$$\mathrm{Pr}_{\mathrm{lin}}(2r) \leq \mathrm{Pr}_{\mathrm{lin}}^{\max}(S)^{r \cdot \mathcal{B}(L)}, \qquad \mathrm{Pr}_{\mathrm{diff}}(2r) \leq \mathrm{Pr}_{\mathrm{diff}}^{\max}(S)^{r \cdot \mathcal{B}(L)}. \tag{1}$$

With the parameters of Sect. 3, this gives:

$$\mathrm{Pr}_{\mathrm{lin}}(2r) \leq 2^{-16 \cdot r}, \qquad \mathrm{Pr}_{\mathrm{diff}}(2r) \leq 2^{-32 \cdot r}.$$

Such bounds prevent simple linear and differential attacks based on a trail over more than 8 rounds. We use 16 (resp. 12) rounds in Robin (resp. Fantomas), so as to have a good (resp. less conservative) security margin. We now study the tightness of these bounds, and how to build optimal differential/linear trails.

Product Trails for Robin. To study differential and linear trails, we first consider a set of special states that can be written as the tensor product of an s-bit vector (corresponding to the S-box input and denoted with Greek letters) and an l-bit vector (corresponding to the L-box input and denoted with Latin letters):

$$\alpha \otimes x = \begin{bmatrix} \alpha_0 x_0 & \alpha_0 x_1 & \alpha_0 x_2 & \alpha_0 x_3 & \alpha_0 x_4 & \cdots & \alpha_0 x_l \\ \alpha_1 x_0 & \alpha_1 x_1 & \alpha_1 x_2 & \alpha_1 x_3 & \alpha_1 x_4 & & \alpha_1 x_l \\ \alpha_2 x_0 & \alpha_2 x_1 & \alpha_2 x_2 & \alpha_2 x_3 & \alpha_2 x_4 & \cdots & \alpha_2 x_l \\ \vdots & & & & \vdots & \ddots & \vdots \\ \alpha_s x_0 & \alpha_s x_1 & \alpha_s x_2 & \alpha_s x_3 & \alpha_s x_4 & \cdots & \alpha_s x_l \end{bmatrix}.$$

For those states, the S-box layer and the L-box layer act independently:

$$\mathsf{S}-\mathsf{layer}(\alpha \otimes x) = \mathsf{S}(\alpha) \otimes x, \qquad \mathsf{L}-\mathsf{layer}(\alpha \otimes x) = \alpha \otimes \mathsf{L}(x).$$

Hence we have the same behavior for differences and for linear masks. Namely, we can build differential characteristics (resp. linear trails) where the differences (resp. selection masks) are written as tensor products. In particular, if $\mathsf{L}[x] = y$, and $\alpha \rightsquigarrow \beta$ with probability p through the S-box, then $x \otimes \alpha \rightsquigarrow y \otimes \beta$ through one round with probability $p^{|x|}$, where $|x|$ denotes the Hamming weight of x.

When the cipher is built as an involution, we further have $\beta \rightsquigarrow \alpha$ with probability p, and $\mathsf{L}[y] = x$, hence $y \otimes \beta \rightsquigarrow x \otimes \alpha$ through one round with probability $p^{|y|}$, giving an iterated two-round trail (as illustrated with a toy example in Fig. 2). If α, β, x, and y are chosen optimally, this path reaches the security bound of Eq. (1), hence showing that the bound is tight. Using the parameters of Robin, optimal choices of α and β give $p = 2^{-4}$, while optimal choices of x and y give $|x| + |y| = 8$. This directly leads to a two-round iterated differential characteristic with probability 2^{-32} (or 2^{-128} for 8 rounds), and a two-round iterated linear trail with bias 2^{-16} (or 2^{-64} for 8 rounds).

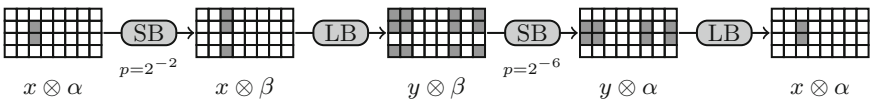


Fig. 2. Toy example of product trail for an involution-based 32-bit cipher, with $\alpha = \texttt{0110}$, $\beta = \texttt{1101}$, $x = \texttt{00100000}$, $y = \texttt{11000101}$, $\Pr[\alpha \rightsquigarrow \beta] = 2^{-2}$.

Truncated Differentials. Since the L-layer and the S-layer operate independently on product states, there are many trails with the same active S-boxes but with different inputs and outputs. As long as all the S-boxes in a round have the same output difference, the state is a product state, and all these differences contribute to the same truncated trail. The space of such product states is quite small and we can search exhaustively for the best truncated differential using product trails. We note that there could be better truncated trails but it seems that the product trails are dominant when we consider an involution.

For Robin, the best truncated trails start with a single active S-box, and have alternatively 1 and 7 active S-boxes. A round with a single active S-box has a probability one (to follow the truncated trail), and a round with 7 active S-boxes has a probability of $2^{-28.5}$ if the input difference is random (if it comes from a previous round, the probability is slightly skewed). The best such trails for 8 and 9 rounds have a probability of $2^{-112.1}$, while the best trails for 10 and 11 rounds have a probability of $2^{-139.8}$. These results can be used in a truncated differential attack as follows. If one takes a pair of states with a single active S-box, there will be a single active S-box (the same one) after 9 rounds with probability 2^{-112}. By using a structure of 256 plaintexts with 120 bits set to a fixed value and the last S-box input taking all possible values, we obtain 2^{15} different pairs with a single active S-box. This gives 2^{112} input pairs if we collect 2^{97} such structures, and we expect one pair with only one active S-box in the output. Since such events only happen with probability 2^{-120} with a random function, we don't expect any false positive after 2^{112} pairs. As a result, we have a distinguisher for 9 rounds with a cost of 2^{104}. We additionally expect that this distinguisher can be extended to a few more rounds using partial decryption.

From Involutive to Non-involutive Components. If we do not restrict the design to involutive S- and L-boxes, we can hope that the bound given by Eq. (1) will not be tight. More precisely, we expect that there should not be any trail reaching the minimal number of active S-boxes with the optimal probability for every S-box transition. For this purpose, we first count the number of active S-boxes for truncated trails. That is, for each state we only care about which columns are non-zero and build all the possible transitions. In this context, it is important to note that a truncated input to the diffusion layer can give several different truncated outputs, and does not necessarily behave linearly. For instance, if we start from 00101000, we have to consider five possible transitions:

$$\mathsf{L}[\texttt{00101000}], \mathsf{L}[\texttt{00100000}] \vee \mathsf{L}[\texttt{00001000}], \mathsf{L}[\texttt{00101000}] \vee \mathsf{L}[\texttt{00100000}],$$
$$\mathsf{L}[\texttt{00101000}] \vee \mathsf{L}[\texttt{00001000}], \mathsf{L}[\texttt{00101000}] \vee \mathsf{L}[\texttt{00100000}] \vee \mathsf{L}[\texttt{00001000}].$$

More generally, the possible transitions are of the following form:

$$x_0 \vee x_1 \vee \ldots \vee x_l \quad \rightsquigarrow \quad \mathsf{L}[x_0] \vee \mathsf{L}[x_1] \vee \ldots \vee \mathsf{L}[x_l]$$

Non-linear transitions will usually lead to states with more active columns, but the extra degrees of freedom from the non-linearity allow better trails than the product trails with only linear transitions. For $l = 8$, we ran an exhaustive search over all L-boxes with branch number 5, and found that the best ones give trails with at least 53 active S-boxes for 16 rounds (rather than 40 active S-boxes when L is an involution). For $l = 16$, building all the possible transitions for a fixed L-box is already a hard problem, so we cannot test many different L-boxes. We ran a randomized search by permuting the lines and columns of the $RM(2, 5)$ systematic generator used in Sect. 2.2. The best non-involutive L-box we found (given in Sect. 2.2) gives truncated trails with at least 64 active S-boxes over 12 rounds. More precisely, we can compute the minimum number of active S-boxes with an involutive L-box and our best non-involutive L-box as:

Rounds	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Involutive	1	8	9	16	17	24	25	32	33	40	41	48	49	56	57	64
Non-involutive	1	8	12	20	24	30	34	40	46	52	58	64	68	74	80	86

If we consider a 64-bit cipher with $l = 16$ and $s = 4$, we can go further in the analysis and find the best differential trails with completely instantiated differences. We expect that this will again improve the upper bound on the probability of trails, because in general it is not possible to select a specific difference so that all S-box transitions have maximal probability. We ran this search using an A* algorithm [24], with some additional ideas from Matsui's branch-and-bound algorithm [35]. We used the best truncated trails as heuristic estimate for future path-cost in A*, and refined it by computing the best trails with increasing numbers of rounds. Once we know the probability of the best instantiated r-round trail, we update the heuristic if some truncated r-round trails were expected to have a higher probability. Doing so, we found that some choices of L-box and S-box give 6-round trails with probability at most 2^{-64} and 8-round trails with probability at most 2^{-90} (the candidate L-box in Sect. 2.2 together with S-box $S = \{6, 1, 0, 7, \mathtt{E}, 4, \mathtt{F}, \mathtt{D}, 5, \mathtt{B}, 2, \mathtt{C}, 3, 8, \mathtt{A}, 9\}$ is an example). These values should be compared with 2^{-48} (resp. 2^{-64}) if L and S are restricted to involutions, and to the previous bound of 2^{-56} (resp. 2^{-80}) for the same components using the analysis with truncated trails. This indicates that LS-designs based on involutive components require about 4/3 as many rounds as with non-involutive components to reach a similar security level for these parameters ($l = 16$ and $s = 4$). The computation took several days and dozens of gigabytes of RAM. We believe it gives a good indication about the relative security of involutive vs. non-involutive ciphers that should also be valid with larger S-boxes, even though we cannot run the search for optimal trails with $l = 16$ and $s = 8$ in practice.

Application to Fantomas. Using our search for truncated trail, we have found the following bounds for linear and differential trails on Fantomas:

$$\Pr_{\text{lin}}(6) \leq 2^{-56}, \qquad \Pr_{\text{diff}}(6) \leq 2^{-112},$$
$$\Pr_{\text{lin}}(7) \leq 2^{-68}, \qquad \Pr_{\text{diff}}(7) \leq 2^{-136}.$$

These bounds imply that simple linear and differential attacks on Fantomas can only work with trails over 6 rounds or less, and we have a security margin of 6 rounds. We expect that this bound is not tight, as shown by our analysis of instantiated trails on a 64-bit LS-design (but we cannot run a similar analysis on Fantomas in practice). In addition, we note that the best attack on Robin is based on a truncated differential corresponding to several simple trails, but this effect will be quite limited for Fantomas because optimal trails do not have the strong structure of product trails (on 6 rounds, the best truncated differential using a collection of product trails has a probability of 2^{-117}).

Impossible Differentials. We finally searched for a class of impossible differentials where we do not use the S-boxes properties, i.e. we only considered which S-boxes are active at each round, and we used the possible transitions of the linear layer combined with the fact that the S-box is a bijection. This search is similar to the search for truncated trails described above, and the hardest part is again to build all the possible transitions through L operations. We found that the longest impossible differential for Robin and Fantomas (in this class) only spans three rounds. For Robin, there are are 48420 impossible input-output patterns, an example is given by `0000000000000001` $\not\rightarrow$ `0000000000000010`. For Fantomas, there are 35951 impossible differentials, an example is given by `0000000000000001` $\not\rightarrow$ `0000000000000001`. We can compare this result with similar results on the AES. The best known impossible differential is in this class of impossible differentials and spans four rounds. Since our diffusion layer mixes all the 16 S-box input every round, it makes sense to have shorter impossible differentials.

4.2 Generic Attacks Against Even-Mansour Ciphers

In 1991, a simple block cipher design was proposed by Even and Mansour, using only one permutation and two different key values [19]. It was later revisited in [19] and extended towards key-alternating ciphers by using n permutations and $n + 1$ key values. One special case of such ciphers is the Single-key Even-Mansour (SEM) scheme, which is defined by using n permutations together with one key. Due to their simplicity, these designs have attracted the attention of several cryptanalysts, and various published results apply to their generic versions [12,17,18] or particular instances such as LED or `Zorro` [23,27,37,38].

Since LS-designs correspond to SEM schemes, we discuss the applicability of these previous results to our case study. In particular, and although we do not claim security against related-key and chosen-key attacks, we briefly look at these

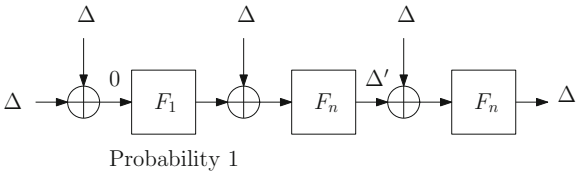


Fig. 3. Three-round related-key characteristic for LS-designs.

adversarial scenarios in a generic manner. Starting with chosen-key differentials, the attacker can not only control the n-bit input value but also the n-bit key value. As a result, the number of freedom degrees he has is doubled (to $2n$) compared to the standard differential cryptanalysis. We know from Sect. 4.1 that the best differential characteristics for LS-designs have probability:

$$\Pr_{\text{diff}}(2r) \leq \Pr_{\text{diff}}^{\max}(S)^{r \times \mathcal{B}(L)}.$$

Therefore, it is possible to mount an attack for $2r$ rounds if $2^{-2n} \leq \Pr_{\text{diff}}(2r)$. Taking the example of Robin, the best 2-round characteristic has probability 2^{-32} and the number of freedom degrees equals 256. Thus, in the worst case it could be possible to attack $8 \cdot 2 = 16$ rounds by using all degrees of freedom. Similar observations can be made in the related-key setting. Namely, by applying the results from [38], the best differential characteristic over two rounds can be extended to three rounds for LS-designs (as represented in Fig. 3). Hence, the required number of rounds to achieve n-bit security will increase by 50 %.

4.3 Algebraic Attacks

In algebraic cryptanalysis, a cipher is expressed as a large system of non-linear equations (typically over $GF(2)$) and a solution for the system is searched. Although it is possible to describe any algorithm in terms of multivariate equations, solving them is an NP-hard problem already for quadratic ones. The precise complexity of algebraic cryptanalysis is difficult to evaluate and security against these attacks is usually argued by exhibiting the size and number of unknowns in the systems, together with a reasoning about the cipher's algebraic degree.

S-boxes in LS-designs can be described in the same number of equations as the number of non-linear gates. Let e denote the number of non-linear gates, l denote the size of the L-box and N_r by the number of rounds. Then, the entire system for a fixed key LS-design consists of $(N_r \cdot e \cdot 128/l)$ quadratic equations in $(N_r \cdot 128 \cdot 2)$ variables. That leads to 3072 equations in 4096 variables for Robin and 2112 equations in 3072 variables for Fantomas (the AES has 6400 equations in 2560 variables). We expect these numbers to be sufficient for both instances to be secure against algebraic attacks, in view of the time and memory complexities needed to solve small-scale AES variants presented in [10]. As for the algebraic degree, we used the work [8] to compute the cumulative algebraic degree in

function of the number of rounds. This algebraic degree reaches maximum after five rounds for both Robin and Fantomas, in which case a partition of size 2^{127} is required to construct zero-sum distinguishers. More precisely, we have:

# of rounds	1	2	3	4	5	6	7	8
Robin	6	36	112	125	127	127	127	127
Fantomas	5	25	110	125	127	127	127	127

4.4 Other Cryptanalyses

We use round constants to make all rounds different and prevent slide attacks [6] (that are based on the self-similarity of the round function). Rotational cryptanalysis [30] is a powerful technique against Addition-Rotation-XOR (ARX)-based block ciphers. But the application of S-box operations in LS-designs makes this attack unlikely to succeed. Integral cryptanalysis or square attacks [13] are primarily purposed for word-oriented ciphers but can be adapted to bitslice ones. Our analyses suggest that up to 4 rounds of Robin or Fantomas can be targeted in this way (which also leaves comfortable security margins). Eventually, boomerang attacks [50] assume that we can find two characteristics for $(2{\cdot}r_1{+}1)$ and $(2{\cdot}r_2{+}1)$ rounds in the target algorithms. By using Eq. (1), we can approximate these probabilities as $\Pr(2 \cdot r + 1) \leq \Pr_{\text{diff}}^{\max}(S)^{r \times \mathcal{B}(L)+1}$. Hence, the probability of a boomerang distinguisher becomes $\Pr_{\text{diff}}^{\max}(S)^{2\cdot((r_1+r_2)\times\mathcal{B}(L)+2)}$, which must be better than 2^{-n}. Setting the parameters of Robin or Fantomas in this equation, we find $2(r_1 + r_2) + 2 < 5.5$ and the attack works for at most five rounds. Therefore, boomerang attacks should not a concern for LS-designs.

5 Performance Evaluations

The main objective of LS-designs is to allow efficient and secure software implementations for 8-bit micro-controllers. Therefore, we first report on the performances of protected implementations of Robin and Fantomas on an Atmel ATmega644p micro-controller, together with the AES, `Zorro`, PICARO and NOEKEON[3]. The results in Fig. 4 (given for different number of shares in the masking scheme) show that the performances of Robin and NOEKEON (both involutive ciphers) are remarkably close. They confirm that bitslice ciphers optimized for Boolean masking allow more efficient implementations that previously obtained, e.g. with the AES, `Zorro` or PICARO. They also illustrate the additional gains that can be obtained by considering non-involutive components (e.g. with Fantomas). Combined with a highly regular design, with all operations operating on well-aligned 8-bit data, we believe this evaluation supports the conclusion that LS-designs are promising ciphers for side-channel resistance.

[3] LED and PRESENT have the same number of non-linear gates, but encrypt only 64-bit. So we do not expect them to bring improvements in our masked setting.

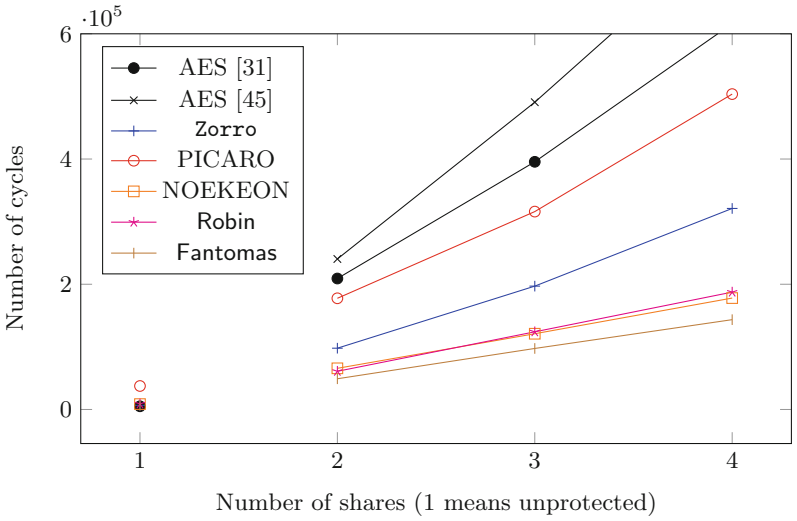


Fig. 4. Encryption time for a 128-bit block in an Atmel AtMega644p.

Besides, we also found that LS-designs are very efficient on desktop CPUs with large SIMD units, at least for unprotected implementations. Taking the example of Fantomas in counter mode, we can evaluate several inputs in parallel and use the full width of SIMD units. Let us describe in more details an implementation using SSSE3 instructions with 128-bit registers. We will compute 16 instances of Fantomas in parallel (with 16 different plaintexts), using 16 SSE registers, every register containing one byte from each copy of Fantomas (8 registers for the high order bytes, and 8 other registers the low order bytes). The S-box layers compute two sets of 128 S-boxes in parallel, using 128-bit wide bitwise operations; this takes 96 instruction. For the L-box layer, we use the `pshufb` instruction as a 4-bit to 8-bit look-up table. The 16-bit L-box is decomposed as eight 4-bit to 8-bit look-up tables and 6 XORs[4]; our implementation requires 280 instructions to compute 16 parallel linear layers (*i.e.* 128 16-bit L-boxes). With these figures, our implementation of Fantomas runs at 6.3 cycles/byte (for long messages) on a Core i7 CPU (Nehalem micro-architecture). Thanks to the `pshufb` instruction, the L-box layer is not subject to cache timing attacks.

As a point of comparison, the bitsliced AES implementation of Käsper and Schwabe [28] would take respectively 326 and 102 cycles for the same number of S-boxes and linear layers (the full AES takes 6.9 cycles/byte on the same CPU – this is the faster known implementation of AES of this CPU). On the one hand our S-box is much easier to implement in a bitslice way than the AES S-box, since it was one of our design goals. On the other hand, our linear-layer is optimized for a table-based implementation, and more complex than that of the AES. It can still be implemented rather efficiently, but it becomes the dominant factor in this implementation. This shows that LS-designs can reach performances comparable

[4] This can be reduced to seven table look-ups for Robin, thanks to the L-box structure.

Table 3. Implementation results with a parallel mode for long messages.

	Fantomas	Robin	AES	
			w/o AES-NI [28]	w/AES-NI
ARM Cortex A15	14.2	18.1	17.8	N/A
Atom	33.3	43.5	17	N/A
Core i7 Nehalem	6.3	8.1	6.9	N/A
Core i7 Ivy Bridge	4.2	5.5	5.4	1.3

to the AES on high-end CPUs, excluding implementations using hardware AES instructions (Table 3). We also expect reasonable performances on Atom or ARM Cortex-A CPUs, which are used in some embedded systems and include a good vector engine with a permutation instruction (SSSE3 and NEON, respectively). Moreover, the latest Intel CPUs support 256-bit wide SIMD operation using AVX2 operations; we expect that this will give even better performances[5].

6 Open Problems

This paper introduces LS-designs as an interesting family of secure and efficient block ciphers, with good properties for masked implementations. Since their instantiation mainly depends on the selection of good S- and L-boxes, a natural scope for further research is to find better such components, in particular for large bit-sizes (e.g. 8-bit and more for S-boxes, 32-bit and more for L-boxes). Improvements in these lines would directly lead to more optimized ciphers.

Besides, our current investigations mainly considered software implementations. But the efficient gate-level representation of Robin and Fantomas makes them potentially suitable for hardware implementations as well. As a result, it would be interesting to study their threshold implementations, and to compare the resulting performances with other algorithms that are efficient in this setting, such as NOEKEON again [39] or more recent designs like FIDES [5].

Acknowledgements. This work has been funded in parts by the ERC project 280141 (acronym CRASH). François-Xavier Standaert is an associate researcher of the Belgian Fund for Scientific Research (FNRS-F.R.S.).

References

1. Barreto, P., Rijmen, V.: The KHAZAD legacy-level block cipher. Primitive submitted to NESSIE, 4 (2000)
2. Bertoni, G., Coron, J.-S.: CHES 2013. LNCS, vol. 8086. Springer, Heidelberg (2013)

[5] At the time of writing we haven't had access to an AVX2-enabled CPU yet, and the 256-bit version of `pshufb` is not available in the first version of AVX.

3. Biham, E.: FSE1997. LNCS, vol. 1267. Springer, Heidelberg (1997)
4. Biham, E., Shamir, A.: Differential cryptanalysis of DES-like cryptosystems. In: Menezes, A., Vanstone, S.A. (eds.) CRYPTO 1990. LNCS, vol. 537, pp. 2–21. Springer, Heidelberg (1991)
5. Bilgin, B., Bogdanov, A., Knezevic, M., Mendel, F., Wang, Q.: Fides: lightweight authenticated cipher with side-channel resistance for constrained hardware. [2], pp. 142–158
6. Biryukov, A., Wagner, D.: Slide Attacks. [32], pp. 245–259
7. Bogdanov, A.A., Knudsen, L.R., Leander, G., Paar, C., Poschmann, A., Robshaw, M., Seurin, Y., Vikkelsoe, C.: PRESENT: an ultra-lightweight block cipher. In: Paillier, P., Verbauwhede, I. (eds.) CHES 2007. LNCS, vol. 4727, pp. 450–466. Springer, Heidelberg (2007)
8. Boura, C., Canteaut, A., De Cannière, C.: Higher-order differential properties of Keccak and *Luffa*. In: Joux, A. (ed.) FSE 2011. LNCS, vol. 6733, pp. 252–269. Springer, Heidelberg (2011)
9. Boyar, J., Peralta, R.: A new combinational logic minimization technique with applications to cryptology. In: Festa, P. (ed.) SEA 2010. LNCS, vol. 6049, pp. 178–189. Springer, Heidelberg (2010)
10. Cid, C., Murphy, S., Robshaw, M.: Small scale variants of the AES. In: Gilbert, H., Handschuh, H. (eds.) FSE 2005. LNCS, vol. 3557, pp. 145–162. Springer, Heidelberg (2005)
11. Coron, J.S.: Higher order masking of look-up tables. Cryptology ePrint Archive, Report 2013/700 (2013). http://eprint.iacr.org/2013/700
12. Daemen, J.: Limitations of the Even-Mansour construction. [25], pp. 495–498
13. Daemen, J., Knudsen, L.R., Rijmen, V.: The block cipher SQUARE. [3], pp. 149–165
14. Daemen, J., Peeters, M., Van Assche, G.: Bitslice ciphers and power analysis attacks. In: Schneier, B. (ed.) FSE 2000. LNCS, vol. 1978, p. 134. Springer, Heidelberg (2001)
15. Daemen, J., Peeters, M., Assche, G.V., Rijmen, V.: Nessie proposal: NOEKEON (2000). http://gro.noekeon.org/Noekeon-spec.pdf
16. Daemen, J., Rijmen, V.: The wide trail design strategy. In: Honary, B. (ed.) Cryptography and Coding 2001. LNCS, vol. 2260, p. 222. Springer, Heidelberg (2001)
17. Dinur, I., Dunkelman, O., Keller, N., Shamir, A.: Key recovery attacks on 3-round Even-Mansour, 8-step LED-128, and full AES^2. Cryptology ePrint Archive, Report 2013/391 (2013). http://eprint.iacr.org/2013/391
18. Dunkelman, O., Keller, N., Shamir, A.: Minimalism in cryptography: the Even-Mansour scheme revisited. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT 2012. LNCS, vol. 7237, pp. 336–354. Springer, Heidelberg (2012)
19. Even, S., Mansour, Y.: A construction of a cipher from a single pseudorandom permutation. [25], pp. 210–224
20. Gérard, B., Grosso, V., Naya-Plasencia, M., Standaert, F.X.: Block ciphers that are easier to mask: how far can we go? [2], pp. 383–399
21. Grassl, M.: Bounds on the minimum distance of linear codes and quantum codes (2007). http://www.codetables.de. Accessed 8 November 2013
22. Grosso, V., Standaert, F.X., Faust, S.: Masking vs. multiparty computation: how large is the gap for AES? [2], pp. 400–416
23. Guo, J., Nikolic, I., Peyrin, T., Wang, L.: Cryptanalysis of Zorro. Cryptology ePrint Archive, Report 2013/713 (2013). http://eprint.iacr.org/2013/713
24. Hart, P.E., Nilsson, N.J., Raphael, B.: A formal basis for the heuristic determination of minimum cost paths. IEEE Trans. Syst. Sci. Cybern. **4**(2), 100–107 (1968)

25. Imai, H., Rivest, R.L., Matsumoto, T.: ASIACRYPT 1991. LNCS, vol. 739. Springer, Heidelberg (1993)
26. Ishai, Y., Sahai, A., Wagner, D.: Private circuits: securing hardware against probing attacks. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 463–481. Springer, Heidelberg (2003)
27. Isobe, T., Shibutani, K.: Security analysis of the lightweight block ciphers XTEA, LED and Piccolo. In: Susilo, W., Mu, Y., Seberry, J. (eds.) ACISP 2012. LNCS, vol. 7372, pp. 71–86. Springer, Heidelberg (2012)
28. Käsper, E., Schwabe, P.: Faster and timing-attack resistant AES-GCM. In: Clavier, C., Gaj, K. (eds.) CHES 2009. LNCS, vol. 5747, pp. 1–17. Springer, Heidelberg (2009)
29. Kerckhof, S., Durvaux, F., Hocquet, C., Bol, D., Standaert, F.-X.: Towards green cryptography: a comparison of lightweight ciphers from the energy viewpoint. In: Prouff, E., Schaumont, P. (eds.) CHES 2012. LNCS, vol. 7428, pp. 390–407. Springer, Heidelberg (2012)
30. Khovratovich, D., Nikolić, I.: Rotational cryptanalysis of ARX. In: Hong, S., Iwata, T. (eds.) FSE 2010. LNCS, vol. 6147, pp. 333–346. Springer, Heidelberg (2010)
31. Kim, H., Hong, S., Lim, J.: A fast and provably secure higher-order masking of AES S-Box. [42], pp. 95–107
32. Knudsen, L.: FSE 1999. LNCS, vol. 1636. Springer, Heidelberg (1999)
33. Kwon, D., et al.: Information security and cryptology - ICISC 2003. In: Lim, J.-I., Lee, D.-H. (eds.) ARIA. LNCS, vol. 2971, pp. 432–445. Springer, Heidelberg (2004)
34. Matsui, M.: Linear cryptanalysis method for DES cipher. In: Helleseth, T. (ed.) EUROCRYPT 1993. LNCS, vol. 765, pp. 386–397. Springer, Heidelberg (1994)
35. Matsui, M.: On correlation between the order of S-boxes and the strength of DES. [46], pp. 366–375
36. Matsui, M.: New block encryption algorithm MISTY. [3], pp. 54–68
37. Mendel, F., Rijmen, V., Toz, D., Varıcı, K.: Differential analysis of the LED block cipher. In: Wang, X., Sako, K. (eds.) ASIACRYPT 2012. LNCS, vol. 7658, pp. 190–207. Springer, Heidelberg (2012)
38. Nikolić, I., Wang, L., Wu, S.: Cryptanalysis of round-reduced LED. In: Moriai, S. (ed.) FSE 2013. LNCS, vol. 8424, pp. 112–130. Springer, Heidelberg (2014)
39. Nikova, S., Rijmen, V., Schläffer, M.: Secure hardware implementation of nonlinear functions in the presence of glitches. J. Cryptol. **24**(2), 292–321 (2011)
40. Nyberg, K.: Linear approximation of block ciphers. [3], pp. 439–444
41. Piret, G., Roche, T., Carlet, C.: PICARO – a block cipher allowing efficient higher-order side-channel resistance. In: Bao, F., Samarati, P., Zhou, J. (eds.) ACNS 2012. LNCS, vol. 7341, pp. 311–328. Springer, Heidelberg (2012)
42. Preneel, B., Takagi, T.: CHES 2011. LNCS, vol. 6917. Springer, Heidelberg (2011)
43. Prouff, E., Roche, T.: Higher-order glitches free implementation of the AES using secure multi-party computation protocols. [42], pp. 63–78
44. Rijmen, V., Barreto, P.: Nessie proposal: WHIRLPOOL (2000). https://www.cosic.esat.kuleuven.be/nessie/
45. Rivain, M., Prouff, E.: Provably secure higher-order masking of AES. In: Mangard, S., Standaert, F.-X. (eds.) CHES 2010. LNCS, vol. 6225, pp. 413–427. Springer, Heidelberg (2010)
46. De Santis, A.: EUROCRYPT 1994. LNCS, vol. 950. Springer, Heidelberg (1995)
47. Toffoli, T.: Reversible computing. In: de Bakker, J., van Leeuwen, J. (eds.) Automata, Languages and Programming. LNCS, vol. 85, pp. 632–644. Springer, Heidelberg (1980)

48. Ullrich, M., Cannière, C.D., Indesteege, S., Küçük, Ö., Mouha, N., Preneel, B.: Finding optimal bitsliced implementations of 4 × 4-bit S-boxes. Symmetric Key Encryption Workshop, p. 20. Copenhagen, DK (2011)
49. Veyrat-Charvillon, N., Gérard, B., Standaert, F.-X.: Security evaluations beyond computing power. In: Johansson, T., Nguyen, P.Q. (eds.) EUROCRYPT 2013. LNCS, vol. 7881, pp. 126–141. Springer, Heidelberg (2013)
50. Wagner, D.: The boomerang attack. [42], pp. 156–170
51. Whitnall, C., Oswald, E., Standaert, F.-X.: The myth of generic DPA...and the magic of learning. In: Benaloh, J. (ed.) CT-RSA 2014. LNCS, vol. 8366, pp. 183–205. Springer, Heidelberg (2014)

A S-Boxes Cryptanalytic Properties

We consider the S-box $\mathsf{S} : \mathbb{F}_{2^n} \to \mathbb{F}_{2^n}$ as a vector of Boolean functions $\mathsf{S} = (f_0, \ldots, f_{n-1})$, $f_i : \mathbb{F}_{2^n} \to \mathbb{F}_2$. For $x \in \mathbb{F}_{2^n}$ and $u \in \mathbb{F}_2^n$, the notation x^u stands for the product $\prod_{i=0}^{n-1} x_i^{u_i}$, with the convention $0^0 = 1$. The cardinality of a set A, is denoted by $\#A$. For two elements $a, b \in \mathbb{F}_{2^n}$, we denote the dot product as: $a.b = \sum_{i=0}^{n-1} a_i b_i$. Hw(.) denotes the Hamming weight function.

Non-linearity. We use the Walsh transform and spectrum to evaluate the correlation of a linear approximation $(a, b) \neq (0, 0)$. Their definitions are given below.

Definition 1. *Walsh transform of a Boolean vector* S*:*

$$W_{\mathsf{S}}(a, b) = \sum_{x \in \mathbb{F}_{2^n}} (-1)^{a.x + b.\mathsf{S}(x)}.$$

Definition 2. *Walsh spectrum of a Boolean vector* S*:*

$$\Omega_{\mathsf{S}} = \{W_{\mathsf{S}}(a, b) | a, b \in \mathbb{F}_{2^n}, (a, b) \neq (0, 0)\}.$$

The smaller is the maximum of Ω_S, the stronger is the S-box regarding linear cryptanalysis [34]. In particular, a value $max(\Omega_{\mathsf{S}})$ for the Walsh spectrum of S implies that its best linear approximation has probability $\mathrm{Pr}_{lin} = \frac{max(\Omega_{\mathsf{S}})}{2^n}$.

Differential profile. We similarly use the differential spectrum to evaluate the resistance of an S-box against differential cryptanalysis [4].

Definition 3. *Differential spectrum of a Boolean vector S:*

$$\Delta_{\mathsf{S}} = \{\#\{X | \mathsf{S}(X + a) = \mathsf{S}(X) + b\} | a, b \in \mathbb{F}_{2^n}, (a, b) \neq (0, 0)\}.$$

The smaller is the maximum of Δ_{S}, the strongest is the S-box regarding differential cryptanalysis. In particular, a value $max(\Delta_{\mathsf{S}})$ for the differential spectrum of S implies that its best differential has probability $\mathrm{Pr}_{diff} = \frac{max(\Delta_{\mathsf{S}})}{2^n}$.

Algebraic degree. Although the tools for analyzing algebraic attacks are not as advanced as for linear and differential ones, the algebraic degree is generally considered as a good indicator of security against them. Moreover, having a non-maximal algebraic degree allows distinguishing a function from a random one. For any Boolean function, the algebraic degree can be defined as follows.

Definition 4. *Algebraic degree of a boolean function f. A Boolean function f can be uniquely represented using its Algebraic Normal Form (ANF):*

$$f(x) = \sum_{u \in \mathbb{F}_2^n} a_u x^u.$$

The algebraic degree of f is defined as:

$$\deg(f) = \max_{u \in \mathbb{F}_2^n} \{\mathrm{Hw}(u), a_u \neq 0\}.$$

Definition 5. *Algebraic degree of a Boolean vector S. The algebraic degree of a vector is defined as the maximum degree of its coordinates:*

$$\deg(\mathsf{S}) = \max_{0 \leq i < n} \deg(f_i).$$

B Secure Computation of Non-linear Operations

In this section, we use the notation $\in_R \mathbb{F}$ to mean that a value is chosen uniformly in $\mathbb{F}$ and $\cdot$ denotes the non-linear operation of $\mathbb{F}$, i.e. the field multiplication for extensions of $\mathbb{F}_2$ and the AND operator for vector spaces over $\mathbb{F}_2$.

Algorithm 2. Non linear operation performed on two masked secrets x and y

Require: Shares $(x_i)_i$ and $(y_i)_i$ satisfying $\oplus_i x_i = x$ and $\oplus_i y_i = y$.
Ensure: Shares $(w_i)_i$ satisfying $\oplus_i w_i = x \cdot y$.

for i from 0 to d **do**
  for j from $i+1$ to d **do**
    $r_{i,j} \in_R \mathbb{F}$;
    $r_{i,j} \leftarrow (r_{i,j} \oplus x_i \cdot y_j) \oplus x_j \cdot y_i$;
  end for
end for
for i from 0 to d **do**
  $w_i \leftarrow x_i \cdot y_i$;
  for j from 0 to d, $j \neq i$ **do**
    $w_i \leftarrow w_i \oplus r_{i,j}$;
  end for
end for

SPRING: Fast Pseudorandom Functions from Rounded Ring Products

Abhishek Banerjee[1], Hai Brenner[2], Gaëtan Leurent[3(✉)], Chris Peikert[1], and Alon Rosen[2]

[1] Georgia Institute of Technology, Atlanta, GA, USA
[2] IDC Herzliya, Herzliyya, Israel
[3] INRIA Team SECRET, Paris, France
gaetan.leurent@normalesup.org

Abstract. Recently, Banerjee, Peikert and Rosen (EUROCRYPT 2012) proposed new theoretical pseudorandom function candidates based on "rounded products" in certain polynomial rings, which have rigorously provable security based on worst-case lattice problems. The functions also enjoy algebraic properties that make them highly parallelizable and attractive for modern applications, such as evaluation under homomorphic encryption schemes. However, the parameters required by BPR's security proofs are too large for practical use, and many other practical aspects of the design were left unexplored in that work.

In this work we give two concrete and practically efficient instantiations of the BPR design, which we call SPRING, for "subset-product with rounding over a ring." One instantiation uses a generator matrix of a binary BCH error-correcting code to "determinstically extract" nearly random bits from a (biased) rounded subset-product. The second instantiation eliminates bias by working over suitable moduli and decomposing the computation into "Chinese remainder" components.

We analyze the concrete security of these instantiations, and provide initial software implementations whose throughputs are within small factors (as small as 4.5) of those of AES.

Keywords: Pseudorandom functions · Noisy learning problems · Learning with rounding · Lattices

A. Banerjee—Research supported by the fourth author's grants.

H. Brenner—Research supported by the ERC under the EU's Seventh Framework Programme (FP/2007–2013) ERC Grant Agreement n. 307952.

C. Peikert—This material is based upon work supported by the National Science Foundation under Grant CNS-0716786 and CAREER Award CCF-1054495, by the Alfred P. Sloan Foundation, by the Defense Advanced Research Projects Agency (DARPA) and the Air Force Research Laboratory (AFRL) under Contract No. FA8750-11-C-0098, and by BSF grant 2010296. The views expressed are those of the authors and do not necessarily reflect the official policy or position of the National Science Foundation, the Sloan Foundation, DARPA or the U.S. Government, or the BSF.

A. Rosen—Research supported by BSF grant 2010296 and by the ERC under the EU's Seventh Framework Programme (FP/2007–2013) ERC Grant Agreement no. 307952.

C. Cid and C. Rechberger (Eds.): FSE 2014, LNCS 8540, pp. 38–57, 2015.
DOI: 10.1007/978-3-662-46706-0_3

1 Introduction

Pseudorandom functions (PRFs) are fundamental objects in symmetric cryptography, which are frequently used in the construction of higher-level objects like block ciphers, message authentication codes, and encryption. A PRF takes as arguments a "key" and an input string of a particular length, and deterministically produces an output string of a particular (possibly different) length. Informally, over the choice of a random (and secret) key that is used for all inputs, a PRF cannot be efficiently distinguished from a truly random function via adaptive oracle (i.e., "black-box") access.

Constructions of PRFs have historically fallen into two broad classes: theoretically sound designs that admit security proofs under mathematically natural intractability assumptions (e.g., [BM82,BBS86,GGM84,NR95,NR97,NRR00]), and designs whose conjectured security is supported by the lack of any known effective cryptanalytic attack (e.g., DES [NIS77], AES [DR02], and countless others). Constructions of the latter type can be very fast and are dominant in practice, but their internal complexity and lack of rigorous security reductions may increase their risk of succumbing to new kinds of attacks, especially for very strong and subtle security notions like PRFs. And while theoretically sound designs tend to be mathematically elegant, they have so far been far too inefficient or otherwise impractical for real-world use.

Mathematical simplicity is sometimes viewed as a weakness in terms of security, because underlying "structure" can sometimes be used as a lever for non-trivial attacks. On the other hand, a number of recently proposed applications of PRFs, requiring properties such as efficient homomorphic evaluation [ACPR13] or "key-homomorphism" [BLMR13], demonstrate that PRFs with algebraic structure can deliver significant performance advantages over those that lack such structure (in some cases offering an improvement of several orders of magnitude; see for instance [ACPR13] vs. [GHS12,CCK+13]).

At the heart of these recent developments is a new class of candidate PRFs constructed by Banerjee, Peikert and Rosen (BPR) [BPR12], which enjoy useful algebraic structure, and may help to bridge the gap between theoretical soundness and practical efficiency. The BPR constructions are based on the pseudorandomness properties of *rounded subset-products* in suitable polynomial rings. BPR give evidence for the asymptotic security of their constructions, proving them secure under well-studied hardness assumptions like "learning with errors" LWE [Reg05,Pei09] and its ring variant ring-LWE [LPR10], and by implication, worst-case problems on point lattices.

1.1 The SPRING Family of Pseudorandom Functions

One of the main constructions in [BPR12] is a class of PRF candidates that we call SPRING, which is short for "**s**ubset-**p**roduct with **r**ounding over a **ring**." Let n be a power of two, and let R denote the polynomial ring $R := \mathbb{Z}[X]/(X^n + 1)$,

which is known as the $2n$th *cyclotomic ring*.[1] For a positive integer p, let R_p denote the quotient ring

$$R_p := R/pR = \mathbb{Z}_p[X]/(X^n + 1),$$

i.e., the ring of polynomials in X with coefficients in $\mathbb{Z}_p$, where addition and multiplication are modulo both $X^n + 1$ and p. (For ring elements $r(X)$ in R or R_p, we usually suppress the indeterminate X.) We often identify $r \in R_p$ with the vector $\mathbf{r} \in \mathbb{Z}_p^n$ of its n coefficients in some canonical order. Let R_p^* denote the multiplicative group of units (invertible elements) in R_p.

For a positive integer k, the SPRING family is the set of functions $F_{a,\boldsymbol{s}} : \{0,1\}^k \to \{0,1\}^m$ indexed by a unit $a \in R_p^*$ and a vector $\boldsymbol{s} = (s_1, \ldots, s_k) \in (R_p^*)^k$ of units. The function is defined as the "rounded subset-product"

$$F_{a,\boldsymbol{s}}(x_1, \ldots, x_k) := S\left(a \cdot \prod_{i=1}^{k} s_i^{x_i}\right), \tag{1}$$

where $S\colon R_p \to \{0,1\}^m$ for some $m \leq n$ is an appropriate "rounding" function. For example, BPR uses the rounding function $\lfloor \cdot \rceil_2\colon R_p \to R_2 \equiv \mathbb{Z}_2^n$ that maps each of its input's n coefficients to $\mathbb{Z}_2 = \{0,1\}$, depending on whether the coefficient is closer modulo p to 0 or to $p/2$. (Formally, each coefficient $b \in \mathbb{Z}_p$ is mapped to $\lfloor \frac{2}{p} \cdot b \rceil \in \mathbb{Z}_2$.)

It is proved in [BPR12] that when a and the s_i are drawn from appropriate distributions, and p is sufficiently large, the above function family is a secure PRF family assuming that the "ring learning with errors" (ring-LWE) problem [LPR10] is hard in R_p. This proof is strong evidence that the family has a sound design and is indeed a secure PRF, at least in an asymptotic sense. The intuition behind the security argument is that the rounding hides all but the most-significant bits of the product $a \cdot \prod_i s_i^{x_i}$, and the rounded-off bits can be seen as a kind of "small" error (though one that is generated deterministically from the subset-product, rather than as an independent random variable as in the LWE problem). And indeed, (ring-)LWE and related "noisy learning" assumptions state that noisy products with secret ring elements are indistinguishable from truly uniform values.

We stress that the *known proof* of security (under ring-LWE) requires the modulus p to be very large, i.e., exponential in the input length k. Yet as discussed in [BPR12], the large modulus appears to be an artifact of the proof technique, and the family appears not to *require* such large parameters for concrete security. Indeed, based on the state of the art in attacks on "noisy learning" problems like (ring-)LWE, it is reasonable to conjecture that the SPRING functions can be secure for rather small moduli p and appropriate rounding functions (see Sect. 4 for further details).

[1] It is the $2n$th cyclotomic ring because the complex roots of $X^n + 1$ are all the $2n$th primitive roots of unity. The BPR functions can be defined over other cyclotomic rings as well, but in this work we restrict to powers of two for simplicity and efficiency.

1.2 Our Contributions

In this work we give two new, optimized instantiations of the SPRING PRF family for parameters that offer high levels of concrete security against known classes of attacks, and provide very high-performance software implementations.

Because we aim to design practical functions, we instantiate the SPRING family with relatively small moduli p, rather than the large ones required by the theoretical security reductions from [BPR12]. This allows us to follow the same basic construction paradigm as in [BPR12], while taking advantage of the fast integer arithmetic operations supported by modern processors. We instantiate the parameters as various (but not all) combinations of

$$n = 128, \quad p \in \{257, 514\}, \quad k \in \{64, 128\},$$

which (as explained below in Sect. 1.3) yields attractive performance, and allows for a comfortable margin of security. The choice of modulus $p \in \{257, 514\}$ is akin to the one made in SWIFFT, for a practical instantiation of a theoretically sound lattice-based collision-resistant hash function [LMPR08]. Also as in SWIFFT, our implementations build on Fast Fourier Transform-like algorithms modulo $q = 257$.

Working with small moduli p requires adjusting the rounding function $S(\cdot)$ in the SPRING construction so that its output on a uniformly random element of R_p^* does not have any noticeable bias (which otherwise would clearly render the function insecure as a PRF). We use rounding functions of the form $S(b) = G(\lfloor b \rceil_2)$, where $\lfloor \cdot \rceil_2 \colon R_p \to R_2$ is the usual coefficient-wise rounding function that provides (conjectured) indistinguishability from a potentially *biased* random function, and $G \colon R_2 \to \{0,1\}^m$ for some $m \leq n$ is an appropriate post-processing function that reduces or removes the bias. (In [BPR12], G is effectively the identity function, because a huge modulus p ensures no noticeable bias in the rounded output.)

For each value of the modulus $p \in \{257, 514\}$ we have a different concrete instantiation, which we respectively call SPRING-BCH and SPRING-CRT. These instantiations differ mainly in the computation of the subset-products in R_p^*, and in the definition of the bias-reducing function G.

SPRING-BCH. In this instantiation, we use an odd modulus $p = q = 257$, which admits very fast subset-product computations in R_q^* using Fast Fourier Transform-type techniques (as mentioned above). However, because p is odd, the usual rounding function $\lfloor \cdot \rceil_2 \colon R_p \to R_2$ has bias $1/q$ on each of the output coefficients (bits). To reduce this bias, the function G multiplies the 128-dimensional, $1/q$-biased bit vector by the 64×128 generator matrix of a binary (extended) BCH error-correcting code with parameters $[n, m, d] = [128, 64, 22]$, yielding a syndrome with respect to the dual code. This simple and very fast "deterministic extraction" procedure (proposed in [AR13]) reduces the bias exponentially in the distance $d = 22$ of the code, and yields a 64-dimensional vector that is 2^{-145}-far from uniform when applied to a 128-dimensional bit vector of independent $1/q$-biased bits. However, this comes at the cost of outputting $m = 64$ bits instead of $n = 128$, as determined by the rate m/n of the code.

SPRING-CRT. In this instantiation, we use an even modulus $p = 2q = 514$, and decompose the subset-product computation over R^*_{2q} into its "Chinese remainder" components R^*_2 and R^*_q. For the R^*_q component we use the same evaluation strategy as in SPRING-BCH, but for fast subset-products in the R^*_2 component we need new techniques. We prove that the multiplicative group R^*_2 decomposes into $n/2$ small cyclic groups, having power-of-two orders at most n. We also give explicit "sparse" generators for these cyclic components, and devise fast algorithms for converting between the "cyclic" representation (as a vector of exponents with respect to the generators) and the standard polynomial one. These tools allow us to transform a subset-product in R^*_2 into a subset-*sum* of vectors of (small) exponents with respect to the generators, followed by one fast conversion from the resulting vector of exponents to the polynomial it represents.

For rounding R^*_{2q}, we show that standard rounding of a uniformly random element of R^*_{2q} to R_2 directly yields $n-1$ independent and unbiased bits, so our function G simply outputs these bits. The main advantage over SPRING-BCH is the larger output size (almost twice as many bits), and hence larger throughput, and in the simpler and tighter analysis of the bias. On the other hand, we also show that the CRT decomposition of R^*_{2q} can be exploited somewhat in attacks, by effectively canceling out the R^*_2 component and recognizing the bias of the rounded R^*_q component. Fortunately, for our parameters the best attacks of this type appear to take almost 2^{128} bit operations, and around 2^{119} space.

We refer to Sects. 2 and 3 for further details on these instantiations, and to Sect. 4 for a concrete security analysis.

1.3 Implementations and Performance

We implement the two variants of SPRING described above, both for standalone evaluations on single inputs, and in a counter-like (CTR) mode that is able to amortize much of the work across consecutive evaluations. For the counter itself we use the Gray code, which is a simple way of ordering the strings in $\{0,1\}^k$ so that successive strings differ in only one position. Then when running SPRING in counter mode, each successive subset-product can be computed from the previous one with just one more multiplication by either a seed element or its inverse. More precisely, we store the currently computed subset-product $b := a \prod_{i=1}^{n} s_i^{x_i}$. (The Gray code starts with 0^k, so the initial subset-product is simply a.) If the next input x' flips the ith bit of x, then we update the old subset-product to $b' = b \cdot s_i$ if $x_i = 0$, otherwise $b' = b \cdot s_i^{-1}$.

For the SPRING-CRT instantiation, which works in $R^*_{2q} \cong R^*_2 \times R^*_q$, we use two methods for computing (subset-)products in the R^*_2 component. The first uses the cyclic decomposition of R^*_2 as described above, and is the fastest method we have found for computing a standalone subset-product "from scratch." The other method uses the native "carryless polynomial multiplication" (PCLMUL) instruction available in recent Intel processors, and/or precomputed tables, for single multiplications in the Gray code counter mode.

We benchmarked our implementations on a range of CPUs with several different microarchitectures. As a point of reference, we use the highly optimized AES benchmarks from eBACS [eBA], and the bitsliced implementation for Käsper and Schwabe [KS09]. We report our performances measures in Table 1, using high-end desktop processors (Core i7), and small embedded CPUs found in tablets and smart-phones (Atom and ARM Cortex). Even though the architectures of those machine are quite different, our results are very consistent: in counter mode, SPRING-BCH is between 8 and 10 times slower than AES (as measured by output throughput), while SPRING-CRT is about 4.5 times slower than AES (disregarding AES implementation with AES-NI when they are available). We expect similar results on other CPUs with similar SIMD engines. Finally, we mention that the very latest Intel CPUs (Haswell microarchitecture) include a new 256-bit wide SIMD engine with support for integer operations (AVX2). We expect that an AVX2 implementation of SPRING would run about twice as fast on those processors, yielding very compelling performance.

Table 1. Implementation results for SPRING-BCH and SPRING-CRT with $n = 128$, in both standalone and Gray code counter mode (CTR). Speeds are presented in processor cycles per output byte, and are compared with the best known AES implementations.

	SPRING-BCH		SPRING-CRT		AES-CTR	
	Standalone	CTR	Standalone	CTR	w/o AES-NI	w/AES-NI
ARM Cortex A15	220	170	250	77	17.8	N/A
Atom	247	137	235	76	17	N/A
Core i7 Nehalem	74	60	76	29.5	6.9	N/A
Core i7 Ivy Bridge	60	46	62	23.5	5.4	1.3

1.4 Organization

The rest of the paper is organized as follows. We discuss the details of the SPRING-BCH and SPRING-CRT instantiations in Sects. 2 and 3 respectively. We follow by analyzing the concrete security of our instantiations against known attacks in Sect. 4.1, expanding on one class of combinatorial attacks on SPRING-CRT in Sect. 4.2. Finally, we present certain implementation details and code optimizations in Sect. 5.

2 SPRING-BCH

Here we describe our first instantiation, SPRING-BCH, which works over R_q^* for a suitable prime q, and uses a BCH code for reducing the bias of the rounded subset-product.

2.1 Fast Subset Product in R_q

Efficient operations in the ring R_q were given in prior work by Lyubashevsky *et al.* [LMPR08] (following [Mic02,PR06,LM06]). They give a Chinese Remainder decomposition of this ring as $R_q \cong \mathbb{Z}_q^n$, for prime $q = 1 \pmod{2n}$, and gave fast FFT-like algorithms for converting between (the standard polynomial representation of) R_q and $\mathbb{Z}_q^n$. In particular, the multiplicative group of units R_q^* is isomorphic to $(\mathbb{Z}_q^*)^n$. Since $\mathbb{Z}_q^*$ is cyclic and of order $q-1$, a subset-product in R_q reduces to a subset-sum of n-dimensional vectors of exponents modulo $q-1$ (with respect to some generator of $\mathbb{Z}_q^*$). Once the final vector of exponents have been computed, the corresponding element in $\mathbb{Z}_q^n$ can be computed by table lookups, and finally converted to its polynomial representation via the FFT-like algorithm from [LMPR08].

2.2 Rounding via BCH Code

Since q is odd, the usual rounding function $\lfloor \cdot \rceil_2 \colon R_q \to R_2$, when applied to a random input in R_q, outputs a ring element in R_2 whose (bit) coefficients are independent and have bias $1/q$. In this subsection we define a function $G \colon R_2 \to \{0,1\}^m$ that dramatically reduces this bias using a BCH code.

Definition 1 ([NN90]). *The* bias *of a distribution $X \in \{0,1\}^m$ with respect to $I \subseteq [m]$ is defined as*

$$bias_I(X) = \left| \Pr\left[\bigoplus_{i \in I} x_i = 0\right] - \Pr\left[\bigoplus_{i \in I} x_i = 1\right] \right|.$$

Let max-bias(X) denote the maximal bias of X over all nonempty $I \subseteq [m]$.

Theorem 1 ([NN90]). *Let $X \in \{0,1\}^m$ be a random variable. Then*

$$2 \cdot \Delta(X, U_m) \leq \sqrt{2^m} \cdot \textit{max-bias}(X)$$

where $\Delta(X, U_m)$ denotes the statistical difference of X from the uniform distribution on m bits.

Proposition 1 ([AR13]). *Let G be a generator matrix of a binary linear code with parameters $[n, m, d]$, and let $D \in \{0,1\}^n$ be a distribution of independent bits such that $bias_{\{i\}}(D) \leq \epsilon$ for every $i \in [n]$. Then max-bias$(G \cdot D) \leq \epsilon^d$.*

From the above we get that when applied to a random input $b \in R_q$, the statistical distance of the distribution $S(b)$ from uniform is at most $(1/q)^d \sqrt{2^m}/2$. Note that in SPRING-BCH, we are actually applying G to $\lfloor b \rceil_2$ for a random *unit* $b \in R_q^*$, in which case the coefficients of $\lfloor b \rceil_2$ are not quite independent. Since we are anyway only heuristically modeling the subset-products as uniformly random and independent, we believe that it is safe to heuristically assume that G provides low bias in our instantiation.

In terms of implementation, generator matrices of BCH codes over $GF(2)$ are preferable, since the rows of the matrix are cyclic shifts of a single row, which facilitates fast implementation. We note that n is a power of 2, and any BCH code over $GF(2)$ is of length $2^t - 1$ for some integer t. To make the matrix compatible with an n that is a power of two, we use the extended-BCH code, which is obtained in a standard way by appending a parity bit to the codewords, and increases the code distance d by one. We finally note that for our chosen parameters $n = 128, m = 64$, the BCH code with parameters $[127, 64, 21]$ and its extension with parameters $[128, 64, 22]$ have the largest known minimum distance for these specific rates.

3 SPRING-CRT

We now describe our second instantiation, called SPRING-CRT, which uses *unbiased* rounding on an *even* modulus of the form $p = 2q$, where q is an odd prime as in the instantiation from the previous section.

By the Chinese Remainder Theorem, the natural ring homomorphism $R_{2q} \to R_2 \times R_q$ is a ring isomorphism, and moreover, there is an explicit map which lets us convert back and forth between the two representations. Specifically, it is easy to verify that the pair $(b_2, b_q) \in R_2 \times R_q$ corresponds to

$$b = q \cdot \bar{b}_2 + (q+1) \cdot \bar{b}_q \pmod{2q} \tag{2}$$

for arbitrary $\bar{b}_2, \bar{b}_q \in R_{2q}$ such that $\bar{b}_2 = b_2 \pmod 2$ and $\bar{b}_q = b_q \pmod q$. The CRT isomorphism also induces a group isomorphism between R_{2q}^* and $R_2^* \times R_q^*$, and thus lets us represent the seed elements and their subset-products as pairs in $R_2^* \times R_q^*$. We compute products in the R_q^* component as detailed in Sect. 2.1 above. In the following subsections, we define an unbiased rounding function from R_{2q}^* to R_2, and give fast algorithms for computing products in the R_2^* component.

3.1 Unbiased Rounding of R_{2q}^*

We start by describing how the rounding function from R_{2q} to R_2 can be computed directly from the Chinese remainder components $(b_2, b_q) \in R_2 \times R_q$ of a given $b \in R_{2q}$. As above, let $\bar{b}_q, \bar{b}_2 \in R_{2q}$ denote arbitrary mod-$2q$ representatives of b_2, b_q. By Eq. (2) and the definition of the rounding function $\lfloor \cdot \rceil : R_{2q} \to R_2$,

$$\lfloor b \rceil_2 = \left\lfloor q(\bar{b}_q + \bar{b}_2) + \bar{b}_q \right\rceil_2 = \left\lfloor (\bar{b}_q + \bar{b}_2) + \bar{b}_q/q \right\rceil.$$

If we choose the coefficients of $\bar{b}_q$ from $[-q/2, q/2) \cap \mathbb{Z}$, then each coefficient of $\bar{b}_q/q$ is in the interval $[-1/2, 1/2)$, so

$$\lfloor b \rceil_2 = \bar{b}_q + \bar{b}_2 \mod 2. \tag{3}$$

Equivalently, the coefficient vector of $\lfloor b \rceil_2$ is the exclusive-or of the coefficient vector of $\bar{b}_2$ and the least-significant bits of the coefficients of $\bar{b}_q$.

In SPRING-CRT, we need an unbiased rounding function S from the *unit group* $R_{2q}^* \cong R_2^* \times R_q^*$ to R_2. An element of R_2, viewed as a polynomial, is a unit if and only if the sum of its coefficients is odd. So for a uniformly random element of R_2^*, any fixed choice of $n-1$ coefficients (e.g., all but the constant term) are uniformly random and independent, and the remaining one is determined. Because of Eq. (3) above, any fixed choice of $n-1$ coefficients of $\bar{b}_2$ are uniformly random and independent, over the random choice of $b_2 \in R_2^*$ alone. Therefore, we define our generalized rounding function on $b \in R_{2q}^*$ to output a fixed $n-1$ bits of $\lfloor b \rceil_2 \in R_2$, which is perfectly unbiased.

Note that the above argument depends only on the random choice of the R_2^* component, and doesn't use any of the randomness in the R_q^* component. Using such an argument, $n-1$ independent and unbiased bits is the most we can possibly obtain. Since the number of units in R_{2q}^* is exactly $(q-1)^n \cdot 2^{n-1}$, which is divisible by 2^n, it seems plausible that there could exist a rounding function that outputs n (nearly) unbiased bits given a random unit in R_{2q}^*, but so far we have not been able to find such a function. The main difficulty seems to be that the coefficients of the representative $\bar{b}_q$ are noticeably biased modulo 2.

3.2 Fast Arithmetic in R_2^*

We now give an algebraic decomposition of the group R_2^*, and present fast algorithms for performing subset-products and associated arithmetic operations.

The following theorem says that the unit group R_2^* decomposes into the product of several small cyclic components, having power-of-2 orders at most n. Due to space constraints, a proof is deferred to the full version.

Theorem 2. *Define* $g_{0,0} = 1+(1+x)$ *and* $g_{i,k} = 1+(1+x)^{2^i+k}$ *for* $1 \le i < \lg(n)$ *and* odd $k \in \{1, \ldots, 2^i\}$. *Then*

$$R_2^* \cong C_2^{n/4} \times C_4^{n/8} \times \ldots \times C_{n/2}^1 \times C_n^1 = \prod_{i=1}^{\lg(n)-1} C_{2^i}^{2^{j-i-1}} \times C_n, \tag{4}$$

with each $g_{i,k}$ *being a generator of one of the* $C_{n/2^i}$ *cyclic components.*

There are several ways of representing elements in R_2^*, which each allow for certain arithmetic operations to be performed more or less efficiently. We use the following three representations, the first of which is very good for fast multiplication, and the last of which is used for rounding. (As we shall see, the middle one is a convenient intermediate representation.)

1. Using the cyclic decomposition given in Theorem 2, we can represent an element by its tuple of integer exponents with respect to the generators $g_{i,k}$. We call this the *exponent representation*.
2. We can represent elements in R_2 by their vectors of $\mathbb{Z}_2$-coefficients with respect to what we call the *radix basis* $\{(1+x)^i\}_{0 \le i < n}$. (An element is in R_2^* if and only if its coefficient for the basis element $(1+x)^0 = 1$ is 1.) The name of this basis arises from the fact that $(1+x)^n = 1 + x^n = 0 \pmod 2$, and therefore the coefficients can be thought of as digits in the "radix" $1+x$.

3. Finally, elements in R_2 can be represented by their vectors of $\mathbb{Z}_2$-coefficients with respect to the *power basis* $\{x^i\}_{0 \le i < n}$, i.e., in the usual way as polynomials in x.

We now give algorithms for efficiently converting from the exponent representation to the power representation, using the radix representation as an intermediary.

From exponents to radix basis. We first make a few useful observations about the radix basis, and how powers of the generators $g_{i,k}$ look in this basis.

1. In the radix basis, multiplication by an element of the form $(1+x)^j$ corresponds to shifting the input's coefficient vector j places (and discarding the "top" j coefficients), since $(1+x)^n = 0$ in R_2. Therefore, multiplication by $1 + (1+x)^j$ corresponds to taking the exclusive-or of the input's coefficient vector with that vector shifted by j positions.
2. For any j and ℓ, we have that $(1 + (1+x)^j)^{2^\ell} = 1 + (1+x)^{j \cdot 2^\ell} \in R_2^*$, since the intermediate binomial coefficients $\binom{2^\ell}{i}$ for $0 < i < 2^\ell$ are all even.
3. Raising any generator $g_{i,k}$ to half its order yields $g_{i,k}^{n/2^{i+1}} = 1 + (1+x)^j$, where $j = n/2 + (n/2^{i+1})k$. Moreover, the product of any two elements of this type, for $n/2 \le j_1, j_2 < n$, is

$$(1 + (1+x)^{j_1})(1 + (1+x)^{j_2}) = 1 + (1+x)^{j_1} + (1+x)^{j_2}.$$

Thus, a subset-product of elements of this type can be computed as $\prod_{j \in \mathcal{I}}(1 + (1+x)^j) = 1 + \sum_{j \in \mathcal{I}}(1+x)^j$, for any $\mathcal{I} \subseteq \{n/2, \ldots, n-1\}$.

Now let the exponent representation of some $b \in R_2^*$ be $\{e_{i,k}\}$, where each $e_{i,k}$ denotes the exponent of the generator $g_{i,k}$. Write $e_{i,k} = \sum_{\ell=0}^{\lg(n)-i-1} e_{i,k,\ell} \cdot 2^\ell$, i.e., each $e_{i,k,\ell}$ is the ℓth bit of $e_{i,k}$, and observe that

$$g_{i,k}^{e_{i,k}} = \prod_{\ell=0}^{\lg(n)-i-1} \left(g_{i,k}^{2^\ell}\right)^{e_{i,k,\ell}}, \tag{5}$$

where we know by Item 2 above that $g_{i,k}^{2^\ell} = 1 + (1+x)^{(2^i+k)\cdot 2^\ell}$.

We can now describe the algorithm that converts from exponent to radix representation. We effectively decompose the given powers $e_{i,k}$ of $g_{i,k}$ according to Eq. (5), which we can then compute by Items 1 and 2 above. We note that Item 3 lets us handle all the most significant bits of all the exponents very quickly in one shot. (This yields a practical but not asymptotic improvement over handling these bits more naively.) The precise details are given in Algorithm 1 below.

If the length of the coefficient vector is considered to be the word-size, then apart from the most significant bits of the exponents (which are handled in one word operation in total), the other bits are handled in one shift-and-XOR operation each, which is a constant number of word operations each. Since the exponents take $n-1$ bits in total, the algorithm performs a total of $O(n)$ word operations. (Since each word is n bits long, the bit complexity of Algorithm 1 is $O(n^2)$.)

Algorithm 1. Algorithm to convert from exponent to radix representation

Input: Exponents $e_{i,k} \in [0, n/2^i)$ for positive odd $k < 2^i$ when $0 < i < \lg(n)$, and $k = 0$ when $i = 0$. ▷ Let $e_{i,k,\ell}$ denote the ℓth bit of $e_{i,k}$.
Output: A bit vector $\mathbf{b} \in \mathbb{Z}_2^n$ representing the coefficients of b in the radix basis.
$\mathbf{b} \leftarrow (1, 0, \ldots, 0)$ ▷ Initialize the vector $\mathbf{b}$ to represent $1 \in R_2^*$
for every valid (i, k) pair **do**
  if $e_{i,k,\lg(n)-i-1} = 1$ **then**
    $b[n/2 + k \cdot n/2^{i+1}] \leftarrow 1$
  end if
end for
for every valid (i, k) pair **do**
  for $\ell = (\lg(n) - 2) - i$ down to 0 **do**
    if $e_{i,k,\ell} = 1$ **then**
      $\mathbf{b} \leftarrow \mathbf{b} \oplus (\mathbf{b} \gg ((2^i + k) \cdot 2^\ell))$ ▷ The shift-and-XOR operation.
    end if
  end for
end for

From radix basis to power basis. For the second step, we have a bit vector $\mathbf{b} \in \mathbb{Z}_2^n$ representing some $b \in R_2$ with respect to the radix basis, and wish to convert to the power basis. We express b as follows:

$$b = \sum_{i=0}^{n-1} b_i(1+x)^i = \sum_{i=0}^{n/2-1} b_i(1+x)^i + (1+x)^{n/2} \sum_{i=0}^{n/2-1} b_{i+n/2}(1+x)^i$$

$$= \sum_{i=0}^{n/2-1} b_i(1+x)^i + (1+x^{n/2}) \sum_{i=0}^{n/2-1} b_{i+n/2}(1+x)^i \tag{6}$$

$$= \left(\sum_{i=0}^{n/2-1} (b_i + b_{i+n/2})(1+x)^i \right) + x^{n/2} \sum_{i=0}^{n/2-1} b_{i+n/2}(1+x)^i \pmod 2, \tag{7}$$

where (6) follows from the fact that $(1+x)^{2^j} = 1 + x^{2^j} \pmod 2$ (since $\binom{2^j}{i}$ is even for every $0 < i < 2^j$), and n is a power of 2. Converting the n-bit vector $\mathbf{b}$ therefore reduces to two conversions of $n/2$-bit vectors, namely, the top half of $\mathbf{b}$ and the exclusive-or of the top and bottom halves of $\mathbf{b}$. This directly yields a simple divide-and-conquer algorithm to transform the coefficient vector $\mathbf{b}$ in place, which is detailed in Algorithm 2 below. The number of bit operations follows the simple recursive equation $T(n) = 2T(n/2) + n/2$, which solves to $T(n) = O(n \log n)$.

4 Security Analysis

In this section we analyze the security of our construction against known classes of attacks, and introduce new attacks specific to the structure of R_{2q}.

Algorithm 2. Algorithm to transform from radix basis to power basis of R_2

procedure RADIX-TO-POWER($\mathbf{b}, f, \ell$)
Input: array $\mathbf{b}$, index f and length ℓ ▷ The initial call is made with $f = 0$ and $\ell = n$
Output: subvector $\mathbf{b}[f, f + \ell - 1]$ converted to power basis
  if $\ell > 1$ **then**
    for $i = 0$ to $\ell/2 - 1$ **do**
      $b[f + i] \leftarrow b[f + i] \oplus b[f + \ell/2 + i]$
    end for
    RADIX-TO-POWER($\mathbf{b}, f, \ell/2$)
    RADIX-TO-POWER($\mathbf{b}, f + \ell/2, \ell/2$)
  end if
end procedure

4.1 Overview of Known Attacks

The concrete security of the SPRING PRF for practical parameters is not well understood, but to date there are no known attacks that nontrivially exploit the internal subset-product structure. As discussed earlier, the SPRING construction follows the paradigm from [BPR12], which results in a PRF that is secure against all efficient adversaries, assuming the hardness of the (ring-)LWE problem (appropriately parameterized). Informally, the ring-LWE problem asks the adversary to distinguish many pairs $(a_i, b_i) \in R_p \times R_p$, where each a_i is chosen uniformly and $b_i \approx a_i \cdot s$ is its *noisy* product with a secret ring element s, from uniformly random pairs. The BPR security reductions make two assumptions that do not hold in our instantiations: (1) the parameter p is exponential in the input length k of the PRF, and (2) the seed elements s_i are all "small" ring elements in R; more precisely, they are drawn from the error distribution from the underlying ring-LWE assumption. However, as we shall see in what follows, relaxing these requirements do not appear to introduce any concrete attacks against the function family.

For the sake of modeling certain attacks against SPRING, we can think of it as a LWE-type learning problem. The difference here is that *all* ring elements output by SPRING have rounding errors in them, whereas ring-LWE releases the multiplicand a without any error. In this respect, attacking SPRING seems potentially harder than attacking ring-LWE.

The main classes of attacks against noisy learning problems akin to LWE are: (1) brute-force attacks on the secret, (2) combinatorial-type attacks following [BKW03, Wag02, MR09], (3) lattice reduction attacks, and (4) algebraic attacks following [AG11]. We consider each of these in turn. We note that the lattice and algebraic attack strategies described below apply to (ring-)LWE with our parameters. It is not clear whether these attacks will adapt to SPRING, where multiplicands are not known exactly, but to be conservative we assume that they might. While most of these attacks take a prohibitively large amount of time and/or space (more than 2^{200}), one kind of birthday-type attack technique performs reasonably well against SPRING-CRT. Even in this case, its running time is nearly 2^{128} bit operations and its space requirements are about 2^{119}, when $n = 128$.

Brute-force and combinatorial attacks. A brute-force attack involves searching for a secret $s_i \in R_p^*$, or for the round-off terms in enough samples to uniquely determine an s_i. The secret and round-off terms come from sets of size at least $(p/2)^n$, which is prohibitively large for all our parameters. Combinatorial (or "generalized birthday") attacks on noisy learning problems [BKW03, Wag02] work by drawing an huge number of samples and finding (via birthday collisions) small combinations that sum to lie in a small enough subgroup, then testing whether the noise can be detected. This works for small error rates because the small combinations still retain small error terms. In the case of SPRING-CRT, this style of attack looks the most promising, and a concrete attack in this vein is developed further in Sect. 4.2.

Lattice attacks. Lattice attacks on (ring-)LWE typically work by casting it as a bounded-distance decoding (BDD) problem on a certain class of random lattices (see for instance [MR09, LP11, LN13, vdPS13]). At a high level, the attack draws a sufficiently large number L of samples $(a_i, b_i) \in R_p \times R_p$, so that the secret (in the LWE case) is uniquely determined with good probability. With error rate $1/2$, we need $L \geq \lg(p/2)$ by a simple information-theoretic argument. The attack collects the samples into vectors $\boldsymbol{a}, \boldsymbol{b} \in R_p^L$, and considers the "$p$-ary" lattice $\mathcal{L}$ of dimension $N = nL$ (over $\mathbb{Z}$) corresponding to the set of vectors $s \cdot \boldsymbol{a} \in R_p^L$ for all $s \in R_p$. It then attempts to determine whether $\boldsymbol{b}$ is sufficiently close to $\mathcal{L}$, which corresponds to whether (a_i, b_i) are LWE samples or uniform. In our setting, because the error rate $1/2$ is so large, the distance from $\boldsymbol{b}$ to $\mathcal{L}$ (in the LWE case) is nearly the minimum distance of the lattice, up to a constant factor no larger than four (this is a conservative bound). Therefore, for the attack to succeed it needs to solve BDD (or the shortest vector problem SVP) on $\mathcal{L}$ to within an very small constant approximation factor. For the parameters in our instantiations, the lattice dimension is at least $N \geq n \lg(p/2) \geq 896$ (and likely more). For this setting, the state of the art in BDD and SVP algorithms [CN11, LN13, MV10], take time at least $2^{0.48N} \geq 2^{430}$, and likely more. Moreover, the SVP algorithm of [MV10], which appears to provide the best heuristic runtime in this setting, as a most conservative estimate requires space at least $2^{0.18N} \geq 2^{160}$.

Algebraic attacks. Finally, the algebraic "linearization" attack of Arora and Ge [AG11] yields a lower bound on p for security. The attack is applicable when every coefficient of every error term is guaranteed to belong to a known set of size d; in our setting, $d = p/2$. The attack requires at least N/n ring-LWE samples to set up and solve a dense linear system of dimension N, where

$$N = \binom{n+d}{n} \approx 2^{(n+d) \cdot H(n/(n+d))}$$

and $H(\delta) = -\delta \lg(\delta) - (1-\delta)\lg(1-\delta)$ is the binary entropy function for $\delta \in (0, 1)$. Therefore, the attack requires time and space at least N^2, which is at least 2^{384} for even the most aggressive of all our parameters.

4.2 Birthday-Type Attack on SPRING-CRT

We now describe a specific birthday-type attack on SPRING-CRT, which exploits the structure of the ring R_{2q}. The main idea is to cancel out the R_2 component and to detect the bias in the remaining R_q component.

To do this, we first split the input x into two parts, as $x = y\|z$ for y, z of certain lengths. Then the SPRING-CRT function (for simplicity, without dropping a bit after rounding) can be written as

$$F(y\|z) = \lfloor a \cdot S_y \cdot S_z \rceil_2,$$

where S_y and S_z respectively represent the subset product of the keys s_i indicated by the bits of y and z.

The basic goal in the attack is to try to find values y and y' such that $S_y = S_{y'} \mod 2$. If we have two such values, then $a \cdot S_y \cdot S_z = a \cdot S_{y'} \cdot S_z \mod 2$ for any z, so the R_2 component of the output will be the same for the inputs $y\|z$ and $y'\|z$. By Eq. (3) in Sect. 3.1, this implies that in $F(y\|z) \oplus F(y'\|z)$, the respective R_2 components cancel each other out. Since rounding of a uniform element in R_q has a bias of $1/q$ in each coefficient, the bits of $F(y\|z) \oplus F(y'\|z)$ will be the sum of two biased bits, i.e., the bias is $1/q^2$. This can be detected using $q^4/4$ pairs of output bits (with varying z).

If we repeat the test for 2^n different choices of (y, y'), with high probability, one choice satisfies $S_y + S_{y'} = 0 \mod 2$, and we would be able to detect the bias by the method detailed above. (By contrast, the test would not detect such bias in a truly random function, with high probability.) We can collect the data for the attack with $2^{n/2}$ distinct choices of y and y', each of them using $q^4/(4n)$ values of z. This requires $2^{n/2} \cdot q^4/(4n)$ queries *and* space, and a time complexity of $2^n \cdot q^4/2$.

Generalizing the attack. We can generalize this analysis using y and y' such that $(S_y + S_{y'})^2 = 0 \mod 2$. This implies that the $S_{y,2} + S_{y',2}$ is a multiple of $x^{n/2} + 1$, where $S_{y,2} = S_y \mod 2$ and similarly for $S_{y',2}$. Thus, c_i and $c_{i+n/2}$, the coefficients of x^i and $x^{n+i/2}$ respectively in $S_{y,2} + S_{y',2}$, are the same. This implies that if we XOR the lower and upper halves of $F(y\|z) \oplus F(y'\|z)$, we can effectively remove the R_2 component as above, and then can recognize the bias in the R_q component. Since we sum four bits to remove the R_2 component, we reduce the bias, but a random pair y, y' satisfies the condition with probability $2^{-n/2}$ instead of 2^{-n}. This gives an attack with query and space complexity $2^{n/4} \cdot q^8/(4n)$ and time complexity $2^{n/2} \cdot q^8/4$. This can be generalized further: if we use y and y' such that $(S_y + S_{y'})^t = 0 \mod 2$ (for t a power of 2), we reach a time complexity of $2^{n/t} \cdot q^{4t}/(2t)$.

With $q = 257$ and $n = 128$, our best attack on SPRING-CRT (using $t = 2$) has time complexity roughly $2^{64+64-2} = 2^{126}$, and query and space complexity roughly $2^{n/2} \cdot q^8/(4n) \approx 2^{119}$.

5 Implementation Details

The SPRING design is targeted for efficient implementation using SIMD instructions, and a well-optimized implemention allows us to reach throughputs that are not too far from those of classical symmetric-key constructions.

SIMD instructions perform a given operation on multiple data in parallel. Processors with a SIMD engine usually come with a set of dedicated registers, which can contain a vector of integers or floating point data, and the SIMD instruction set computes arithmetic operations in parallel on the vectors elements, e.g., addition, multiplication, bitwise operations, rotations, etc. as well as some permutations of the vector elements. SIMD instructions were introduced in personal computers to improve the efficiency of multimedia computations, and are now very widely available. SIMD engines with 128-bit wide vectors are available on all desktop processors (SSE2 on x86/x86_64, Altivec on PowerPC, NEON on ARM), and even on embedded platforms such as smart-phones and tablets, with ARM Cortex-A or Intel Atom. Very recently, Intel has introduced AVX2, with integer operations on 256-bit SIMD vectors.

We implemented the two instantiations of SPRING defined in Sects. 2 and 3. SPRING-BCH involves a subset-product in R_q^*, followed by rounding and bias reduction (using the BCH code), while SPRING-CRT involves a subset product in R_{2q}^* followed by rounding. As described in Sect. 3.1, this can be implemented as separate subset-products in R_2^* and R_q^*, followed by an extraction of the least significant bits in the R_q^* component and an exclusive-or with the R_2^* component. For each version, we have an implementation of the PRF with standalone subset-products, and an amortized implementation in Gray code counter mode where we just perform one ring multiplication before each rounding operation. In the following subsections we explain how to efficiently implement the main operations.

5.1 Computations in R_2^*

Subset-sum and conversion from exponent to power basis. We use the cyclic decomposition of R_2^* given in Theorem 2, and store the key in exponent representation, so that the subset-product is a subset-sum of the exponents. A polynomial in R_2 (of degree less than 128) is represented by 32 one-bit indices, 16 two-bit indices, 8 three-bit indices, 4 four-bit indices, 2 five-bit indices, 1 six-bit index, and 1 seven-bit index. We store all 64 indices as 8-bit integers, and use SIMD instructions to compute the sum. Since all the cyclic groups have an order that is a power of 2, we can use 8-bit additions, and remove the extra bits at the end. The conversion to the power basis is done using Algorithms 1 and 2. Algorithm 2 is rewritten iteratively using shift, mask and xor instructions, taking advantage of the inherent parallelism of bitwise operations, as shown in Algorithm 3.

Algorithm 3. Iterative version of Algorithm 2 using the parallelism of bitwise operations

```
procedure RADIX-TO-POWER(b)
Input: 128-bit vector b
    b ← b ⊕ (b ∧ 0xffffffffffffffff0000000000000000) ≫ 64
    b ← b ⊕ (b ∧ 0xffffffff00000000ffffffff00000000) ≫ 32
    b ← b ⊕ (b ∧ 0xffff0000ffff0000ffff0000ffff0000) ≫ 16
    b ← b ⊕ (b ∧ 0xff00ff00ff00ff00ff00ff00ff00ff00) ≫ 8
    b ← b ⊕ (b ∧ 0xf0f0f0f0f0f0f0f0f0f0f0f0f0f0f0f0) ≫ 4
    b ← b ⊕ (b ∧ 0xcccccccccccccccccccccccccccccccc) ≫ 2
    b ← b ⊕ (b ∧ 0xaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa) ≫ 1
end procedure
```

Polynomial multiplication. In counter mode, we found it more efficient to compute a single ring multiplication directly than to use the exponent representation.

On recent Intel CPUs (starting from the Westemere architecture introduced in 2010) and AMD CPUs (starting from the Bulldozer architecture introduced in 2011), there is a carry-less multiplication operation, `pclmulqdq`, that computes a 64-bit polynomial multiplication modulo two. This gives a very efficient implementation of the R_2 multiplication.

Alternatively, we take take advantage of the fact that one of the operands is always a polynomial from the key (or its inverse). Therefore, we can see it as a multiplication by a fixed element in R_2, which is a linear operation. We can precompute tables corresponding to this linear operation with 8-bit subsets of the input range, and compute the full multiplication using $n/8$ table accesses and xors.

More precisely, we precompute $z \cdot s$ for all polynomials z of degree less than 8, and we write a degree-128 polynomial z as $z_0 + x^8 \cdot z_1 + \cdots + x^{120} \cdot z_{15}$, where all the z_i are of degree at most 7. Then we can compute $z \cdot s$ as $(z_0 \cdot s) + x^8 \cdot (z_1 \cdot s) + \cdots + x^{120} \cdot (z_{15} \cdot s)$, which requires only 16 table accesses, rotations, and xors. This trick takes about 1MB of extra memory to store the tables, but this is negligible on the platforms we target.

5.2 Computations in R^*_{257}

Following [LMPR08], we use the Chinese Remainder Theorem isomorphism of the ring $R_q \cong \mathbb{Z}_q^n$ when $q = 1 \pmod{2n}$ is prime. A product in R_q therefore corresponds to a component-wise multiplication of vectors in $\mathbb{Z}_q^n$. Moreover, there are fast FFT-like algorithms, often called "number theoretic transforms" (NTT), for converting between the polynomial representation of R_q and the n-fold product ring $\mathbb{Z}_q^n$.

Subset-sum and conversion. Since the ring elements we multiply are all part of the key, we can generate and store them as vectors in the product ring $\mathbb{Z}_q^n$. Moreover, since these elements are all unit, their entries in $\mathbb{Z}_q^n$ are non-zero, and

we can actually store the discrete logarithms of the entries (with respect to some generator of $\mathbb{Z}_q^*$), so that the subset-product becomes a subset-sum.

Exponentiation by the final summed exponents can be implemented with a simple table lookup. However, we found a slightly more efficient version using vector permutation (`pshufb` in SSSE3) instructions as a 4-bit to 8-bit parallel table lookup. We use the fact that $g^{a+16\times b} = g^a \cdot (g^{16})^b$, where a and b are both 4-bit values, and we use 4-bit to 8-bit tables for g^x and $(g^{16})^x$.

Product and conversion. In Gray code counter mode, we do not use the exponent representation, because a point-wise multiplication is more efficient than a point-wise addition followed by exponentiations. This is because the point-wise multiplication can be parallelized easily while the exponentiation requires either serial table lookups, or a more complex sequence of SIMD operations.

NTT. The bottleneck of our function is the NTT computation, therefore we have to optimize this part aggressively. In our implementation, we reuse the code from the SIMD hash function [LBF08] which happens to use the same parameters as the transformation needed in SPRING. The main tricks used in this implementation are:

Representation of elements. Element in $\mathbb{Z}_{257}$ are stored as signed 16-bit words. The choice of the modulus 257 allows an efficient implementation of the field operations, because 257 is a prime and $256 = -1 \pmod{257}$. Moreover, $\mathbb{Z}_{257}^*$ is a cyclic group of 256 elements, where the subset sum of the logarithms can be computed with a simple 8-bit addition.

Reduction. We use `(x&255) - (x>>8)` to do a partial reduction modulo 257, with the output in $[-127, 383]$. When a full reduction to a smaller range is needed, we subtract 257 to values greater than 128 to reduce the range to $[-128, 128]$. This can be performed completely with SIMD instructions and does not require any division. We note that it is not necessary to perform a reduction after each field operation, because we have some extra bits in a 16-bit word; we have to study the NTT algorithm to find out where reductions are needed.

Multiplication. To compute a multiplication in $\mathbb{Z}_{257}$, we reduce both operands to $[-128, 128]$, and the result can be computed with a single 16-bit multiplication without any overflow.

Using a two-dimensional NTT. Because SIMD instructions compute the same operation on each element of the vectors, we do not use the classical radix-2 NTT algorithm. Instead, we rewrite the one-dimensional NTT as a two-dimensional one. In our implementation, we rewrite an NTT of size 64 as a two-dimensional NTT of size 8×8. The input data is seen as a 8×8 matrix, and the computation of the NTT_{64} is done in three steps:

– First we compute 8 parallel NTT_8 on the columns of the matrix using a decimation in time algorithm.

– We multiply by the twiddle factors, transpose the matrix, and permute the row and the columns following the bit reversal order.
– Then we compute 8 parallel NTT_8 on the columns of the matrix using a decimation in frequency algorithm.

The first and the last step are easy to parallelize with SIMD instructions because they compute the same transformation on 8 independent inputs. Moreover, the root of unity used in the NTT_8 is 4, so the multiplications needed for the NTT_8 are simply bit shifts. The transposition can be implemented using merge operations available on most SIMD instruction sets (e.g., `punpcklwd`/`punpckhwd` in SSE).

For the 128-dimensional NTT, we reused the code of the NTT_{64}, and we have to perform an extra layer of butterfly operations and multiplications by twiddle factors (we decompose the NTT_{128} as an NTT_{64} and a NTT_2).

5.3 Reducing Bias with a BCH Code

After rounding the R_{257} computation output, we are left with a n-dimensional vector over $\mathbb{Z}_2$, each element with a bias of $1/257$. We apply the generator polynomial of a BCH code in order to reduce the output's bias. Specifically, for the case of $n = 128$ we apply the extension using a parity bit on the BCH code with parameters $[127, 64, 21]$ in order to gain a generator for a code with distance 22. Therefore, the whole computation is done using just a few shift and xor instructions, and one final negate instruction. We use the BCH generator polynomial

$$1 + x^2 + x^7 + x^8 + x^{10} + x^{12} + x^{14} + x^{15} + x^{16} + x^{23} + x^{25} + x^{27} + x^{28} + x^{30} + x^{31}$$
$$+x^{32} + x^{33} + x^{37} + x^{38} + x^{39} + x^{40} + x^{41} + x^{42} + x^{44} + x^{45} + x^{48} + x^{58} + x^{61} + x^{63}$$

since it matches our desired code parameters and has a minimal number of nonzero coefficients. The output is 64 bits which makes consecutive outputs easy to maintain in a packed array of 64-bit words. We note that if the PCLMUL instruction is available, we can apply the generator polynomial on 127 rounded output bits immediately by xoring outputs of such two instructions.

References

[ACPR13] Alberini, G., Crockett, E., Peikert, C., Rosen, A.: Fast homomorphic evaluation of symmetric-key primitives (2013) (Manuscript)
[AG11] Arora, S., Ge, R.: New algorithms for learning in presence of errors. In: Aceto, L., Henzinger, M., Sgall, J. (eds.) ICALP 2011, Part I. LNCS, vol. 6755, pp. 403–415. Springer, Heidelberg (2011)
[AR13] Alberini, G., Rosen, A.: Efficient rounding procedures of biased samples (2013) (Manuscript)
[BBS86] Blum, L., Blum, M., Shub, M.: A simple unpredictable pseudo-random number generator. SIAM J. Comput. **15**(2), 364–383 (1986)

[BKW03] Blum, A., Kalai, A., Wasserman, H.: Noise-tolerant learning, the parity problem, and the statistical query model. J. ACM **50**(4), 506–519 (2003)
[BLMR13] Boneh, D., Lewi, K., Montgomery, H., Raghunathan, A.: Key homomorphic PRFs and their applications. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013, Part I. LNCS, vol. 8042, pp. 410–428. Springer, Heidelberg (2013)
[BM82] Blum, M., Micali, S.: How to generate cryptographically strong sequences of pseudo-random bits. SIAM J. Comput. **13**(4), 850–864 (1984). Preliminary version in FOCS 1982
[BPR12] Banerjee, A., Peikert, C., Rosen, A.: Pseudorandom functions and lattices. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT 2012. LNCS, vol. 7237, pp. 719–737. Springer, Heidelberg (2012)
[CCK+13] Cheon, J.H., Coron, J.-S., Kim, J., Lee, M.S., Lepoint, T., Tibouchi, M., Yun, A.: Batch fully homomorphic encryption over the integers. In: Johansson, T., Nguyen, P.Q. (eds.) EUROCRYPT 2013. LNCS, vol. 7881, pp. 315–335. Springer, Heidelberg (2013)
[CN11] Chen, Y., Nguyen, P.Q.: BKZ 2.0: better lattice security estimates. In: Lee, D.H., Wang, X. (eds.) ASIACRYPT 2011. LNCS, vol. 7073, pp. 1–20. Springer, Heidelberg (2011)
[DR02] Daemen, J., Rijmen, V.: The Design of Rijndael: AES — The Advanced Encryption Standard. Information Security and Cryptography. Springer, Heidelberg (2002). doi:10.1007/978-3-662-04722-4
[eBA] eBACS: ECRYPT Benchmarking of Cryptographic Systems. http://bench.cr.yp.to. Accessed 11 Nov 2013
[GGM84] Goldreich, O., Goldwasser, S., Micali, S.: How to construct random functions. J. ACM **33**(4), 792–807 (1986). Preliminary version in FOCS 1984
[GHS12] Gentry, C., Halevi, S., Smart, N.P.: Homomorphic evaluation of the AES circuit. In: Safavi-Naini, R., Canetti, R. (eds.) CRYPTO 2012. LNCS, vol. 7417, pp. 850–867. Springer, Heidelberg (2012)
[KS09] Käsper, E., Schwabe, P.: Faster and timing-attack resistant AES-GCM. In: Clavier, C., Gaj, K. (eds.) CHES 2009. LNCS, vol. 5747, pp. 1–17. Springer, Heidelberg (2009)
[LBF08] Leurent, G., Bouillaguet, C., Fouque. P.-A.: SIMD Is a Message Digest. Submission to NIST (2008). http://www.di.ens.fr/~leurent/files/SIMD.pdf
[LM06] Lyubashevsky, V., Micciancio, D.: Generalized compact knapsacks are collision resistant. In: Bugliesi, M., Preneel, B., Sassone, V., Wegener, I. (eds.) ICALP 2006. LNCS, vol. 4052, pp. 144–155. Springer, Heidelberg (2006)
[LMPR08] Lyubashevsky, V., Micciancio, D., Peikert, C., Rosen, A.: SWIFFT: a modest proposal for FFT hashing. In: Nyberg, K. (ed.) FSE 2008. LNCS, vol. 5086, pp. 54–72. Springer, Heidelberg (2008)
[LN13] Liu, M., Nguyen, P.Q.: Solving BDD by enumeration: an update. In: Dawson, E. (ed.) CT-RSA 2013. LNCS, vol. 7779, pp. 293–309. Springer, Heidelberg (2013)
[LP11] Lindner, R., Peikert, C.: Better key sizes (and attacks) for LWE-based encryption. In: Kiayias, A. (ed.) CT-RSA 2011. LNCS, vol. 6558, pp. 319–339. Springer, Heidelberg (2011)
[LPR10] Lyubashevsky, V., Peikert, C., Regev, O.: On ideal lattices and learning with errors over rings. In: Gilbert, H. (ed.) EUROCRYPT 2010. LNCS, vol. 6110, pp. 1–23. Springer, Heidelberg (2010)
[Mic02] Micciancio, D.: Generalized compact knapsacks, cyclic lattices, and efficient one-way functions. Comput. Complex. **16**(4), 365–411 (2007). Preliminary version in FOCS 2002

[MR09] Micciancio, D., Regev, O.: Lattice-based cryptography. In: Bernstein, D.J., Buchmann, J., Dahmen, E. (eds.) Post Quantum Cryptography, pp. 147–191. Springer, Heidelberg (2009)

[MV10] Micciancio, D., Voulgaris, P.: Faster exponential time algorithms for the shortest vector problem. In: SODA, pp. 1468–1480 (2010)

[NIS77] NIST. FIPS 46–3. Data Encryption Standard. Federal Information Processing Standards, National Bureau of Standards, US Department of Commerce (1977)

[NN90] Naor, J., Naor, M.: Small-bias probability spaces: efficient constructions and applications. SIAM J. Comput. **22**(4), 838–856 (1993). Preliminary version in STOC 1990

[NR95] Naor, M., Reingold, O.: Synthesizers and their application to the parallel construction of pseudo-random functions. J. Comput. Syst. Sci. **58**(2), 336–375 (1999). Preliminary version in FOCS 1995

[NR97] Naor, M., Reingold, O.: Number-theoretic constructions of efficient pseudo-random functions. J. ACM **51**(2), 231–262 (2004). Preliminary version in FOCS 1997

[NRR00] Naor, M., Reingold, O., Rosen, A.: Pseudorandom functions and factoring. SIAM J. Comput. **31**(5), 1383–1404 (2002). Preliminary version in STOC 2000

[Pei09] Peikert, C.: Public-key cryptosystems from the worst-case shortest vector problem. In: STOC, pp. 333–342 (2009)

[PR06] Peikert, C., Rosen, A.: Efficient collision-resistant hashing from worst-case assumptions on cyclic lattices. In: Halevi, S., Rabin, T. (eds.) TCC 2006. LNCS, vol. 3876, pp. 145–166. Springer, Heidelberg (2006)

[Reg05] Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. J. ACM **56**(6), 1–40 (2009). Preliminary version in STOC 2005

[vdPS13] van de Pol, J., Smart, N.P.: Estimating key sizes for high dimensional lattice based systems. Cryptology ePrint Archive, Report 2013/630 (2013). http://eprint.iacr.org/

[Wag02] Wagner, D.: A generalized birthday problem. In: Yung, M. (ed.) CRYPTO 2002. LNCS, vol. 2442, pp. 288–303. Springer, Heidelberg (2002)

Cryptanalysis I

Match Box Meet-in-the-Middle Attack Against KATAN

Thomas Fuhr and Brice Minaud(✉)

ANSSI, 51, boulevard de la Tour-Maubourg, 75700 Paris 07 SP, France
thomas.fuhr@ssi.gouv.fr, brice.minaud@gmail.com

Abstract. Recent years have seen considerable interest in lightweight cryptography. One particular consequence is a renewed study of meet-in-the-middle attacks, which aim to exploit the relatively simple key schedules often encountered in lightweight ciphers. In this paper we propose a new technique to extend the number of rounds covered by a meet-in-the-middle attack, called a match box. Furthermore, we demonstrate the use of this technique on the lightweight cipher KATAN, and obtain the best attack to date on all versions of KATAN. Specifically, we are able to attack 153 of the 254 rounds of KATAN32 with low data requirements, improving on the previous best attack on 115 rounds which requires the entire codebook.

Keywords: Cryptanalysis · Meet-in-the-middle · Biclique · Match box · KATAN

1 Introduction

Over the past few years, ultra-lightweight embedded systems such as RFID tags and sensor nodes have become increasingly common. Many such devices require cryptography, typically for authentication purposes. However, traditional ciphers such as AES were not primarily designed for use in this context. Highly constrained devices impose a very small hardware footprint; on the other hand, they typically do not require a security level as high as that offered by AES.

To cater for this need, a number of lightweight ciphers have been developed, such as PRESENT [5], KATAN [7], LED [9], or Simon [2]. These ciphers aim to offer a trade-off between security and the constraints of embedded systems. This is often achieved by innovative designs that look to push the boundaries of traditional ciphers. The security of these new designs needs to be carefully assessed; in this process, new cryptanalytic techniques have emerged.

In particular, there has been a resurgence in the study of meet-in-the-middle attacks in the context of block ciphers [6,10]. This type of attack requires a fairly simple key schedule, and is rarely applicable to traditional ciphers. However, many lightweight ciphers rely on simple round functions and key schedules, which are compensated by a high number of rounds. This makes them good targets for meet-in-the-middle attacks.

C. Cid and C. Rechberger (Eds.): FSE 2014, LNCS 8540, pp. 61–81, 2015.
DOI: 10.1007/978-3-662-46706-0_4

Our Contribution. In this paper, we propose a new way to extend meet-in-the-middle attacks, which we call a match box. This technique may be seen as a form of sieve-in-the-middle [8] or three-subset meet-int-the-middle attack [6], in that it extends the rounds covered in the middle section of the attack. It does so by relying on a large precomputed lookup table with a special structure. As such, it is also a form of time/memory trade-off.

We demonstrate this technique on the lightweight block cipher KATAN. As a result, we improve on previous results on all three versions of KATAN, both in terms of number of rounds as well as data requirements. Of independent interest is our construction of bicliques on KATAN, which takes full advantage of the linearity of the key schedule, and improves on previous attacks with negligible memory requirements

Related Work. Previous results on KATAN include a conditional differential analysis by Knellwolf, Meier and Naya-Plasencia [11,12] and a differential cryptanalysis of 115 rounds of KATAN32 by Albrecht and Leander [1]. In [10], Isobe and Shibutani describe meet-in-the-middle attacks on reduced versions of all three variants of KATAN. The attack that reaches the highest number of rounds on all three versions is a multidimensional meet-in-the-middle attack by Zhu and Gong [15]. However, this attack may be regarded as an optimized exhaustive search, as it involves performing a partial encryption under every possible value of the key.

Table 1 gives a summary of these results, including our own.

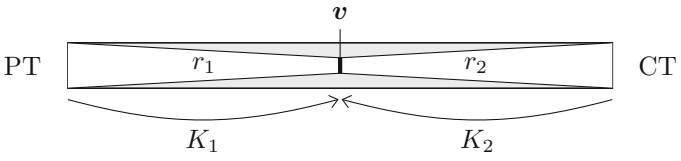


Fig. 1. Meet-in-the-middle attack.

2 Meet-in-the-Middle Attacks

2.1 Meet-in-the-Middle Framework

A meet-in-the-middle attack assumes that a few bits $\boldsymbol{v}$ of internal state may be computed from a plaintext by using a portion K_1 of the key; and that these same bits $\boldsymbol{v}$ may also be computed from the corresponding ciphertext with a portion K_2 of the key. The attack uses one plaintext/ciphertext pair as follows (Fig. 1):

Table 1. Summary of results.

	Model	Data	Memory	Time	Rounds	Reference
KATAN32	CP	2^{22}	-	2^{22}	78	[11]
	KP	138	2^{75}	2^{77}	110	[10]
	CP	2^{32}	-	2^{79}	115	[1]
	KP	3	$2^{79.58}$	$2^{79.30}$	175	[15]
	KP	4	2^{5}	$2^{77.5}$	121	Sect. 4.3
	CP	2^{7}	2^{5}	$2^{77.5}$	131	Sect. 4.5
	CP	2^{5}	2^{76}	$2^{78.5}$	153	Sect. 4.7
KATAN48	CP	2^{34}	-	2^{34}	70	[11]
	KP	128	2^{78}	2^{78}	100	[10]
	KP	2	$2^{79.00}$	$2^{79.45}$	130	[15]
	KP	4	2^{5}	$2^{77.5}$	110	Sect. 4.3
	CP	2^{6}	2^{5}	$2^{77.5}$	114	Sect. 4.5
	CP	2^{5}	2^{76}	$2^{78.5}$	129	Sect. 4.7
KATAN64	CP	2^{35}	-	2^{35}	68	[11]
	KP	116	$2^{77.5}$	$2^{77.5}$	94	[10]
	KP	2	$2^{79.00}$	$2^{79.45}$	112	[15]
	KP	4	2^{5}	$2^{77.5}$	102	Sect. 4.3
	CP	2^{7}	2^{5}	$2^{77.5}$	107	Sect. 4.5
	CP	2^{5}	2^{74}	$2^{78.5}$	119	Sect. 4.7

– For each partial key $k_\cap \in K_\cap = K_1 \cap K_2$[1]:
  - For each partial key $k_1 \in K_1$ extending $k_\cap$, $\boldsymbol{v}$ is computed. For each possible value of $\boldsymbol{v}$, the k_1's leading to that value are stored in a table.
  - For each partial key $k_2 \in K_2$, $\boldsymbol{v}$ is computed. The k_1's leading to this same $\boldsymbol{v}$ are retrieved from the previous table. Each k_2 merged with each k_1 leading to the same $\boldsymbol{v}$ provides a candidate master key.

The actual encryption key is necessarily among candidate keys. Indeed, for the actual key, encryption from the plaintext and decryption from the ciphertext are mirrors of each other, and agree on the intermediate value $\boldsymbol{v}$. If we denote by $|\boldsymbol{v}|$ the size of $\boldsymbol{v}$, candidate keys form a proportion $2^{-|\boldsymbol{v}|}$ of the total key space.

In order to compute the actual encryption key, it remains to test candidate keys against enough plaintext/ciphertext pairs to ensure only one key remains. Each plaintext/ciphertext pair divides the number of candidates keys by $2^{|B|}$, where $|B|$ denotes the block size. Thus, in order to have only one key left, $\lceil |K|/|B| \rceil$ pairs are necessary on average, where $|K|$ denotes the key size.

[1] This notation assumes, for the sake of simplicity, that the key schedule is linear (cf. Sect. 4.2). In general, the requirement is that once $K_\cap$ is guessed, the remaining information in K_1, and the remaining information in K_2 should be independent.

In the end, the attack complexity in number of encryptions is:

$$2^{|K_\cap|} \cdot \left(2^{|K_1 - K_\cap|} \cdot \frac{r_1}{r} + 2^{|K_2 - K_\cap|} \cdot \frac{r_2}{r}\right) + \sum_{i=0}^{\lceil |K|/|B| \rceil - 1} 2^{|K| - |\boldsymbol{v}| - i|B|} \quad (1)$$

where r_1 is the number of rounds in the encryption direction, r_2 is the number of rounds in the decryption direction, and $r = r_1 + r_2$.

Simultaneous Matching. As we have seen, overall, meet-in-the-middle attacks proceed in two stages: a key filtering stage that produces key candidates, followed by a verification stage that tests the key candidates against a few plaintext/ciphertext pairs. This division in two stages is reflected in the complexity of the attack. The complexity of the first stage is determined mostly by the sizes of K_1 and K_2; the complexity of the second stage depends only on the size of $\boldsymbol{v}$ (for a fixed cipher).

Directly tweaking the size of $\boldsymbol{v}$ is one way to try and evenly spread the load between the two stages. However, increasing $\boldsymbol{v}$ will often disproportionately impact the sizes of K_1 and K_2. Simultaneous matching provides a very efficient alternate way of increasing the size of $\boldsymbol{v}$. The idea is to use n plaintext/ciphertext pairs instead of just one. For each guess of K_1 and K_2, we concatenate the $\boldsymbol{v}$'s produced by each pair in order to have a larger global $\boldsymbol{v}$, and use that for matching, as before.

In other words, what we are doing is perform a standard meet-in-the-middle attack, but on a cipher formed by n parallel applications of the basic cipher. This increases only linearly the complexity of the first stage, while exponentially decreasing the complexity of the second stage.

Indirect Matching. With the newfound interest in meet-in-the-middle attack occasioned by lightweight ciphers, a number of techniques originally developed for the cryptanalysis of hash functions have been adapted to meet-in-the-middle attacks on block ciphers. A short survey of these techniques has already been presented in, for example [14], and is out of the scope of this article. Still, we briefly mention one of these techniques, namely indirect matching, as we will use it later on KATAN. We also generalize this technique slightly.

In a regular meet-in-the-middle attack, some value $\boldsymbol{v}$ of the internal state is computed from the left as $e(k_1)$ and from the right as $d(k_2)$, where e and d are essentially a partial encryption and decryption. Keys are filtered by checking $e(k_1) = d(k_2)$. Now assume some key bit k in k_1 only has a linear impact on the value of $e(k_1)$, i.e. $e(k_1) = e'(k'_1) \oplus k$, where k'_1 is k_1 minus the knowledge of k. Then if knowledge of k is included in K_2, the equality in the middle $e(k_1) = d(k_2)$ may be rewritten as $e'(k'_1) = d(k_2) \oplus k = d'(k_2)$. In this way, guessing k is no longer necessary in the encryption direction, and the associated complexity decreases accordingly.

Here, we assumed that k is included in K_2, i.e. k is in $K_\cap$ since it is already in K_1. But we can get the same benefit even if k is in $K_1 - K_\cap$: the only real requirement is that it linearly impacts $e(k_1)$. To show this, the proof is a little

more elaborate than in the previous case. Assume that k is in $K_1 - K_\cap$, and write $e(k_1) = e'(k'_1) \oplus k$ as before.

Up to now, k_1 together with k_2 was assumed to contain knowledge of the entire key. We guessed k_1 from the left, then k_2 from the right and matched compatible guesses by checking $e(k_1) = d(k_2)$. Instead, we are now going to guess k'_1 from the left and k_2 from the right, so the combination of the two does not encompass the entire key (k is missing). Furthermore, all guesses of k'_1 and k_2 are compatible. However, for each pair of guesses, we set $k = e(k'_1) \oplus d(k_2)$, and the combination of k'_1, k_2 and k gives us one candidate master key.

Thus, the number of candidate master keys is unchanged. However, we need not guess k from the left, and the complexity of guessing k_1 is reduced accordingly. Thus the benefit is exactly the same as in the case where k belonged to $K_\cap$. Note that we remain compatible with simultaneous matching: if we use several plaintext/cipherext pairs, they all must agree on k, which yields the usual filter on the candidate master keys.

3 Match Box

We now introduce the match box technique. This technique fits within the general sieve-in-the-middle framework introduced in [8], which we recall here.

3.1 Sieve-in-the-Middle

Let us still denote by $K_\cap$ the information on the key common to K_1 and K_2; furthermore, let K'_1 (resp. K'_2) be the proper part of K_1 (resp. K_2), i.e. the part not already in $K_\cap$. In a standard meet-in-the-middle attack, a few bits of internal state $\boldsymbol{l}$ are computed from the left by guessing $k_1 \in K_1$, then the same bits $\boldsymbol{r}$ are computed from the right by guessing $k_2 \in K_2$. Valid key candidates are determined by checking $\boldsymbol{l} = \boldsymbol{r}$.

However it would often be desirable to compute a few bits of information $\boldsymbol{l}$ from the left and $\boldsymbol{r}$ from the right, and discriminate keys by checking $\mathcal{R}(\boldsymbol{l}, \boldsymbol{r})$ for some general relation $\mathcal{R}$ expressing that $\boldsymbol{l}$ and $\boldsymbol{r}$ are compatible. It arises naturally if, say, $\boldsymbol{l}$ and $\boldsymbol{r}$ contain partial information about the internal state on either side of an S-box. In that case, $\mathcal{R}(\boldsymbol{l}, \boldsymbol{r})$ holds iff there exists an input/output pair of the S-box such that the input extends $\boldsymbol{l}$, and the output extends $\boldsymbol{r}$ (Fig. 2).

When applying this idea, the following problem arises. Once having guessed $k_\cap \in K_\cap$, the natural way to proceed would be to compute $\boldsymbol{l}$ and $\boldsymbol{r}$ for each $k'_1 \in K'_1$ and $k'_2 \in K'_2$ respectively, and exhaustively test $\mathcal{R}(\boldsymbol{l}, \boldsymbol{r})$ for every pair (k'_1, k'_2). However, this would amount to a brute force search since $K_\cap \times K'_1 \times K'_2$ is in fact the entire key. It should be noted that there is no completely general solution to this problem, since $\mathcal{R}$ does need to be tested for every pair $(\boldsymbol{l}, \boldsymbol{r})$ yielded by every (k'_1, k'_2).

In the sieve-in-the-middle paper, this issue is solved by using merging algorithms originally introduced in [13]. These algorithms tend to assume, roughly, that the size of $\boldsymbol{l}$ is less than the size of K'_1 (divided by a sieving factor).

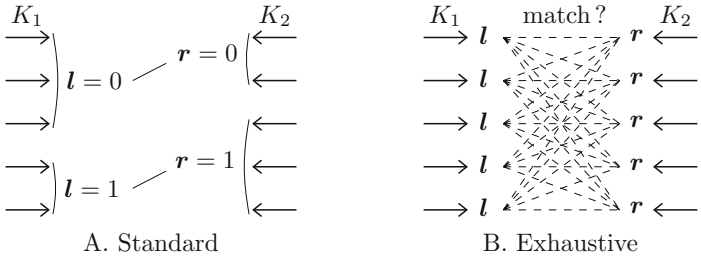


Fig. 2. Comparison of standard and exhaustive matching.

We refer the reader to [8] for a complete explanation of merging techniques. What we propose is a different way of matching $\boldsymbol{l}$ and $\boldsymbol{r}$ while avoiding exhaustive search, which we call a match box.

3.2 Match Box

As we have mentioned in the previous section, the aim of the match box technique is to find compatible partial keys k_1 and k_2, such that the corresponding $\boldsymbol{l}$ and $\boldsymbol{r}$ satisfy a relation $\mathcal{R}$. The idea is to move the computation of $\mathcal{R}$ outside of the loop on $K_\cap$. In order to do this, we anticipate and precompute all possible matchings between $\boldsymbol{l}$ and $\boldsymbol{r}$. We start with an example.

Consider the situation depicted on Fig. 3. Here, $\boldsymbol{l}$ contains some partial information $\boldsymbol{l'}$ about the internal state entering an S-box. At the output of this S-box, some round key is added, and $\boldsymbol{r}$ contains the entire state after the key addition. Now assume that the round key may be decomposed as a sum of some $f_1(k'_1)$ depending on $k'_1 \in K'_1$, and some $f_2(k_2)$ depending on $k_2 \in K_2$. Note that this is automatically true if the key schedule is linear. Since K_2 is known when computing from the right, the component $f_2(k_2)$ may be directly added into $\boldsymbol{r}$.

So in this situation, $\boldsymbol{l} = (\boldsymbol{l'}, k'_1)$ and $\boldsymbol{r}$ are compatible iff $S^{-1}(\boldsymbol{r} \oplus f_1(k'_1))$ equals $\boldsymbol{l'}$ (wherever $\boldsymbol{l'}$ is defined). If $\boldsymbol{r}$ is larger than k'_2, since k'_1 is included in $\boldsymbol{l}$,

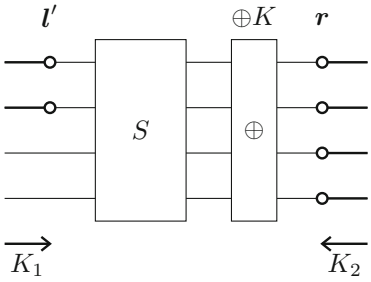


Fig. 3. A typical situation where a match box can apply.

$\boldsymbol{l}$ is also larger than k'_1, and a merging technique in the style of [8] cannot apply. However, a match box is possible.

In general, assume that $\boldsymbol{l} = (\boldsymbol{l'}, k'_1)$ contains the partial key $k'_1 \in K'_1$ plus some extra bits of information $\boldsymbol{l'}$, and $\boldsymbol{r}$ is as before. In order to anticipate all possible computations in the middle, we consider the function $f : k'_1 \mapsto \boldsymbol{l'}$ as a whole. For each value of this function, and each value of $\boldsymbol{r}$, we can precompute a list of the k'_1's leading to $\boldsymbol{l'}$'s such that $\boldsymbol{l'}$, k'_1, and $\boldsymbol{r}$ are compatible. Formally, let us denote by L' the set of values of $\boldsymbol{l'}$, and by R the set of values of $\boldsymbol{r}$. Then we precompute the following table, which we call a "match box":

$$\begin{aligned} M : {L'}^{K'_1} &\to {K'_1}^R \\ f &\mapsto \big(\boldsymbol{r} \mapsto \{k'_1 : \mathcal{R}(\boldsymbol{l}, \boldsymbol{r})\}\big) \text{with } \boldsymbol{l} = (k'_1,\ f(k'_1)) \end{aligned}$$

This table takes as input a function $f : K'_1 \to L'$, and produces as output the function that to each $\boldsymbol{r}$ associates all compatible k'_1's. Once this table has been precomputed, the attack proceeds as follows:

- For each $k_\cap \in K_\cap$:
  - For each $k'_1 \in K'_1$, $\boldsymbol{l'}$ is computed. This yields a function $f : K'_1 \to L'$, from which we obtain $M(f)$.
  - For each $k'_2 \in K'_2$, $\boldsymbol{r}$ is computed. Candidate master keys are those corresponding to the pairs (k'_1, k'_2) for each k'_1 in $M(f)(\boldsymbol{r})$.

The main limitation of this technique is the size of the table, which is approximated by:

$$2^{|l'|^{|K'_1|}+|r|+|K'_1|}$$

In particular, the size of K'_1 (in terms of number of bits) must be exponentially small compared to the size of K. This is not surprising, since we are moving all computations of $\mathcal{R}$ outside of the loop on $K_\cap$: this constraint expresses the fact that there must be less possible situations in the middle than the size of the loop, otherwise we gain nothing.

3.3 Compressing $\mathcal{R}$

Looking more closely at the example in the previous section, the natural way to write $\mathcal{R}$ is in the form:

$$\begin{cases} l'_1 &= f_1(k'_1, \boldsymbol{r}) \\ l'_2 &= f_2(k'_1, \boldsymbol{r}) \\ \dots & \\ l'_{|\boldsymbol{l'}|} &= f_{|\boldsymbol{l'}|}(k'_1, \boldsymbol{r}) \end{cases} \tag{2}$$

where the f_i's are boolean functions.

In this situation, each f_i is a boolean function of k'_1, so it may be written as a polynomial in the bits of k'_1. As such, each f_i can be fully expressed by no more than $2^{|k'_1|}$ coefficients $(f_i^n)_{n<2^{|k'_1|}}$. This is beneficial as long as $|\boldsymbol{l'}| \cdot 2^{|k'_1|} < |\boldsymbol{r}|$,

i.e. there are less f_i^n's than bits of $\boldsymbol{r}$. In this manner, $\boldsymbol{r}$ is effectively shortened to $|\boldsymbol{l'}| \cdot 2^{|k'_1|}$.

The only limit is the size and complexity necessary to build the table converting $\boldsymbol{r}$ into the f_i^n's. Note that in general, $\boldsymbol{r}$ is more or less a set of internal state bits, with potentially some partial keys added in; so computing $\boldsymbol{r}$ and the f_i's is akin to a partial encryption. In that case, for a given $\boldsymbol{r}$, the f_i^n's can be indirectly computed by evaluating the f_i's for all values of k'_1. In this way, for each $\boldsymbol{r}$ and each i, the value of the f_i^n's can be computed in at most $2^{|k'_1|}$ encryption equivalents.

4 Application to KATAN

KATAN is an ultra-lightweight block cipher presented by Christophe de Cannière, Orr Dunkelman and Miroslav Knežević at CHES 2009 [7]. Its design is inspired by the stream cipher Trivium, and relies on two nonlinear feedback registers. This is rather unique for a block cipher, and makes the cryptanalysis of KATAN especially interesting, since it indirectly evaluates the strength of this type of design.

In [7] the authors describe two families of block ciphers, KATAN and KTANTAN, which only differ in their key schedule. In KATAN, the key is stored in a register, while in KTANTAN, it is hardcoded into the circuit. The trade-off is that while the key cannot be modified, the circuit area is significantly reduced by avoiding the need for a register dedicated to the storage of the key. However, KTANTAN been broken [6,14], mostly due to weaknesses in its key schedule. Hereafter we focus solely on KATAN.

4.1 Description of KATAN

KATAN is a family of three block ciphers with block sizes 32, 48, and 64 bits, denoted by KATAN32, KATAN48, and KATAN64 respectively. In all cases the key size is 80 bits, and the total number of rounds is 254. We begin by giving a brief description of KATAN32. KATAN48 and KATAN64 are very similar, as we shall see. We refer the reader to [7] for more details about the design of KATAN.

Key Schedule. The master key is loaded into a 80-bit linear feedback register $(rk_0, \ldots, rk_{79})$, and new round keys are generated by the linear feedback relation:

$$rk_{i+80} = rk_i \oplus rk_{i+19} \oplus rk_{i+30} \oplus rk_{i+67}, \quad 0 \leq i \leq 428 \tag{3}$$

Round Function. The 32-bit plaintext is loaded into two registers A and B of sizes 13 and 19 bits. The round function depicted on Fig. 4 is then applied 254 times, where c_n is a round constant defined by $(c_0, \ldots, c_7) = (1, \ldots, 1, 0)$ and $c_{i+8} = c_i \oplus c_{i+1} \oplus c_{i+3} \oplus c_{i+5}$.

Formally, KATAN32 encryption may be defined as follows. By a_n (resp. b_n), we denote the bit entering register A (resp. B) at round n. Hence, after round n, the content of register A is $(a_{n-12}, \ldots, a_n)$, and the content of register B

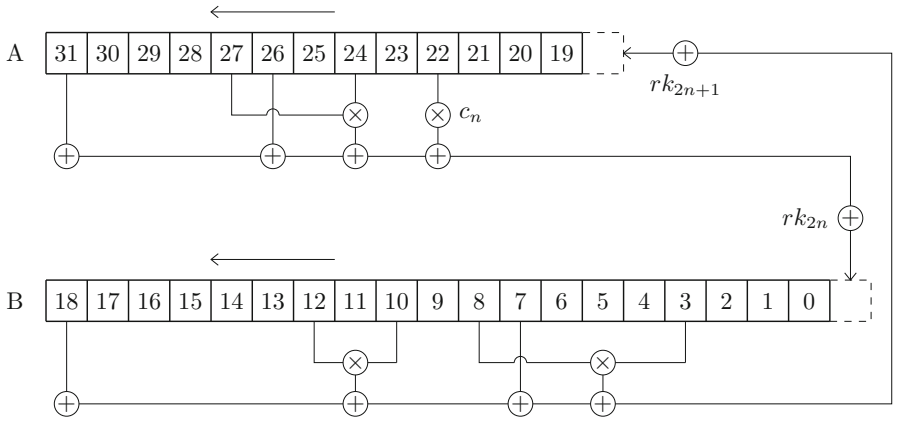


Fig. 4. Round function of KATAN32.

is $(b_{n-18}, \ldots, b_n)$. By convention, the plaintext is $(a_{-13}, \ldots, a_{-1}, b_{-19}, \ldots, b_{-1})$. Then encryption is recursively defined by:

$$\begin{cases} a_n = b_{n-19} \oplus b_{n-8} \oplus b_{n-11} \cdot b_{n-13} \oplus b_{n-4} \cdot b_{n-9} \oplus rk_{2n+1} \\ b_n = a_{n-13} \oplus a_{n-8} \oplus c_n \cdot a_{n-4} \oplus a_{n-6} \cdot a_{n-9} \oplus rk_{2n} \end{cases} \tag{4}$$

and the ciphertext is $(a_{241}, \ldots, a_{253}, b_{235}, \ldots, b_{253})$.

KATAN48. KATAN48 uses two registers of sizes 19 and 29 bits. The registers are updated *twice* per round by the following feedback relation, using the same round keys:

$$\begin{cases} a_n = b_{n-29} \oplus b_{n-20} \oplus b_{n-14} \cdot b_{n-22} \oplus b_{n-7} \cdot b_{n-16} \oplus rk_{2 \cdot \lfloor \frac{n}{2} \rfloor + 1} \\ b_n = a_{n-19} \oplus a_{n-13} \oplus c_n \cdot a_{n-7} \oplus a_{n-8} \cdot a_{n-16} \oplus rk_{2 \cdot \lfloor \frac{n}{2} \rfloor} \end{cases}$$

After 254 rounds, the ciphertext is $(a_{489}, \ldots, a_{507}, b_{479}, \ldots, b_{507})$.

KATAN64. KATAN64 uses two registers of sizes 25 and 39 bits. The registers are updated *three times* per round by the following feedback relation, using the same round keys:

$$\begin{cases} a_n = b_{n-39} \oplus b_{n-26} \oplus b_{n-22} \cdot b_{n-34} \oplus b_{n-10} \cdot b_{n-15} \oplus rk_{2 \cdot \lfloor \frac{n}{3} \rfloor + 1} \\ b_n = a_{n-25} \oplus a_{n-16} \oplus c_n \cdot a_{n-10} \oplus a_{n-12} \cdot a_{n-21} \oplus rk_{2 \cdot \lfloor \frac{n}{3} \rfloor} \end{cases}$$

After 254 rounds, the ciphertext is $(a_{737}, \ldots, a_{761}, b_{723}, \ldots, b_{761})$.

4.2 Linear Key Partition

We now introduce a few notions that will prove useful to mount a meet-in-the-middle attack against KATAN. Let RK_1 (resp. RK_2) denote the set of round keys necessary to compute some fixed bits of internal state at an intermediate

round from the left (resp. from the right). The first step of a meet-in-the-middle attack is to guess the bits of information on the master key common to RK_1 and RK_2 (see Sect. 2.1). Hence it is necessary to define an intersection of RK_1 and RK_2 in terms of bits of information on the master key.

In general, this intersection may be impossible to define. In [10], a generic solution is proposed: all round keys are regarded as independent, i.e. the master key is redefined as the union of all round keys. This yields good results on various lightweight ciphers, including KATAN. However, it has a significant impact on the attack complexity. This can be avoided when the key schedule is linear: indeed, in that case, the intersection of RK_1 and RK_2 can be cleanly defined, as we now show for KATAN.

Let us regard a master key of KATAN as a vector in $E = (\mathbb{Z}/2\mathbb{Z})^{80}$. The value of the master key corresponds to the coordinates of this vector along the canonical basis. Each round key is a linear combination of bits of the master key; that is, it is the image of the master key through some map $(x_i) \mapsto \sum \lambda_i x_i$, i.e. a linear functional on E. Let us denote by $\mathcal{L}(E)$ the space of linear functionals on E.

From this standpoint, the information carried by RK_1 (resp. RK_2) is the value of the master key on the subspace E_{K_1} (resp. E_{K_2}) of $\mathcal{L}(E)$ generated by the round keys of RK_1 (resp. RK_2). Let $E_{K_\cap} = E_{K_1} \cap E_{K_2}$. Then the bits of information on the master key common to RK_1 and RK_2 are exactly the value of the key on the functionals of $E_{K_\cap}$.

Let us choose an arbitrary basis $B_\cap$ of $E_{K_\cap}$, and extend it to a basis B_1 of E_{K_1}, and B_2 of E_{K_2}. Then in concrete terms a partial key in $K_\cap$ is a mapping $B_\cap \to \{0,1\}$; likewise, K_1 and K_2 are regarded as the set of mappings $B_1 \to \{0,1\}$ and $B_2 \to \{0,1\}$ respectively. We are now able to apply the meet-in-the-middle attack framework exactly as it was presented in Sect. 2.1.

In the remainder, it will always be assumed that $B = B_1 \cup B_2$ is a basis for the whole space $\mathcal{L}(E)$. In particular, knowledge of the value of a key on B amounts to knowing the entire key; it will be convenient at times to identify the key space with $\{0,1\}^B$, which we will denote by K, by analogy with K_1 and K_2.

4.3 Key Dependencies

A first step towards building a meet-in-the-middle attack is to choose a value $\boldsymbol{v}$ extracted from an internal state at an intermediate round to serve as a meeting point. In order to make this choice, it is necessary to evaluate which key bits are necessary to compute $\boldsymbol{v}$ from the plaintext, and from the ciphertext (presumably for some reduced version of the cipher). We have carried out this computation using an algorithm similar to Algorithm 1 in [10].

The principle of such an algorithm is that once some round key enters the state, the impacted bit is marked as depending on that key. Then this dependency is propagated along the cipher each time this internal state bit affects other internal state bits. In our case, because we will use indirect matching, we keep track separately of key bits whose impact is linear, and those whose impact is nonlinear.

Table 2. Key dependency of the bit at position 9 (middle) of register B.

KATAN32 Encryption (starting from round 0):

Number of rounds		58	59	60	61	62	63	64	65	66
Dimension of key space	nonlinear	70	71	75	77	78	80	78	79	80
	linear	2	2	2	2	2	0	1	1	0

KATAN32 Decryption (starting from round 254):

Number of rounds		57	58	59	60	61	62	63	64	65
Dimension of key space	nonlinear	68	70	72	73	75	76	78	79	80
	linear	4	3	3	2	1	1	0	0	0

By nature, such an analysis follows a worst case scenario, i.e. it assumes any key bit than could possibly affect an internal state bit, does. In reality, fortuitous simplifications may occur. However, in the case of KATAN, our algorithm was precise enough that experimental tests observed the same dependencies between internal state bits and key bits. Table 2 shows our results for KATAN32.

Basic Meet-in-the-Middle Attack Against KATAN. With what we have so far, we can mount a first meet-in-the-middle attack against KATAN. While this is not the best attack we will propose, it is still worth mentioning because it has a simple description, requires only known plaintexts and minimal data requirements, and improves on previously published attacks.

For KATAN32, if we aim at a complexity around 2^{77}, we can attack $60+61 = 121$ rounds (cf. Table 2). The meeting point is b_{50} (which is indeed at position 9 of register B after 60 rounds). The dimensions of K_1, K_2, $K_\cap$ are 75, 75, 70 respectively (after ignoring linear contributions thanks to indirect matching). We use 4 plaintext/ciphertext pairs for simultaneous matching to ensure that the key verification stage is in 2^{76}. Using (1), and taking into account that we use 4 plaintext/ciphertext pairs, the overall complexity is:

$$4 \cdot \left(2^{75} \cdot \frac{60}{121} + 2^{75} \cdot \frac{61}{121} \right) + \sum_{i=0}^{2} 2^{76-32i} \approx 2^{77.5}$$

In a similar way, we can attack $56+54 = 110$ rounds of KATAN48, and $51+51 = 102$ rounds of KATAN64, both of them with 4 plaintext/ciphertext pairs and complexity $2^{77.5}$.

4.4 Bicliques

The number of rounds covered by a meet-in-the-middle attack may be extended by a biclique. This technique was also originally developed for the cryptanalysis of hash functions [4], and first applied to block ciphers in [3] to produce an accelerated key search against AES. Such a search requires all possible keys to be tried, but each try costs significantly less than a full encryption.

However, bicliques may also be used in the context of a traditional attack, where not all keys are tried. This is the model known as "long bicliques" in [3], and corresponds to [4] for hash functions. We will use this approach against KATAN, and so we recall it here briefly.

Definition 1. *A* biclique *is a triple* $((A_i)_{i\leq n}, (B_i)_{i\leq n}, (K_{i,j})_{i,j\leq n})$*, where the* A_i*'s are internal states at some round* a*, the* B_i*'s are internal states at some round* b*, and the* $(K_{i,j})$*'s are keys satisfying the following property:*

$$\forall i, j \leq n, \quad \mathrm{Enc}_{K_{i,j}}^{a\to b}(A_i) = B_j$$

where $\mathrm{Enc}_{K_{i,j}}^{a\to b}$ *denotes encryption from round* a *to round* b *with key* $K_{i,j}$*.*

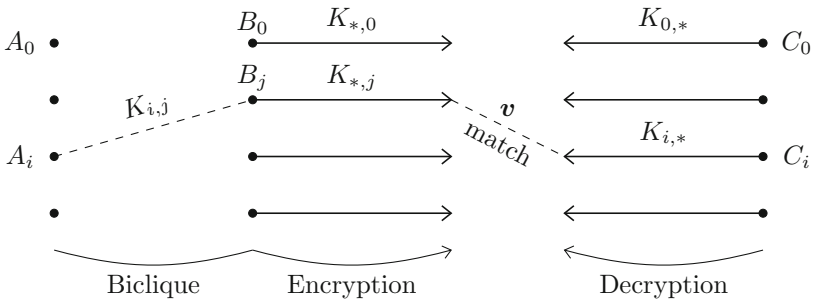


Fig. 5. A meet-in-the-middle attack and compatible biclique.

For simplicity, assume $a = 0$, and we have a biclique covering rounds a to b as in the above definition. In order to construct an attack up to round r, the remaining rounds from b to r must be covered by a meet-in-the-middle attack. Furthermore, the biclique and the meet-in-the-middle segments must be compatible in the following sense. Let C_i be the ciphertext corresponding to A_i after r encryption rounds, et $\boldsymbol{v}$ be the internal value used as a meeting point for the meet-in-the-middle attack. Let $K_{i,*}$ denote the partial information on the key expressing the fact that it is one of the $K_{i,j}$'s, for fixed i and variable j. Let $K_{*,j}$ be defined in the same way.

Then the biclique and the meet-in-the-middle segments of the attack are *compatible* iff the middle value $\boldsymbol{v}$ can be computed starting from B_j with only knowledge of $K_{*,j}$, and from C_i with only knowledge of $K_{i,*}$. The situation is illustrated on Fig. 5. This requirement is quite restrictive. However, it becomes easier to enforce if the key schedule is linear, as we shall see with KATAN.

Attack process.

– For each partial key $k_\cap \in K_\cap = \bigcup K_{i,*} \cap K_{*,j}$:
  - For each $j \leq n$, $\boldsymbol{v}$ is computed starting from B_j using $K_{*,j}$. For each possible value of $\boldsymbol{v}$, the j's leading to that value are stored in a table.

- For each $i \leq n$, $\boldsymbol{v}$ is computed starting from C_i using $K_{i,*}$. The j's leading to this same $\boldsymbol{v}$ are retrieved from the previous table. For each pair (i, j) leading to the same $\boldsymbol{v}$, $K_{i,j}$ is a candidate master key.

If the actual encryption key is among the $K_{i,j}$'s, then it is necessarily a candidate. Indeed, encryption by $K_{i,j}$ will follow the path depicted on Fig. 5. As with a standard meet-in-the-middle attack, it remains to test candidate keys on a few additional plaintext/ciphertext pairs to single out the right key. This step is unchanged. Finally, to ensure that the actual key is among the $K_{i,j}$'s, the key space must be covered by bicliques, and the previous attack is repeated for each biclique.

Construction of a biclique varies depending on the cipher, but in general the construction cost is negligible with respect to the global complexity. Note that it is implicitly assumed that the construction of a biclique on a set of keys $K_{i,j}$ does not imply that each key be computed, i.e. there is a structure. The overall complexity is then the same as that of the meet-in-the middle segment if it were simply applied to a fixed plaintext/ciphertext pair.

4.5 Bicliques on KATAN

We have presented bicliques in the previous section. It remains to show how to construct bicliques on KATAN. Once again, it all comes down to the linearity of the key schedule, and the weak non-linearity of the cipher reduced to a few rounds. In fact, these two properties make it possible to adjoin a biclique to any pre-existing, arbitrary meet-in-the-middle attack, in a compatible manner. Furthermore, a single biclique will suffice to cover the entire key space.

Assume we have a pre-existing meet-in-the-middle attack, with the notation of the previous sections. Recall that $K_\cap$ (resp. K_1, K_2) are regarded as maps $B_\cap \to \{0,1\}$ (resp. $B_1 \to \{0,1\}$, $B_2 \to \{0,1\}$). Let us denote by K_1' (resp. K_2') the proper part of K_1 (resp. K_2) with respect to $K_\cap$, i.e. its restriction to $B_1 - B_\cap$ (resp. $B_1 - B_\cap$).

Let us denote by $\mathrm{Enc}_{a\to b}^k M$ and $\mathrm{Dec}_{b\to a}^k M$ the encryption and decryption of a message M between rounds a and b with key k. We extend this notation to the case where k is a partial key (i.e. an element of K_1, K_1', K_2, K_2', or $K_\cap$) by completing the key by 0 on the rest of B. In addition, for $k \in K$, let us write $k_1 \in K_1$ for its restriction to B_1, and define in a similar way k_1', k_2, and k_2'. Finally, let $k(i)$ denote the value of the i-th round key generated by k; again, if k is only partially defined, it is completed by 0 on the rest of the basis B.

Definition 2. *Let :*

$$\begin{aligned} \mathrm{Bic}_n(K_1, K_2) &= ((A_{k_2} : k_2 \in K_2), (B_{k_1} : k_1 \in K_1), K_2' \oplus K_1) \\ \text{With :} \quad A_{k_2} &= \mathrm{Dec}_{n\to 0}^{k_2'}(0) \\ B_{k_1} &= \mathrm{Enc}_{0\to n}^{k_1}(0) \end{aligned}$$

where 0 *is the null block. Observe that* A_{k_2} *is decrypted by the projection* k_2'.

Proposition 1. *For each version of* KATAN*, there exists $n > 0$ such that for any K_1, K_2, $\mathrm{Bic}_n(K_1, K_2)$ is a biclique. That is, for $k \in K$, with k_1 and k_2 its projections on K_1 and K_2:*

$$\mathrm{Enc}^{k}_{0\to n}(A_{k_2}) = B_{k_1} \tag{5}$$

Proof. The proof is essentially the same for all three versions of KATAN. For the sake of simplicity, we only present the proof for KATAN32. It will be convenient to designate bits in each register by their position, in the order depicted on Fig. 4.

We claim that the proposition holds for $n = 10$. The core of the proof lies in the following property:

$$\forall i \leq 10, \quad \mathrm{Enc}^{k}_{0\to i}(A_{k_2}) = \mathrm{Enc}^{k_1}_{0\to i}(0) \oplus \mathrm{Dec}^{k'_2}_{10\to i}(0) \tag{6}$$

For $i = 10$, this equation becomes:

$$\mathrm{Enc}^{k}_{0\to 10}(A_{k_2}) = \mathrm{Enc}^{k_1}_{0\to 10}(0) = B_{k_1}$$

which is precisely what we want to prove. So (6) implies the proposition. We are going to prove (6) by recursion on $0 \leq i \leq 10$.

For $i = 0$, (6) yields the definition of A_{k_2}, so it holds. Assume that it holds for some round $i < 10$. When we step forward one encryption round, since we are dealing with shift registers, the equality remains true everywhere, except possibly on the two new bits entering the registers (at positions 0 and 19 on Fig. 4). Let us show for instance that the equality remains true for the bit entering register B (position 0). Let us denote by f the feedback function from register A into register B.

Then for the bit entering register B, (6) at round $i + 1$ means:

$$f\big(\mathrm{Enc}^{k}_{0\to i}(A_{k_2})\big) \oplus k(i) = f\big(\mathrm{Enc}^{k_1}_{0\to i}(0)\big) \oplus k_1(i) \oplus f\big(\mathrm{Dec}^{k'_2}_{10\to i}(0)\big) \oplus k'_2(i) \tag{7}$$

where $k(i)$ denotes the value of the i-th round key generated by key k. Since $k = k_1 \oplus k'_2$, using the recursion hypothesis, we get:

$$f\big(\mathrm{Enc}^{k_1}_{0\to i}(0) \oplus \mathrm{Dec}^{k'_2}_{10\to i}(0)\big) = f\big(\mathrm{Enc}^{k_1}_{0\to i}(0)\big) \oplus f\big(\mathrm{Dec}^{k'_2}_{10\to i}(0)\big) \tag{8}$$

Since f is nonlinear, this is not automatically true. However, the only nonlinear interaction in f is a multiplication of bits 24 and 27. Now observe that $\mathrm{Enc}^{k_1}_{0\to i}(0)$ is 0 on bits $19 + i, \ldots, 31$, and $\mathrm{Dec}^{k'_2}_{10\to i}(0)$ is 0 on bits $19, \ldots, 21 + i$. Hence for $n \leq 5$, $\mathrm{Enc}^{k_1}_{0\to i}(0)$ is 0 on bits 24 and 27, and for $n \geq 6$, $\mathrm{Dec}^{k'_2}_{10\to i}(0)$ is 0 on bits 24 et 27.

Assume for instance we are in the first case. Then in (8), bits 24 and 27 are equal for $\mathrm{Enc}^{k_1}_{0\to i}(0) \oplus \mathrm{Dec}^{k'_2}_{10\to i}(0)$ and $\mathrm{Dec}^{k'_2}_{10\to i}(0)$, and null for the last term, thus the only nonlinear component of f has the same contribution on each side of the equation. On the rest f is linear, so we are done. □

In essence, there is only one multiplication on register A of KATAN32, between bits 24 and 27. By restricting ourselves to $13 - (27 - 24) = 10$ rounds, where 13 is the length of register A, we ensure that there is no nonlinear interaction between the bits of B_{k_1} dependent on k_1, and the bits of $A_{k_2'}$ dependent on k_2'. With register B the same computation yields $19 - \max(12 - 10, 8 - 3) = 14$ rounds. That is why we can build a biclique of length 10 on KATAN32. The same reasoning shows that we can build bicliques of length 5 on KATAN48, and 5 again on KATAN64.

Building Several Bicliques. Later on, we will want to use simultaneous matching (cf. Sect. 2.1). For this purpose, we need several distinct bicliques, and the previous proposition only gives us one. Fortunately, it is possible to build new distinct bicliques on the same model as that of Proposition 1, by adding parameters to Definition 2. There are several ways to proceed. In particular, it is possible to either modify the bits of A_{k_2} that do not actually depend on k_2, or those that do. We only describe the second option, as it is enough for our purpose.

Pick any arbitrary key k_P as parameter. In fact, only the first ten pairs of round keys derived from k_P will have an impact, so we have 20 degrees of freedom. Then define A_{k_2} and B_{k_1} by:

$$A_{k_2} = \mathrm{Dec}_{n\to 0}^{k_2' \oplus k_P}(0)$$
$$B_{k_1} = \mathrm{Enc}_{0\to n}^{k_1 \oplus k_P}(0)$$

and replace the recursion Eq. (6) in the proof by:

$$\forall i \leq 10, \quad \mathrm{Enc}_{0\to i}^{k}(A_{k_2}) = \mathrm{Enc}_{0\to i}^{k_1 \oplus k_P}(0) \oplus \mathrm{Dec}_{10\to i}^{k_2' \oplus k_P}(0)$$

The proof is exactly the same, except in (7), the contribution of k_P on the right-hand side cancels itself out.

Biclique Attack Against KATAN. In Sect. 4.3, we attacked 121 rounds of KATAN32. If we use four bicliques instead of four plaintext/ciphertext pairs, we gain an additional 10 rounds, as explained in the previous section. Meanwhile, the core of the attack remains the same, except we meet on b_{60} starting from round 10, instead of b_{50} starting from round 0. In particular, the complexity is unchanged. However we now require chosen plaintexts. Because the dimension of K_2' is 5, each biclique requires 2^5 chosen plaintexts, so the data requirements increase to $4 \cdot 2^5 = 2^7$. The attack covers 131 rounds with complexity $2^{77.5}$. In the same way, we can extend the previous attacks on KATAN48 and KATAN64 respectively to 114 rounds with 2^6 CP, and 107 rounds with 2^7 CP, both with complexity $2^{77.5}$.

4.6 Match Box

We now explain how the match box technique applies to KATAN32. Variants for KATAN48 and KATAN64 will be very similar. Assume we are meeting in the

middle on b_{62} (this will be the case in the final attack). The idea is that we are going to isolate round keys whose impact on the value of b_{62} when computing from the right can be evaluated with knowledge of only a few bits of information. We do not consider round keys whose impact is only linear however, since those can be ignored thanks to indirect matching.

When decrypting from the right, the value of b_{62} is (cf. (4)):

$$\begin{aligned} b_{62} &= a_{81} \oplus b_{73} \oplus b_{68} \cdot b_{70} \oplus b_{72} \cdot b_{77} \oplus rk_{163} \\ &= x_0 \oplus b_{68} \cdot b_{70} \qquad \text{with } x_0 = a_{81} \oplus b_{73} \oplus b_{72} \cdot b_{77} \oplus rk_{163} \end{aligned} \tag{9}$$

Let us further decompose b_{68} and b_{70} in the above formula as:

$$\begin{aligned} b_{68} &= x_1 \oplus rk_{175}, \quad (x_1 = a_{87} \oplus b_{89} \oplus b_{76} \cdot b_{74} \oplus b_{83} \cdot b_{78}) \\ b_{70} &= x_2 \oplus rk_{179}, \quad (x_2 = a_{89} \oplus b_{91} \oplus b_{78} \cdot b_{76} \oplus b_{85} \cdot b_{80}) \end{aligned}$$

Since $K_2 \oplus K_1' = K$, each round key rk_n may be written as $rk_n = rk_n^2 \oplus rk_n^{1'}$, with $rk_n^2 \in K_2$ and $rk_n^{1'} \in K_1'$. We define $\boldsymbol{r}$ as:

$$\begin{aligned} r_0 &= x_0 \\ r_1 &= x_1 \oplus rk_{175}^2 \\ r_2 &= x_2 \oplus rk_{179}^2 \end{aligned}$$

Putting everything together, $\boldsymbol{l}$ contains $a_0 = b_{62}$ computed from the left, as well as $k_1' \in K_1'$; and $\boldsymbol{r}$ is (r_0, r_1, r_2). The matching relation $\mathcal{R}(\boldsymbol{l}, \boldsymbol{r})$ witnessing the fact that computations from either side agree on b_{62} is:

$$\mathcal{R}(\boldsymbol{l}, \boldsymbol{r}) : \quad l_0 = r_0 \oplus (r_1 \oplus k_1'(rk_{175}^{1'})) \cdot (r_2 \oplus k_1'(rk_{179}^{1'}))$$

Note that this is merely a rewriting of (9), where the contribution of round keys 175 and 179 has been isolated, and then split in order to extract their K_1' component.

What we have gained in this example is that round keys 175 and 179 no longer need to be known when computing from the right. This decreases the dimension of K_2 by two, and thus spares a factor 2^2 when guessing its value. Moreover this gain can be spent in order to extend the attack to one more round. Indeed, we can now append one extra round at the end of the (reduced) cipher, and simply add the two bits of round key for that round into K_2. This increases the dimension of K_2 by two, back to its original value. Then the attack proceeds as before. In short, every time we are able to decrease the dimension of K_2 by two by needing less round keys in order to compute r_{62}, we can re-increase it in order to extend the attack to one more round.

Thus, to extend the attack further, we need only isolate the contribution of more round keys in (9), as we have done we round keys 175 and 179. The limit is the size of $\boldsymbol{r}$, which impacts the size of the match box table. For example, the next step would be to expand $b_{72} \cdot b_{77}$ in x_0, in the same manner we have

Table 3. Sizes of $\boldsymbol{r}$ sufficient to spare a certain number of round keys.

KATAN32 (starting from r_{62}):

$\lvert\boldsymbol{r}\rvert$	3	5	7	9	11	15	17	23	27	31	39	43	53	61	65	71	73
Round keys	2	5	7	9	10	12	13	16	18	20	23	24	28	30	32	34	35

KATAN48 (starting from r_{109}):

$\lvert\boldsymbol{r}\rvert$	5	9	17	21	25	27	35	43	55	63	73
Round keys	3	5	7	8	10	12	14	16	18	20	22

KATAN64 (starting from r_{153}):

$\lvert\boldsymbol{r}\rvert$	7	11	15	17	25	33	45	65	71
Round keys	3	5	7	8	10	12	14	17	18

previously expanded $b_{68} \cdot b_{70}$ in b_{62}. That is, we develop the expression of these two bits according to (4), making round keys rk_{183} and rk_{193} appear:

$$b_{72} = x_3 \oplus rk_{183}, \quad (x_3 = a_{91} \oplus b_{93} \oplus b_{78} \cdot b_{80} \oplus b_{82} \cdot b_{87})$$
$$b_{77} = x_4 \oplus rk_{193}, \quad (x_4 = a_{96} \oplus b_{98} \oplus b_{83} \cdot b_{85} \oplus b_{87} \cdot b_{92})$$

We now define a new 5-bit $\boldsymbol{r}$ by:

$$r'_0 = x_0 \oplus b_{72} \cdot b_{77} = a_{81} \oplus b_{73} \oplus rk_{163}$$
$$r_1 = x_1 \oplus rk^2_{175}$$
$$r_2 = x_2 \oplus rk^2_{179}$$
$$r_3 = x_3 \oplus rk^2_{183}$$
$$r_4 = x_4 \oplus rk^2_{193}$$

And the relation $\mathcal{R}(\boldsymbol{l}, \boldsymbol{r})$ becomes:

$$l_0 = r'_0 \oplus \left(r_1 \oplus k'_1(rk^{1'}_{175})\right) \cdot \left(r_2 \oplus k'_1(rk^{1'}_{179})\right) \oplus \left(r_3 \oplus k'_1(rk^{1'}_{183})\right) \cdot \left(r_4 \oplus k'_1(rk^{1'}_{193})\right)$$

It so happens that rk_{175} and rk_{179} (as well as rk_{183} and rk_{193}) only appear in one place in the development of b_{62}. Due to the diffusion of the cipher, later round keys will appear in several places in the expansion of b_{62}. As a result the "cost" for each new round key in terms of the increase in the size of $\boldsymbol{r}$ will grow. In order to choose which round keys to isolate, we have implemented a greedy algorithm that essentially adds the cheapest round key (in terms of the growth of $\boldsymbol{r}$) at each step. Results are shown on Table 3.

Note that this table only indicates the number of round keys spared. One can expect that every two round keys spared gains one round, but this is dependent on the two round keys being linearly independent of the rest of K_2. This in turn depends on which round keys are in K_2, i.e. which rounds are covered by K_2.

4.7 Final Attack

In this section, we describe the final version of the attack we propose against KATAN32, combining all components from the previous sections. We aim at having K_1 and K_2 both of dimension 77. Starting from round 10 after the biclique, this allows us to cover 62 rounds in the forward direction. This corresponds to meeting on b_{62} (i.e. position 9 of register B after 72 rounds). The number of rounds covered in the backwards direction will depend on the match box. We will ensure that in the end, the dimension of K_1' is 3.

Compression Table (cf. Sect. 3.3)**.** We have $|k_1'| = 3$, so $2^{|k_1'|} = 8$, which makes it worthwhile to use the compression technique. We want to build a compression table C converting a $\boldsymbol{r}$ of size greater than 8 into 8 f^n's. Note that we meet on a single bit, so there is only one line in (2), which is why we talk about f^n's and not f_i^n's. On the other hand, we will use simultaneous matching on three bicliques, but always on the same bit, so the conversion from $\boldsymbol{r}$ into the f^n's is the same for each pair: we only need one compression table.

As observed in Sect. 3.3, for each $\boldsymbol{r}$, the f^n's can be computed with $2^{|k_1'|}$ partial encryptions. Hence for KATAN32 we can choose $\boldsymbol{r} = 73$ (see Table 3), yielding a table of size 2^{76} in complexity 2^{76}. This spares 35 round keys in the decryption direction. In the end, we can begin the backwards computation from round 153.

Match Box (cf. Sect. 4.6)**.** We perform simultaneous matching on b_{62} for 3 distinct bicliques; $\boldsymbol{l'}$ contains the value of b_{62} computed from the left for each biclique, so $|\boldsymbol{l}| = 3$. Meanwhile $\boldsymbol{r}$ contains the 8 bits f^n computed from the right, again for each biclique, hence $|\boldsymbol{r}| = 8 \times 3 = 24$. This yields a match box table of size $2^{3^3+24+3} = 2^{54}$ in less than 2^{54} encryptions. Note that both the compression table and the match box table are absolute precomputations, in the sense that they do not depend on the actual plaintext/ciphertext pairs and need only be built once.

In the end, we attack 153 rounds: the first 10 are covered by the bicliques; the next 71 are the forward part of the meet-in-the-middle attack; the next 19 are covered by the match box; and the final 53 are the backwards part of the meet-in-the-middle attack. See the Appendix for the list of round keys involved in K_1 and K_2 and the list of round keys spared by using the match box.

Attack process.

- Precompute the compression table C.
- Precompute the match box M.
- For each partial key $k_\cap \in K_\cap$:
  - For each partial key k_1', knowing $k_1 = k_\cap \oplus k_1'$, compute b_{62} from the left for each biclique, and denote their concatenation by $\boldsymbol{l'}$.
    This yields a function $F : k_1' \to \boldsymbol{l'}$. Retrieve $M(F)$.
  - For each partial key k_2':
    - For each biclique, knowing $k_2 = k_\cap \oplus k_2'$, compute the 31-bit $\boldsymbol{r}$ from the right for that biclique. Convert it into 8 bits f^n through C.

  - Having done this for all 3 bicliques, the concatenation of the f^n's makes up the 24-bit $\boldsymbol{r}$ entry of the match box. Match k_2' with the k_1''s in $M(F)(\boldsymbol{r})$ to form candidate master keys.
- Test candidates master keys on 3 plaintext/ciphertext pairs as in a standard meet-in-the-middle attack. This should be done on the fly.

The overall attack complexity is:

$$2^{74} + 2^{54} + 3 \cdot \left(2^{77} \cdot \frac{62}{153} + 2^{77} \cdot \frac{81}{153} \right) + \sum_{i=0}^{2} 2^{77-32i} \approx 2^{78.5}$$

For KATAN48, aiming at the same complexity, we can cover $57 + 56 = 113$ rounds with the meet-in-the-middle section, 5 rounds with the bicliques, and an additional 11 rounds with the match box, for a total of 129 rounds. For KATAN64, the bicliques cover 5 rounds as well, the match box 9, and the meet-in-the-middle portion of the attack reaches $52 + 53 = 105$ rounds, for a total of 119 rounds. The complexity and data requirements are shown on Table 1.

5 Conclusion

In this paper, we presented a new technique to extend meet-in-the-middle attacks. This technique makes it possible to extend the middle portion of the attack with no increase in the overall complexity, but at the cost of significant precomputation. As such, it is a form of time/memory trade-off. We have applied this technique to the lightweight cipher KATAN, and significantly improve on previous results on this cipher.

Acknowledgments. The authors would like to thank Henri Gilbert for many helpful discussions, as well as Anne Canteaut and María Naya-Plasencia for their insightful remarks, including the idea of compression (Sect. 3.3).

References

1. Albrecht, M.R., Leander, G.: An all-in-one approach to differential cryptanalysis for small block ciphers. In: Knudsen, L.R., Wu, H. (eds.) SAC 2012. LNCS, vol. 7707, pp. 1–15. Springer, Heidelberg (2013)
2. Beaulieu, R, Shors, D., Smith, J., Treatman-Clark, S., Weeks, B., Wingers, L.: The Simon and Speck families of lightweight block ciphers. Cryptology ePrint Archive, Report 2013/404 (2013). http://eprint.iacr.org/2013/404
3. Bogdanov, A., Khovratovich, D., Rechberger, C.: Biclique cryptanalysis of the full AES. In: Lee, D.H., Wang, X. (eds.) ASIACRYPT 2011. LNCS, vol. 7073, pp. 344–371. Springer, Heidelberg (2011)
4. Khovratovich, D., Rechberger, C., Savelieva, A.: Bicliques for preimages: Attacks on Skein-512 and the SHA-2 family. In: Canteaut, A. (ed.) FSE 2012. LNCS, vol. 7549, pp. 244–263. Springer, Heidelberg (2012)

5. Bogdanov, A.A., Knudsen, L.R., Leander, G., Paar, C., Poschmann, A., Robshaw, M., Seurin, Y., Vikkelsoe, C.: PRESENT: An ultra-lightweight block cipher. In: Paillier, P., Verbauwhede, I. (eds.) CHES 2007. LNCS, vol. 4727, pp. 450–466. Springer, Heidelberg (2007)
6. Bogdanov, A., Rechberger, C.: A 3-subset meet-in-the-middle attack: Cryptanalysis of the lightweight block cipher KTANTAN. In: Biryukov, A., Gong, G., Stinson, D.R. (eds.) SAC 2010. LNCS, vol. 6544, pp. 229–240. Springer, Heidelberg (2011)
7. De Cannière, C., Dunkelman, O., Knežević, M.: KATAN and KTANTAN — A family of small and efficient hardware-oriented block ciphers. In: Clavier, C., Gaj, K. (eds.) CHES 2009. LNCS, vol. 5747, pp. 272–288. Springer, Heidelberg (2009)
8. Canteaut, A., Naya-Plasencia, M., Vayssière, B.: Sieve-in-the-middle: Improved MITM attacks. Cryptology ePrint Archive, Report 2013/324 (2013, to appear). http://eprint.iacr.org/2013/324
9. Guo, J., Peyrin, T., Poschmann, A., Robshaw, M.: The LED block cipher. In: Preneel, B., Takagi, T. (eds.) CHES 2011. LNCS, vol. 6917, pp. 326–341. Springer, Heidelberg (2011)
10. Isobe, T., Shibutani, K.: All subkeys recovery attack on block ciphers: extending meet-in-the-middle approach. In: Knudsen, L.R., Wu, H. (eds.) SAC 2012. LNCS, vol. 7707, pp. 202–221. Springer, Heidelberg (2013)
11. Knellwolf, S., Meier, W., Naya-Plasencia, M.: Conditional differential cryptanalysis of NLFSR-based cryptosystems. In: Abe, M. (ed.) ASIACRYPT 2010. LNCS, vol. 6477, pp. 130–145. Springer, Heidelberg (2010)
12. Knellwolf, S., Meier, W., Naya-Plasencia, M.: Conditional differential cryptanalysis of trivium and KATAN. In: Miri, A., Vaudenay, S. (eds.) SAC 2011. LNCS, vol. 7118, pp. 200–212. Springer, Heidelberg (2012)
13. Naya-Plasencia, M.: How to improve rebound attacks. In: Rogaway, P. (ed.) CRYPTO 2011. LNCS, vol. 6841, pp. 188–205. Springer, Heidelberg (2011)
14. Wei, L., Rechberger, C., Guo, J., Wu, H., Wang, H., Ling, S.: Improved meet-in-the-middle cryptanalysis of KTANTAN (Poster). In: Parampalli, U., Hawkes, P. (eds.) ACISP 2011. LNCS, vol. 6812, pp. 433–438. Springer, Heidelberg (2011)
15. Zhu, B., Gong, G.: Multidimensional Meet-in-the-Middle Attack and Its Applications to KATAN32/48/64. Cryptology ePrint Archive, Report 2011/619 (2011). http://eprint.iacr.org/2011/619

Appendix: Additional Details for the Attack on KATAN32

- K_1 contains the following 80 round keys (dimension 77):

{20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 93, 94, 96, 97, 98, 100, 104, 107, 113}.

The following keys have a linear contribution: {92, 99, 109, 124}.

- K_2 contains the following 81 round keys (dimension 77):

{213, 215, 217, 219, 221, 223, 225, 227, 229, 231, 233, 235, 237, 238, 239, 240, 241, 242, 243, 244, 245, 246, 247, 248, 249, 250, 251, 252, 253, 254, 255, 256, 257, 258, 259, 260, 261, 262, 263, 264, 265, 266, 267, 268, 269, 270, 271, 272, 273, 274,

275, 276, 277, 278, 279, 280, 281, 282, 283, 284, 285, 286, 287, 288, 289, 290, 291, 292, 293, 294, 295, 296, 297, 298, 299, 300, 301, 302, 303, 304, 305}.
The following keys have a linear contribution: {163, 185, 188, 198}.

- The following 35 round keys are spared by the match box:

{175, 179, 183, 187, 191, 193, 195, 196, 197, 199, 200, 201, 202, 203, 204, 205, 206, 207, 208, 209, 210, 211, 212, 214, 216, 218, 220, 222, 224, 226, 228, 230, 232, 234, 236}.

Collision Spectrum, Entropy Loss, T-Sponges, and Cryptanalysis of GLUON-64

Léo Perrin(✉) and Dmitry Khovratovich

University of Luxembourg, Walferdange, Luxembourg
{leo.perrin,dmitry.khovratovich}@uni.lu

Abstract. In this paper, we investigate the properties of iterative non-injective functions and the security of primitives where they are used. First, we introduce the Collision Probability Spectrum (CPS) parameter to quantify how far from a permutation a function is. In particular, we show that the output size decreases linearly with the number of iterations whereas the collision trees grow quadratically.

Secondly, we investigate the T-sponge construction and show how certain CPS and rate values lead to an improved preimage attack on long messages. As an example, we find collisions for the GLUON-64 internal function, approximate its CPS, and show an attack that violates the security claims. For instance, if a message ends with a sequence of 1 Mb (respectively 1 Gb) of zeros, then our preimage search takes time $2^{115.3}$ (respectively $2^{105.3}$) instead of 2^{128}.

Keywords: Random function · Collision probability spectrum · Collision tree · T-sponge · GLUON · Collision search

1 Introduction

Consider a function $g : \mathcal{S} \to \mathcal{S}$ where $\mathcal{S}$ is some finite space of size 2^N and suppose that it is not a permutation, i.e. that it has collisions. It is well known that for a random g the complexity of a collision search is of $2^{N/2}$ calls to g. However, not only the collision search complexity but also some related problems are not well studied when collisions have a certain structure, which is the case in several designs [1,2]. It might be clear that iterating such a function may lead to an entropy loss, but again, the scale of this loss and its implications on the security of stream ciphers and hash functions is not well known or underestimated. In this paper we introduce a particular parameter called the Collision Probability Spectrum (CPS), which is based on the number of solutions for the following equation

$$g(a + y) = g(a). \tag{1}$$

We study the CPS for several designs and show, as an illustration of our methodology, a preimage attack on the sponge-based lightweight hash function GLUON-64.

L. Perrin—The first author is supported by the core ACRYPT project from the *Fond National de la Recherche* (Luxembourg).

C. Cid and C. Rechberger (Eds.): FSE 2014, LNCS 8540, pp. 82–103, 2015.
DOI: 10.1007/978-3-662-46706-0_5

Related work. Bellare and Kohno [3] studied how the number of preimages to $g(a)$ affects the complexity of the collision search with the notion of *balance* of a function. In [4], Flajolet and Odlyzko studied several characteristics of random mappings, in particular the distribution of preimage sizes, the cycle size and the size of the iterated image. Their result was applied by Hong and Kim [5] to the MICKEY [1] cipher. Indeed, they found experimentally that the size of the iterated images of this function was essentially the size of the space divided by the number of iterations, a behavior which they showed experimentally to correspond to the prediction of Flajolet et al. However, the resulting attacks were found to be less efficient than the simple collision search [6], though they allow a time/memory trade-off.

Overview of our results. We introduce the *Collision Probability Spectrum* parameter which quantifies how many solutions Eq. (1) has on average and investigate its consequences over the iterated images and preimages of $\mathcal{S}$ by g. We assume that the composition of two such functions has certain properties, which is formalized as an independence assumption. For a large class of mappings two important facts are proved in Theorem 2 (a reader may refer to Fig. 1):

- First, the size of the iterated image of g is inversely proportional with the number i of iterations:
$$|g^i(\mathcal{S})| \sim \frac{|\mathcal{S}|}{\frac{\kappa}{2} \cdot i},$$
where κ depends on the CPS and where i has to be smaller than $\sqrt{|\mathcal{S}|}$ — otherwise, the result does not hold because of the cycles in the functionnal graph.
- Second, an element $y \in g^i(\mathcal{S})$ is the root of a *collision tree* consisting in elements x_l such that any of $g(x_l), g^2(x_l), \ldots, g^i(x_l)$ is equal to y. The average size of this tree is ν_i:
$$\nu_i \sim \frac{\kappa}{4} \cdot i^2,$$
with the same restriction on i: $i < \sqrt{|\mathcal{S}|}$.

Then we discuss the security of the T-sponge construction provided the CPS of the update function. We amend the collision search bound in the flat sponge claim [7]:

$$P = \frac{Q^2}{2^{c+1}} \cdot \left(1 + \frac{\kappa - 1}{2^r}\right),$$

where c is the capacity and r is the rate of the sponge.

Next, in Theorem 6, we show for small r an improved preimage attack with complexity

$$2^c \cdot 2^{r+2}/(\kappa z),$$

where z is the number of zero bytes in the end of the hashed message (actually, any constant suffices).

Finally, we construct an attack on GLUON-64. Aided with a SAT-solver, we find collisions for the update function and demonstrate a preimage attack of

complexity 2^{105} for a message ending with 1 GByte of zeros, which violate the claimed preimage resistance level of 128 bits.

Structure. This paper is organized as follows. We introduce our theoretical framework in Sect. 2 and discuss its application to existing primitives. We investigate the security of T-sponge against collision and preimage search in Sect. 3. Finally, in Sect. 4, we obtain inner-collisions of the update function of GLUON-64 with the help of a SAT-solver and show a preimage attack. For the sake of concision, the proofs are moved to Appendix A.

Notations. We denote by $|E|$ and $\#E$ the size of a set E, by $\mathbb{P}[\omega]$ the probability of an event ω and by $a \xleftarrow{\$} E$ the fact that a is drawn uniformly at random from a set E.

2 Theoretical Framework

In this section we introduce a model of random functions and highlight its difference with the usual approach. We then give several properties of the (iterated) images and preimages of an element by such functions.

2.1 Collision Probability Spectrum and Function Model

Definition 1 (Collision Probability Spectrum). *Let $\mathcal{S}$ be a finite space and let $g : \mathcal{S} \to \mathcal{S}$ be a function. We denote $\mathfrak{c}_k$ the probability that the following equation has exactly k solutions for $a \in \mathcal{S}$ picked uniformly at random in $\mathcal{S}$:*

$$g(a + x) = g(a), \tag{2}$$

so that

$$\mathfrak{c}_k = \mathbb{P}[\#\{x \in \mathcal{S}, g(a + x) = g(a)\} = k \mid a \xleftarrow{\$} \mathcal{S}] \tag{3}$$

The solutions x of this equation are called vanishing differences. *The set of all the elements a of $\mathcal{S}$ such that Eq. (2) has exactly k solutions is denoted V_k. Finally, the set $\mathfrak{C} = \{\mathfrak{c}_k\}_{k \geq 1}$ is the* Collision Probability Spectrum *(CPS) of g.*

An equivalent definition of the CPS is that it is the probability distribution of the number of solutions of Eq. 2. We now make some remarks regarding these definitions:

- Since 0 is always a solution of Eq. (2), we have that $\mathfrak{c}_0 = 0$.
- If g is a permutation, then $\mathfrak{C}(g) = \{\mathfrak{c}_1 = 1, \mathfrak{c}_k = 0 \text{ for } k > 1\}$.
- The input space can be partitioned in the following way: $\mathcal{S} = \bigcup_{k=1}^{\infty} V_k$. Furthermore, the output space can be partitioned as $g(\mathcal{S}) = \bigcup_{k=1}^{\infty} g(V_k)$. This is also a disjoint union. Indeed, $y \in g(V_k)$ has exactly k preimages, by definition.
- The size of $g(V_k)$ is $|g(V_k)| = |\mathcal{S}| \cdot \mathfrak{c}_k/k$ because to each element in $g(V_k)$ correspond k elements in V_k (see Fig. 2). As a consequence,

$$|g(\mathcal{S})| = |\mathcal{S}| \cdot \sum_{k=1}^{\infty} \frac{\mathfrak{c}_k}{k}$$

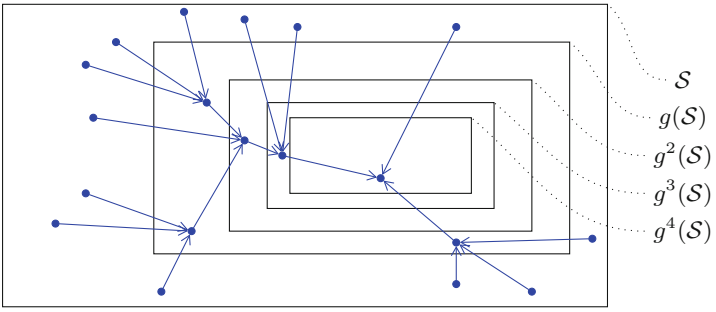


Fig. 1. Collision trees and output shrinkage of iterative non-injective functions. The dots represent elements of $\mathcal{S}$ and there is an edge from x to y if $g(x) = y$. Here, $g(a + x) = g(a)$ always has exactly 3 solutions.

2.2 Composition of Functions with Known CPS

The most interesting application of our theory is the properties of iterative constructions where the iterated function has some known CPS. However, to make meaningful and correct statements about composition of such functions, some independency must be assumed.

Assumption 1 (Independence Assumption). *Let g be a function with* CPS *$\mathfrak{C}$. Then there is no correlation between the events $x \in V_j$ and $g(x) \in V_k$ for any j, k.*

This assumption, as we will see, holds for a few (but not for all) real primitives. For the rest of the paper, we implicitly assume that it holds unless stated otherwise.

Definition 2. *Suppose g is a function on $\mathcal{S}$. Then ℓ_i defined as*

$$\ell_i = \frac{|\mathcal{S}|}{|g^i(\mathcal{S})|}$$

is called the shrinking ratio *of g.*

Our first theorem allows to compute the shrinking ratio of the composition of two functions with given CPS.

Theorem 1. *Let g and g' be functions with* CPS *$\mathfrak{C} = \{\mathfrak{c}_k\}_{k\geq 1}$ and $\mathfrak{C}' = \{\mathfrak{c}'_k\}_{k\geq 1}$, respectively. Then the shrinking ratio of the composition $g \circ g'$ is computed as follows:*

$$\ell_1(g \circ g') \;=\; \Big(\frac{1}{\ell_1} - \sum_{k=1}^{\infty} \frac{\mathfrak{c}_k}{k}\big(1 - \frac{1}{\ell'_1}\big)^k\Big)^{-1}.$$

In particular, when $g' = g^i$:

$$\ell_{i+1} = \Big(\frac{1}{\ell_1} - \sum_{k=1}^{\infty} \frac{\mathfrak{c}_k}{k}\big(1 - \frac{1}{\ell_i}\big)^k\Big)^{-1}.$$

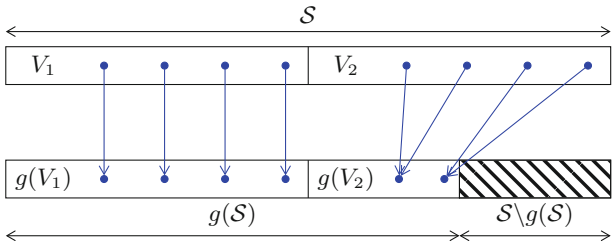


Fig. 2. The effect of g with CPS $\{c_1 = c_2 = 1/2\}$ on $\mathcal{S}$.

A detailed proof is given in Appendix A.1 but we provide a high level view of its structure.

Proof Sketch 1. We consider an element $x_0 \in g'(\mathcal{S})$ such that there exists $\{x_0, ..., x_k\}$ with $g(x_l) = g(x_0)$, i.e. $x_0 \in V_{k+1}$. The number of solutions of $g(x_0 + x) = g(x_0)$ in $g'(\mathcal{S})$ for x_0 drawn at random in $g'(\mathcal{S}) \cap V_{k+1}$ follows a binomial distribution $(m, k, 1/\ell'_1)$ as $x_l \in g'(\mathcal{S})$ with probability $|g'(\mathcal{S})|/|\mathcal{S}| = 1/\ell'_1$.

Using this observation, we can compute the probability that $g(x_0+x) = g(x_0)$ has m solutions in $g'(\mathcal{S})$ for all m: if it has $m+1$ solutions, then it must be that $x_0 \in V_{k+1}$ and that only m of the k non zero solutions "made it" to $g'(\mathcal{S})$. Then, we deduce the size of the image of $g'(\mathcal{S})$ by g, i.e. we give an expression of $\ell_1(g \circ g')$.

Using this theorem, we can give the asymptotic behavior of ℓ_i and of the size of the collision trees as i increases while remaining small enough so that $g(x)$ is not on a cycle. The results stated below have been checked experimentally on the functions for which the independence assumption presumably holds. We need two more definitions.

Definition 3. *Suppose g is a function on $\mathcal{S}$ with* CPS $\mathfrak{C}$. *Then*

- $\kappa(\mathfrak{C})$ *is the* collision average of g — *the average number of non-zero solutions of Eq. (2):* $\kappa = \sum_{k\geq 1} \mathfrak{c}_k \cdot k - 1$.
- $\nu_i(g)$ *is the* average tree size of g — *the average number of elements in a collision tree rooted in $g^i(\mathcal{S})$. Formally, it is the average number of pairs $(x_l, k_l) \in \mathcal{S} \times [1, i]$ such that $g^{k_l}(x_l) = y$ for $y \in g^i(\mathcal{S})$.*

Theorem 2. *Let g be a function with* CPS $\mathfrak{C}$, *then for $i < \sqrt{|\mathcal{S}|}$ the shrinking ratio and the average tree size are approximated as follows for large enough i:*

$$\ell_i \sim \frac{\kappa}{2} \cdot i,\ \nu_i \sim \frac{\kappa}{4} \cdot i^2.$$

Proof Sketch 2. The asymptotic behaviour of ℓ_i can be deduced by using Theorem 1 with $g' = g^i$ and then using the finite expansion of $(1 - 1/\ell_i)^k$ to see that $\ell_{i+1} = \ell_i + \kappa/2$. For ν_i, we simply note that $\nu_i = \sum_{k=1}^{i} \ell_k$. More details are given in Appendix A.2.

Finally, we define the following quantities in the same way as Flajolet et al. [4].

Definition 4. *We call* cycle length *and* tail length*, denoted respectively μ and λ, the average smallest values such that*

$$g^{\lambda}(x) = g^{\lambda+\mu}(x)$$

for x drawn uniformly at random in $\mathcal{S}$.

Experiments (see Appendix A.3) lead us to the following conjecture.

Conjecture 1. Let g be a function of $\mathcal{S}$ with CPS $\mathfrak{C}$. Experimentally, we found the following values for the tail length λ and the cycle length μ:

$$\lambda \sim \sqrt{\frac{\pi}{8 \cdot \kappa}|\mathcal{S}|},\ \mu \sim \sqrt{\frac{\pi}{8 \cdot \kappa}|\mathcal{S}|}.$$

2.3 Independence Assumption in Practice

In this Section, we investigate some results from the literature about particular functions and see how relevant our model is. A summary of this Section is given in Table 1.

Table 1. Characteristics derived from the CPS of some functions.

Function	κ	ℓ_1	ℓ_i/i	ν_i/i^2	Reference for the CPS
MICKEY's update function	0.625	1.407	$2^{-1.7}$	$2^{-2.7}$	[8]
Random mapping	1	1.582	2^{-1}	2^{-2}	[4]
GLUON-64's update function	6.982	3.578	$2^{1.8}$	$2^{0.8}$	Sect. 4.2

Random Mappings. The authors of [4] study random mappings and give the probability that some $x \in \mathcal{S}$ has r preimages by a random mapping g. From this we deduce that the CPS of a random function is given by the Poisson distribution with $\lambda = 1$:

$$\mathfrak{C} = \{e^{-1}/(k-1)!\}_{k \geq 1}.$$

Our framework implies

$$\kappa = 1 \quad \text{and} \quad \ell_1 = 1/(1-e^{-1}) \quad \text{and} \quad \ell_{i+1} = 1/\big(1 - \exp(-1/\ell_i)\big)$$

which fits the results of [4](see also Appendix A.1). The authors of [5] observed that

$$\log_2(\ell_i) \approx \log_2(i) - 1,$$

which also corresponds to $\kappa = 1$. Finally, the trail and cycle length given in Conjecture 1 match those predicted by [4] if we replace κ by 1.

A5/1. The update function of A5/1 does not satisfy the independence assumption. The author of [2] computed its CPS and established that

$$\ell_1 = 1.6, \quad \kappa = 1.25,$$

If the assumption held, then the probability for an element in $\mathcal{S}$ to be in $g^{100}(\mathcal{S})$ would be about 2^{-6}, which is very different from the $2^{-2.5}$ actually observed by Biryukov et al. [9]. The reason is that the update function A5/1 may keep one of its three LFSR's untouched, which means that $x \in V_j$ and $g(x) \in V_k$ are *not* independent events in its case.

MICKEY. The update function of the MICKEY [1] stream-ciphers (v1 and v2) fits our model. Hong and Kim [5] performed some experiments on the first version of MICKEY and, in particular, estimated the size of $g^{2^k}(\mathcal{S})$ for several values of k. Their results are coherent with our model. For instance, they observed that $\log_2(\ell_i)$ (which they denote by $\overline{EL}(f^i)$) is approximated as

$$log_2(\ell_i) \approx \log_2(i) - 1.8$$

The constant term 1.8 implies

$$\kappa/2 \approx 2^{-1.8}.$$

In turn, from the CPS values computed in [8](actually, the values $\mathfrak{c}_k/k$) we obtain the theoretical value

$$\kappa = 0.625,$$

which corresponds to a difference of about 7 % with the experiments in [5].

3 Improved Collision and Preimage Search

In this section we explore generic collision and preimage search methods in their application to functions with fixed collision spectrum.

3.1 Basic Collision Search

First, we reformulate the result from Bellare and Kohno [3] with our notation.

Theorem 3 *[3]. Let g be a function with CPS $\mathfrak{C}$, and let κ be its collision average. Then the birthday collision attack on g requires about*

$$Q = \sqrt{\frac{|\mathcal{S}|}{\kappa + 1}}. \tag{4}$$

queries to g.

The original paper [3] used the parameter *balance* of g, denoted $\mu(g)$, which is computed as

$$\mu(g) = \log_{|g(\mathcal{S})|}\left(\frac{|\mathcal{S}|^2}{\sum_{y\in\mathcal{S}}|g^{-1}(y)|^2}\right) \tag{5}$$

If we know the CPS of g, the balance can be expressed as follows:

$$\mu(g) \approx 1 - \frac{\log_2\left(\sum_{k=1}^{\infty} k\cdot\mathfrak{c}_k\right) + \log_2\left(\sum_{k=1}^{\infty}\mathfrak{c}_k/k\right)}{\log_2\left(|\mathcal{S}|\right)}. \tag{6}$$

If Conjecture 1 holds, then the memory-less collision search based on Floyd's cycle finding algorithm should be $\sqrt{\kappa}$ as fast as in the case of a random function.

3.2 Collision Attacks on T-sponge

Now we demonstrate that the entropy loss because of collisions in the T-sponge construction, though observable, can be mitigated by a large rate parameter.

Sponge Construction. The sponge construction [7] is characterised by its *rate* r, its *capacity* c and its update function g. It is based on an internal state of size $r + c$ where, at each round, r bits of the message are xor-ed. Then the sponge alternates the application of g function with the message injection until the message has been entirely *absorbed*. The digest is then *squeezed* by extracting r bits of the internal state and applying the update function to the internal state again. This is repeated as many times as necessary to obtain a digest of desired length. A representation of a sponge is given in Fig. 3. The sponge-based hash function is indifferentiable from a random oracle in the random-function model up to $2^{c/2}$ queries to g [10]. If g is not a permutation, the sponge is called *transformative sponge* or T-sponge.

We denote a sponge-based hash function by $H : \mathbb{F}_2^* \to \mathbb{F}_2^{rj}$, the internal state space by $\mathcal{S} = \mathbb{F}_2^{r+c}$, and the update function by $g : \mathcal{S} \to \mathcal{S}$.

Collision Search in T-sponge. The following theorem shows that to get a significant speed-up in the collision search, the collision average κ should be at least of the same magnitude as 2^r.

Theorem 4. *Let g be a random mapping from $\mathbb{F}_2^{r+c}$ with* CPS $\mathfrak{C}$. *Let H be a* T*-sponge of capacity c and rate r updated with g. Then the probability of success of a brute-force collision attack on H is*

$$P = \frac{Q^2}{2^{c+1}}\cdot\left(1 + \frac{\kappa - 1}{2^r}\right)$$

where Q is the number of queries to g.

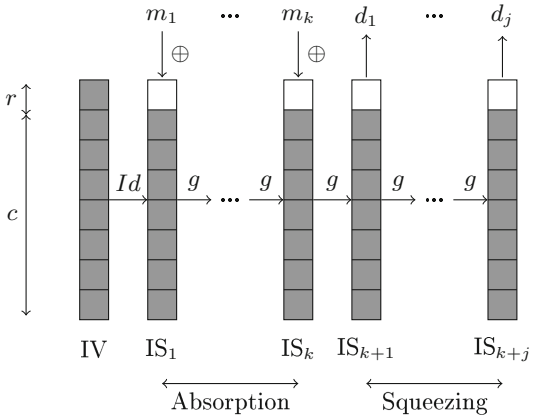


Fig. 3. Principle of a sponge construction. The message m is sliced into k blocks of r bits and "absorbed" during the first phase. Then, the j blocks of size r constituting the digest h are "squeezed".

The proof of Theorem 4 is in Appendix A.4. For a completely random mapping we have $\kappa = 1$, so that the theorem has the same form as in [7].

Nevertheless, since in practice the functions are not drawn at random from the set of all functions, it is of interest to be able to predict the effect of their properties over the security they provide. In particular, we see that a function with $\kappa > 1$ does not exactly provide $c/2$ bits of security against birthday attacks. Such functions can be found in real cryptographic primitives, see Sect. 4. However, we also immediately see that this effect is small since typical value of κ are of order of magnitude 1, 10 being already rather bad, while 2^r is at least in the hundreds. The designers of a T-sponge need not really worry about the number of collisions in the update function *if the rate is high enough.*

3.3 Improved Preimage Attack

Principle of the Iterated Preimage Attacks. Consider a set $\{g_k\}_{k\in\mathcal{K}}$ of random functions of $\mathcal{S}$ with CPS's $\{\mathfrak{C}_k\}_{k\in\mathcal{K}}$ and a fixed starting point $x_0 \in \mathcal{S}$ and let $\{k_1, ..., k_l\}$ be a set of l elements of $\mathcal{K}$. We call *keyed walk* the sequence

$$\big(x_1 = g_{k_1}(x_0),\ x_2 = g_{k_2}(x_1),\ ..., x_l = g_{k_l}(x_{l-1}) = d\big).$$

and it can for instance correspond to the successive values of the internal state of a T-sponge or of a Davies-Meyer based Merkle-Damgård hash function as we discuss in the next sections. Consider a keyed walk directed by a sequence $\{k_1, k_2, ..., \alpha, \alpha, ..., \alpha\}$ ending with z copies of the same symbol α. Then, intuitively, much entropy will have been lost because of the z iterations of g_α so that it should be easier to find a second sequence of keys leading to the same final value. This is formalized by the next theorem and a graphical representation of the phenomena we use is given in Fig. 4.

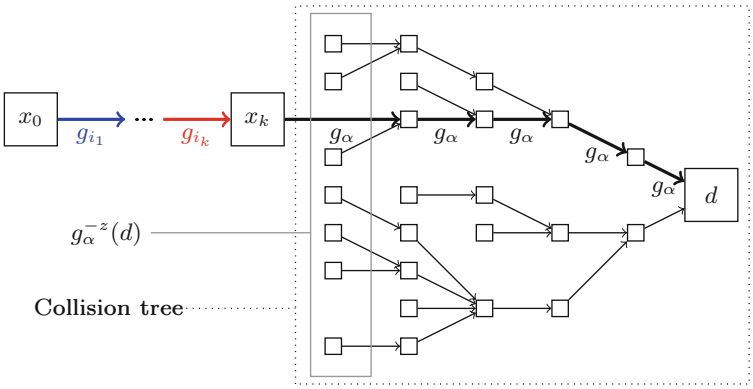


Fig. 4. The two targets of the iterated preimage attacks on d where d is in $g_\alpha^z(\mathcal{S})$ and $z = 5$. Different colors correspond to different function calls (Color figure online).

Theorem 5. *Let $\{g_k\}_{k\in\mathcal{K}}$ be a set of random mappings of $\mathcal{S}$ with* CPS*'s $\{\mathfrak{C}_k\}_{k\in\mathcal{K}}$ and consider a sequence $\{k_1, k_2, ..., \alpha, ..., \alpha\}$ of l keys from $\mathcal{K}$ ending with z identical keys α. Given the final value d of the corresponding keyed walk, the value of α and the number z, it is possible to find, for large enough z:*

1. *a keyed walk ending in d in time $|\mathcal{S}| \cdot 4/(\kappa z)$,*
2. *a keyed walk ending in d after precisely z calls to g_α in time $|\mathcal{S}| \cdot 2/\kappa$.*

where κ is the collision average of $\mathfrak{C}_\alpha$.

Proof. Let d be the final element in the walk. From the structure of the walk, we know that $d \in g_\alpha^z(\mathcal{S})$. Using Theorem 2, we know that there are $(\kappa/2)\cdot z$ elements in $g_\alpha^{-z}(d)$ and that the collision tree rooted at d contains $(\kappa/4) \cdot z^2$ elements. Therefore, such an element of $g_\alpha^{-z}(d)$ is found with probability $\kappa \cdot z/(|\mathcal{S}| \cdot 2)$ and an element in the collision tree with probability $\kappa \cdot z^2/(|\mathcal{S}| \cdot 4)$.

However, in both cases, we need to call g_α z times to know if the element we picked at random is mapped to d after exactly z iterations of g_α (first case) or at most z iterations (second case). Therefore, finding an element in the collision tree (first case) requires $|\mathcal{S}| \cdot z/(\kappa \cdot z^2/4) = |\mathcal{S}| \cdot 4/(\kappa z)$ calls to g_α and finding an element in $g_\alpha^{-z}(d)$ requires $|\mathcal{S}| \cdot z/(\kappa \cdot z/2) = |\mathcal{S}| \cdot 2/\kappa$. □

Note that these attacks can be generalized to the case where the end of the message is periodic instead of constant, i.e. if it ends with z copies of $(\alpha_1, \alpha_2, ..., \alpha_p)$. We simply need to replace g_α by $g' = g_{\alpha_1} \circ ... \circ g_{\alpha_p}$. The κ involved in the complexity computations is then that of g', i.e. $\sum_{i=1}^{p} \kappa_i$ where κ_i is the collision average of g_{α_i} (see Lemma 2 in Sect. 5). The constraint on z being large is only such that we can assume that z has the asymptotical behaviours described in Theorem 2.

Application to a T-sponge. Hashing a message with a T-sponge can be seen as performing a keyed walk where the keys are the message blocks of length r and the initial value x_0 is the all-zero vector. The function g_k is $g_k(x) = g(x \oplus k)$ where k is set to zero after its r first bits and g is the update function of the T-sponge. Clearly, g_k has the same CPS as g.

While the flat sponge claim provides a good description of the security offered by a sponge (be it a T-sponge or a P-sponge) against collision search and, for P-sponge, against second preimage search, there is a gap between the number of queries it allows and the best algorithm known for preimage search. In particular, there is to the best of our knowledge no algorithm allowing a preimage search with complexity below 2^c calls to the sponge function.[1] Theorem 6 bridges the gap between the $2^{c/2}$ bound of the flat sponge claim and the 2^c bound for preimage search by applying Theorem 5 to the T-sponge structure.

Theorem 6. *Let H be a T-sponge with update function g, and let κ be the collision average of g. Let M be a message such that its last z injections to the sponge are identical. Then a preimage to $H(M)$ can be found with complexity*

$$2^c \cdot 2^{r+2}/(\kappa z)$$

Such messages can be quite common. For instance, the last z calls of g can be blank calls for the sole purpose to slow down the hashing as suggested by NIST [12].[2] Such an attack can be prevented by setting an upper-bound of about $2^{r+2}/\kappa$ for the length of the message which in turn means that r has to be high in a T-sponge.

Similarity to the Herding Attack. This attack was introduced in [13] and is also refered to as the Nostradamus attack. In a herding attack, an attacker commits to a digest d and, when given a challenge P, has to find a suffix S such that $H(P||S) = d$. To achieve this she builds, during an offline phase, a so-called *diamond structure* which is essentially a binary collision tree with 2^ℓ nodes and rooted at d. The nodes of the tree contain the value of the internal state as well as the message block which needs to be absorbed to go to its child. During the online phase, she uses the diamond to find efficiently the suffix S: all she has to do is find a way to reach any of the $2^{\ell+1} - 1$ nodes in the diamond from the given starting point.

Application to a Merkle-Damgård Construction. When a block cipher is used in Davies-Meyer mode to build a Merkle-Damgård-based hash function,

[1] This case corresponds to the case where the attacker inverts the squeezing operations in time 2^c to retrieve the last internal state of the sponge before the squeezing and then uses a meet-in-the middle approach to find a valid message leading to this internal state in time $2^{c/2}$ (see [11]). Furthermore, this second step cannot be carried out in the case of a T-sponge since the update function cannot be inverted.

[2] Here, we consider that the message hashed is of a length equal to a multiple of r to begin with, so that the padding consisting in appending a one to the end of the message can be seen as part of the squeezing.

the successive chaining values $h_i \in \mathcal{S}$ are obtained from the previous one and the i-th message block: $h_i = E_{m_i}(h_{i-1}) \oplus h_{i-1} = g_{m_i}(h_{i-1})$. Because of the feedback of h_{i-1}, we do not expect g_k to be a permutation and, therefore, expect such a construction to be vulnerable to iterated preimage attacks. The padding used for Merkle-Damgård constructions usually takes into account the length of the message so that we need a message of the same length. Therefore, it is not enough to aim at an element in the collision tree, we need to find an element which is precisely in $g_\alpha^{-z}(d)$ so that a preimage search requires $2^{N+1}/\kappa$: if the CPS of g_k is such that $\kappa > 2$ then the iterated preimage attack is more efficient than brute-force. Furthermore, if there is an efficient way around the padding (e.g., with expandable messages [14]), the efficiency becomes $2^{N+2}/(\kappa z)$ where N is the size of the internal state of the hash function.

4 Preimage Attack on GLUON-64

4.1 The GLUON Family of Hash Functions

Introduced in [15], the GLUON family of hash functions consists of three members corresponding to different security levels: GLUON-64, GLUON-80 and GLUON-112. They have a T-sponge structure and have characteristics summarized in Table 2.

Table 2. Characteristics of the hash functions of the GLUON family.

name	rate r	capacity c	collision search	preimage search
GLUON-64	8	128	2^{64}	2^{128}
GLUON-80	16	160	2^{80}	2^{160}
GLUON-112	32	224	2^{112}	2^{224}

The function g used to update the internal state has the same structure in the three cases. It can be seen as a stream-cipher based on a Feedback with Carry Shift Register (FCSR). The concept of FCSR has evolved through time as the first stream-cipher [16] based on a component bearing this name got broken [17]. When we talk about FCSR in this paper, we refer to the last version of this structure, i.e. the one used in X-FCSR v2 [18] and, of course, GLUON. For example, the algebraic representation of the FCSR used by GLUON-64 is given in Fig. 5.

A FCSR is made of w cells of r bits. Each cell may be on the receiving end of a feedback. If the cell i receives no feed-backs, then its content at time $t+1$ is the content of the cell of index $i+1$ at time t. Consider now that the cell i receives a feedback. This cell contains an additional memory to store the value of the carry from one clock to the next. The content of the cell at time t is denoted m_i^t and that of the carry register c_i^t. Since it receives a feedback there are a cell of index j and a value of shift s (possibly equal to zero) such that:

$$\begin{aligned} m_i^{t+1} &= m_{i+1}^t + \left(m_j^t \ll s\right) + c_i^t \\ c_i^{t+1} &= m_{i+1}^t \cdot \left(m_j^t \ll s\right) + m_{i+1}^t \cdot c_i^t + \left(m_j^t \ll s\right) \cdot c_i^t \end{aligned}$$

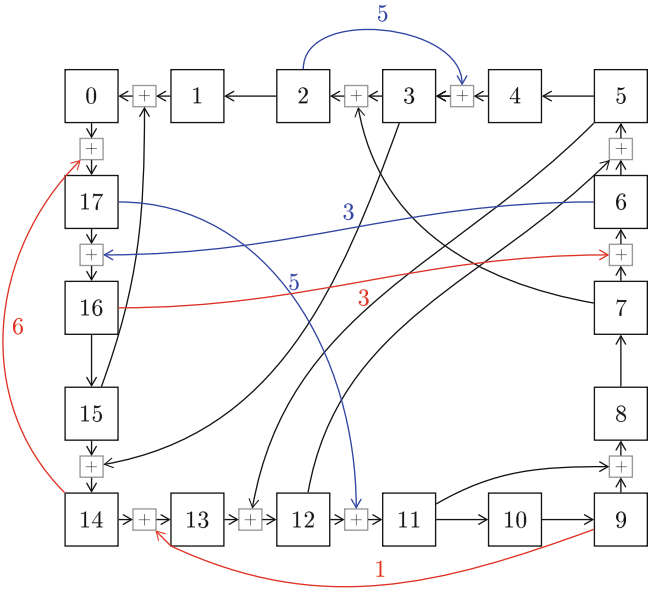


Fig. 5. Algebraic representation of the FCSR used in GLUON-64. Blue arrows correspond to feed-backs shifted to the right and red ones to the left. The value of the shift is given by the label of the arrow (Color figure online).

where "$\ll s$" is a C-style notation for the shift of the content of a cell by s bits to the left (or $|s|$ bits to the right if $s \leq 0$) and $+$ and $\cdot$ are respectively the bitwise addition and multiplication in $\mathbb{F}_2^r$.

The update function of every member of the GLUON family is made of three steps: padding of the input and loading in a FCSR (pad), clocking of the FCSR (ρ) and filtering Φ. We describe these steps separately.

Pad. The internal state of the sponge is of size $r(w-1)$, so that $r(w-1) = r+c$. The padding consists simply in appending a block of r bits all equal to one to the end of the internal state. The rw bits thus obtained are then loaded in a FCSR with an internal state made of w cells of size r. All the carries of the FCSR are set to zero. This operation is denoted $\text{pad} : \mathbb{F}_2^{r+c} \to \mathbb{F}_2^{rw} \times \mathbb{F}_2^{rw}$ as the output is made of the main register *and* the carry register of the FCSR.

ρ^{d+4}. The FCSR is clocked $d+4$ times. One clocking is denoted $\rho : \mathbb{F}_2^{rw} \times \mathbb{F}_2^{rw} \to \mathbb{F}_2^{rw} \times \mathbb{F}_2^{rw}$.

Φ. The output of g is extracted r bits by r bits using the following method: fixed words of the main register are rotated and then xor-ed to obtain r bits and then the FCSR is clocked. This operation is repeated $w-1$ times so as to have $r(w-1) = r+c$ bits of output. The action of clocking the FCSR $w-1$ times while filtering r bits each time is denoted $\Phi : \mathbb{F}_2^{rw} \times \mathbb{F}_2^{rw} \to \mathbb{F}_2^{r+c}$.

Overall, g is equal to $\Phi \circ \rho^{d+4} \circ \text{pad}$. The function pad is a bijection and we shall consider that the restriction of Φ over the set of the pairs main register/carry

register reachable after $d + 4$ calls to ρ starting in the image of pad is collision-free. The designers of GLUON claim:

> After a few iterations from an initial state, the automaton is in a periodic sequence of states of length P. The average number of required iterations to be in such a state is experimentally less than $\log_2(n)$, where n is the size of the main register [...] This leads to consider a function $[g]$ which is really close to a permutation from $\{0,1\}^b$ into itself because the surjective part of the construction is really limited once the function $[g]$ acts on the main cycle.

However, what happens during these first rounds, *before* the main cycle is reached? It is possible to encode the equation

$$(\rho^k \circ \mathrm{pad})(a + x) = (\rho^k \circ \mathrm{pad})(x) \tag{7}$$

for a fixed a into a CNF-formula solvable by a SAT-solver as long as k is not too big, say 10. The encoding is fairly straight-forward and we shall not go into the details for the sake of concision. Note that solving the equation $(\rho^k \circ \mathrm{pad})(x) = y$ using a SAT-solver is fast, meaning that it is possible to run a FCSR backward. However, we tried encoding the filtering so as to solve $(\Phi \circ \rho^k \circ \mathrm{pad})(x) = y$ but no SAT-solver was able to handle the corresponding CNF-formula — we killed the process after 1 week of running time for GLUON-112 (simplest filtering of the three), and for $k = 1$ instead of $k = d + 4 = 46$.

We solved (7) for many values of a and for $k = 10$ for each member of the GLUON family. While no non-zero solutions were found for any a for GLUON-80 and GLUON-112, it turns out that (7) has many solutions for GLUON-64. We used Algorithm 1 to find to which V_k any element $a \in \mathcal{S}$ belongs by enumerating all the values of x such that (7) holds. It works by solving (7) for x, thus (possibly) finding a solution x_1; then solving (7) with the constraint that the solution must be different from x_1, thus (possibly) finding x_2, etc. until no more solutions can be found. If there are k such $x \neq 0$, then a is in V_{k+1}.

4.2 CPS and Preimage Attack on GLUON-64

We ran Algorithm 1 for GLUON-64 on 24,000 different elements chosen uniformly at random in $\mathcal{S} = \mathbb{F}_2^{r+c}$. This allowed us to approximate the CPS of the update function. Our results are Fig. 6.

We deduce that $\mathfrak{c}_1 = 0.065$, $\ell_1 = 3.578$ and $\kappa = 6.982$ which are much worse than what one should expect from a random function, namely $\mathfrak{c}_1 = e^{-1} \approx 0.368$, $\ell_1 = 1/(1 - e^{-1}) \approx 1.582$ and $\kappa = 1$. This means that finding a preimage in a scheme equivalent to appending z identical words at the end of the message has a complexity of $2^{136+2}/(6.982 \cdot z) = 2^{128} \cdot (146.7/z)$. For $z > 147$, this is more efficient than classical brute-force. The complexities for some values of $z < 2^{(r+c)/2} = 2^{68}$ are given in Table 3 (Fig. 7).

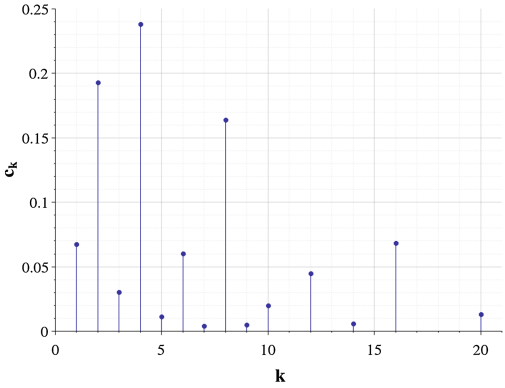


Fig. 6. Approximation of the CPS of the function used by GLUON-64 to update its internal state over 24,000 random elements of $\mathbb{F}_2^{136}$. Note that non-zero $\mathfrak{c}_k$ were observed well after $k = 20$.

Table 3. Complexity $\mathcal{C}$ of a preimage search for $d = H(m)$ where H is GLUON-64 and m is unkown except for the z identical bytes in its end.

z	$\log_2(\mathcal{C})$
147 b	127.99
500 b	126.23
1 kb	125.23
1 Mb	115.27
1 Gb	105.30
10^9 Gb	75.19

5 Other Properties of CPS

The CPS of a function is preserved by some transformation as shown in Lemma 1. The collision average of $g_1 \circ g_2$ has a simple expression given in Lemma 2.

Lemma 1. *Let g be a function with* CPS $\mathfrak{C}$*, $P : \mathcal{S} \to \mathcal{S}$ be a permutation and $J : \mathcal{S} \to \mathcal{S}$ be injective over $g(\mathcal{S})$. Then $g' = J \circ g \circ P$ has* CPS $\mathfrak{C}$ *as well.*

Proof. Since J is injective over $g(\mathcal{S})$, we have $g'(y) = g'(a)$ if and only if $g(P(y)) = g(P(a))$. Since the events "$g'(y) = g'(a)$ has k solutions" and "$g(x) = g(P(a))$ has k solutions" have the same probability, namely $\mathfrak{c}_k$, we see that g and g' have the same CPS. □

Lemma 2. *Let g_1 have collision average κ_1 and g_2 have collision average κ_2. Then $g_1 \circ g_2$ has collision average $\kappa_1 + \kappa_2$.*

Proof. Suppose that $(g_2 \circ g_1)(x) = (g_2 \circ g_1)(y)$ with $x \neq y$. There are two distinct ways this could happen:

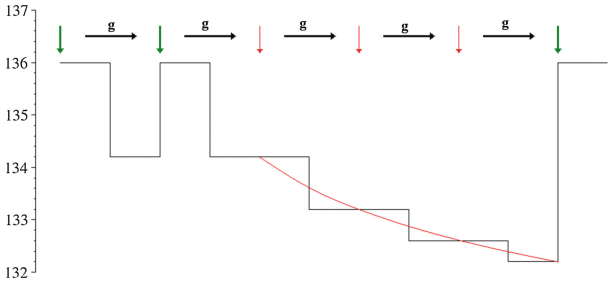


Fig. 7. Evolution of the number of possible values for the internal state of GLUON-64 when a message block is absorbed (thick vertical arrow) or when it absorbs a constant several times (thin vertical arrow) $z = 3$ times.

- if $g_1(x) = g_1(y)$, which happens in κ_1 cases on average,
- or if $g_1(x) \neq g_1(y)$ but $g_2\big(g_1(x)\big) = g_2\big(g_1(y)\big)$. There are on average κ_2/ℓ_1 solutions for $g_2(X) = g_2(Y)$ in $g_1(\mathcal{S})$ where ℓ_1 is the shrinking ratio of g_1. However, each of these solutions is the image of ℓ_1 elements of $\mathcal{S}$ by g_1.

Overall, the equation has $\kappa_1 + \ell_1 \cdot \kappa_2/\ell_1 = \kappa_1 + \kappa_2$ solutions, which proves the lemma. □

Note that Lemma 2 had to hold at least for $g_1 = g_2$ because otherwise we would have had a contradiction with the asymptotic behaviour of ℓ_i described in Theorem 2.

6 Conclusion

We introduced the notion of CPS and of the collision average κ, which is computed from the CPS. The collision average value determines the shrinking ratio and the collision tree size of an iterative construction, and hence directly affects their security, in particular preimage and collision resistance.

We have showed that the T-sponge is a fragile object when the rate parameter is small. For instance, preimages to long messages of specific structure become much easier to find. We gave specific recommendations for the designers of such constructions. Hopefully, our framework might become a useful tool in the design.

Finally, we demonstrated a practical application of our methodology. Aided with a SAT-solver, we found collisions for the GLUON-64 update function and then approximated its CPS and the collision average κ. We showed that for not so long messages of 1 Gigabyte a preimage can be found with complexity 2^{105} compared to the security claim of 2^{128}, and shortcut attacks are possible for messages of only a kilobyte long.

Acknowledgement. The authors thank the designers of the GLUON family of hash functions for providing a reference implementation, Alex Biryukov for very helpful discussions and the anonymous reviewers for their comments.

References

1. Babbage, S., Dodd, M.: The MICKEY stream ciphers. In: Robshaw, M., Billet, O. (eds.) New Stream Cipher Designs. LNCS, vol. 4986, pp. 191–209. Springer, Heidelberg (2008)
2. Golić, J.D.: Cryptanalysis of alleged A5 stream cipher. In: Fumy, W. (ed.) EUROCRYPT 1997. LNCS, vol. 1233, pp. 239–255. Springer, Heidelberg (1997)
3. Bellare, M., Kohno, T.: Hash function balance and its impact on birthday attacks. In: Cachin, C., Camenisch, J.L. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 401–418. Springer, Heidelberg (2004)
4. Flajolet, P., Odlyzko, A.M.: Random mapping statistics. In: Quisquater, J.-J., Vandewalle, J. (eds.) EUROCRYPT 1989. LNCS, vol. 434, pp. 329–354. Springer, Heidelberg (1990)
5. Hong, J., Kim, W.-H.: TMD-tradeoff and state entropy loss considerations of streamcipher MICKEY. In: Maitra, S., Veni Madhavan, C.E., Venkatesan, R. (eds.) INDOCRYPT 2005. LNCS, vol. 3797, pp. 169–182. Springer, Heidelberg (2005)
6. Röck, A.: Stream ciphers using a random update function: study of the entropy of the inner state. In: Vaudenay, S. (ed.) AFRICACRYPT 2008. LNCS, vol. 5023, pp. 258–275. Springer, Heidelberg (2008)
7. Bertoni, G., Daemen, J., Peeters, M., Van Assche, G.: Sponge functions. In: ECRYPT hash Workshop, vol. 2007, Citeseer (2007)
8. Teo, S.G., Bartlett, H., Alhamdan, A., Simpson, L., Wong, K.K.H., Dawson, E.: State convergence in bit-based stream ciphers (2013)
9. Biryukov, A., Shamir, A., Wagner, D.: real time cryptanalysis of A5/1 on a PC. In: Schneier, B. (ed.) FSE 2000. LNCS, vol. 1978, pp. 1–18. Springer, Heidelberg (2001)
10. Bertoni, G., Daemen, J., Peeters, M., Van Assche, G.: On the indifferentiability of the sponge construction. In: Smart, N.P. (ed.) EUROCRYPT 2008. LNCS, vol. 4965, pp. 181–197. Springer, Heidelberg (2008)
11. Guo, J., Peyrin, T., Poschmann, A.: The PHOTON family of lightweight hash functions. In: Rogaway, P. (ed.) CRYPTO 2011. LNCS, vol. 6841, pp. 222–239. Springer, Heidelberg (2011)
12. Bertoni, G., Daemen, J., Peeters, M., Van Assche, G.: Keccak and the sha-3 standardization. Slides of a NIST presentation (2013). http://csrc.nist.gov/groups/ST/hash/sha-3/documents/Keccak-slides-at-NIST.pdf
13. Kelsey, J., Kohno, T.: Herding hash functions and the nostradamus attack. In: Vaudenay, S. (ed.) EUROCRYPT 2006. LNCS, vol. 4004, pp. 183–200. Springer, Heidelberg (2006)
14. Kelsey, J., Schneier, B.: Second preimages on n-bit hash functions for much less than 2^n work. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 474–490. Springer, Heidelberg (2005)
15. Berger, T.P., D'Hayer, J., Marquet, K., Minier, M., Thomas, G.: The GLUON family: a lightweight hash function family based on FCSRs. In: Mitrokotsa, A., Vaudenay, S. (eds.) AFRICACRYPT 2012. LNCS, vol. 7374, pp. 306–323. Springer, Heidelberg (2012)
16. Arnault, F., Berger, T.P.: F-FCSR: design of a new class of stream ciphers. In: Gilbert, H., Handschuh, H. (eds.) FSE 2005. LNCS, vol. 3557, pp. 83–97. Springer, Heidelberg (2005)
17. Hell, M., Johansson, T.: Breaking the stream ciphers F-FCSR-H and F-FCSR-16 in real time. J. Cryptology **24**(3), 427–445 (2011)

18. Arnault, F., Berger, T., Lauradoux, C., Minier, M., Pousse, B.: A new approach for FCSRs. In: Jacobson Jr., M.J., Rijmen, V., Safavi-Naini, R. (eds.) SAC 2009. LNCS, vol. 5867, pp. 433–448. Springer, Heidelberg (2009)

A Proofs and Experiments

A.1 Proof of the Iterated Values of ℓ_i

Let us prove Theorem 1.

Proof. We shall look at the effect multiple iterations of g have over sets $\{x_0, ..., x_k\}$ where $g(x_j) = g(x_{j'})$ for all j, j'.

Let x_0 be in $g'(\mathcal{S})$ and such that there are k other elements $\{x_1, ..., x_k\}$ such that $g(x_0) = g(x_j)$, i.e. $x \in V_{k+1}$.

As every element in $\mathcal{S}$ is in $g'(\mathcal{S})$ with probability only $1/\ell'_1$, the number of elements colliding with x in $g'(\mathcal{S})$ follows a binomial distribution with parameters $(m, k, 1/\ell'_1)$ (we consider that the output of g' are uniformly distributed over $\mathcal{S}$ and that they are independent from one another). Thus, there are m elements colliding with $x \in g'(\mathcal{S})$ with probability $\binom{k}{m}(1/\ell'_1)^m(1-1/\ell'_1)^{k-m}$. Let C_{m+1} be the probability that $g(x_0 + x) = g(x_0)$ has m non-zero solutions in $g'(\mathcal{S})$ knowing that $x_0 \in g'(\mathcal{S})$:

$$C_{m+1} = \sum_{k=m}^{\infty} \mathfrak{c}_{k+1} \binom{k}{m} \Big(\frac{1}{\ell'_1}\Big)^m \Big(1 - \frac{1}{\ell'_1}\Big)^{k-m}. \tag{A.1}$$

Furthermore, we have:

$$\frac{\ell'_1}{\ell_1(g \circ g')} = \frac{|(g \circ g')(\mathcal{S})|}{|g'(\mathcal{S})|} = \sum_{m=1}^{\infty} \frac{C_m}{m},$$

and so:

$$\begin{aligned}\frac{\ell'_1}{\ell_1(g \circ g')} &= \sum_{m=1}^{\infty} \frac{1}{m} \sum_{k=m-1}^{\infty} \mathfrak{c}_{k+1} \binom{k}{m-1} \Big(\frac{1}{\ell'_1}\Big)^{m-1} \Big(1 - \frac{1}{\ell'_1}\Big)^{k-m+1} \\ &= \sum_{k=0}^{\infty} \sum_{m=0}^{k} \frac{\mathfrak{c}_{k+1}}{m+1} \binom{k}{m} \Big(\frac{1}{\ell'_1}\Big)^m \Big(1 - \frac{1}{\ell'_1}\Big)^{k-m}.\end{aligned}$$

This expression can be simplified because $\binom{k}{m}/(m+1) = \binom{k+1}{m+1}/(k+1)$, so that:

$$\begin{aligned}\frac{\ell'_1}{\ell_1(g \circ g')} &= \sum_{k=0}^{\infty} \frac{\mathfrak{c}_{k+1}}{k+1} \sum_{m=0}^{k} \binom{k+1}{m+1} \Big(\frac{1}{\ell'_1}\Big)^m \Big(1 - \frac{1}{\ell'_1}\Big)^{k-m} \\ &= \sum_{k=0}^{\infty} \frac{\mathfrak{c}_{k+1}}{k+1} \sum_{m=1}^{k+1} \binom{k+1}{m} \Big(\frac{1}{\ell'_1}\Big)^{m-1} \Big(1 - \frac{1}{\ell'_1}\Big)^{k-(m-1)} \\ &= \sum_{k=0}^{\infty} \frac{\mathfrak{c}_{k+1} \cdot \ell'_1}{k+1} \Big(\sum_{m=0}^{k+1} \binom{k+1}{m} \Big(\frac{1}{\ell'_1}\Big)^m \Big(1 - \frac{1}{\ell'_1}\Big)^{k+1-m} - \big(1 - \frac{1}{\ell'_1}\big)^{k+1} \Big).\end{aligned}$$

Note that $\sum_{m=0}^{k+1} \binom{k+1}{m} \left(\frac{1}{\ell_1'}\right)^m \left(1-\frac{1}{\ell_1'}\right)^{k+1-m} = 1$ (binomial theorem), so in the end we obtain:

$$\begin{aligned}\frac{1}{\ell_1(g\circ g')} &= \sum_{k=0}^{\infty} \frac{\mathfrak{c}_{k+1}}{k+1}\Big(1-\big(1-\frac{1}{\ell_1'}\big)^{k+1}\Big) \\ &= \sum_{k=1}^{\infty} \frac{\mathfrak{c}_k}{k}\Big(1-\big(1-\frac{1}{\ell_1'}\big)^{k}\Big) = \frac{1}{\ell_1} - \sum_{k=1}^{\infty} \frac{\mathfrak{c}_k}{k}\big(1-\frac{1}{\ell_1'}\big)^k\end{aligned}$$

which proves the Theorem. □

Note that this result is coherent with the one found by [4] in the case of random functions, i.e. when $\{\mathfrak{c}_k\}_{k\geq 1} = \{e^{-1}/\big((k-1)!\big)\}_{k\geq 1}$. Indeed, they prove that

$$\frac{1}{\ell_{i+1}} = 1 - \exp\big(\frac{-1}{\ell_i}\big),$$

which is the same as the one we found:

$$\begin{aligned}\frac{1}{\ell_{i+1}} &= \sum_{k=1}^{\infty} \frac{\mathfrak{c}_k}{k}\Big(1-\big(1-\frac{1}{\ell_i}\big)^{k}\Big) \\ &= e^{-1}\sum_{k=1}^{\infty} \frac{1}{k!}\Big(1-\big(1-\frac{1}{\ell_i}\big)^{k}\Big) \;=\; 1-\exp\big(\frac{-1}{\ell_i}\big).\end{aligned}$$

A.2 Proof of the Asymptotic Behaviors

Theorem 1 gives the recurrence relation ℓ_i satisfies so we can prove its asymptotic behavior.

Proof. Since ℓ_i is obviously increasing (the output space keeps shrinking and we keep $i < 2^{n/2}$ to remain away from the main cycle) we have, for large enough i:

$$\begin{aligned}\frac{1}{\ell_{i+1}} &= \sum_{k=1}^{\infty} \frac{\mathfrak{c}_k}{k}\Big(1-\big(1-\frac{k}{\ell_i}+\frac{k(k-1)}{2\cdot\ell_i^2}+\mathrm{o}(\ell_i^{-2})\big)\Big) \\ &= \frac{1}{\ell_i}\sum_{k=1}^{\infty} \mathfrak{c}_k\,\big(1-\frac{k-1}{2\cdot\ell_i}+\mathrm{o}(\ell_i^{-1})\big),\end{aligned}$$

so that we have:

$$\begin{aligned}\frac{\ell_i}{\ell_{i+1}} &= \sum_{k=1}^{\infty}\mathfrak{c}_k - \sum_{k=1}^{\infty}\frac{\mathfrak{c}_k\cdot(k-1)}{2\cdot\ell_i} + \mathrm{o}(\ell_i^{-1}) \\ &= 1-\frac{\kappa/2}{\ell_i}+\mathrm{o}(\ell_i^{-1})\end{aligned}$$

which in turns implies

$$\begin{aligned}\ell_{i+1} &= \frac{\ell_i}{1 - \frac{\kappa/2}{\ell_i} + \mathrm{o}(\ell_i^{-1})} \\ &= \ell_i + \frac{\kappa}{2} + \mathrm{o}(1),\end{aligned}$$

so that $\ell_i = \frac{\kappa}{2} \cdot i + \mathrm{o}(i)$. This observation concludes the proof of the behavior of ℓ_i.

Let us now look at ν_i. There are on average ℓ_w elements reaching $y \in g^w(\mathcal{S})$ in exactly w iterations. Since $g^i(\mathcal{S}) \subseteq g^w(\mathcal{S})$ for all $w \leq i$, we have that $y \in g^i(\mathcal{S})$ is reached, on average, by:

- ℓ_1 elements in exactly 1 iteration,
- ℓ_2 elements in exactly 2 iterations,

  ...
- ℓ_i elements in exactly i iterations.

Overall, there are on average $\sum_{w=1}^{i} \ell_w \approx \sum_{w=1}^{i} (\kappa/2)w \approx (\kappa/4)i^2$ elements reaching $y \in g^i(\mathcal{S})$ after at most i iterations of g. □

A.3 Experimental Justification of Conjecture 1

For every N between 12 and 17 included, we generated 100 functions with a given CPS and, for each of them, we picked 40 elements at random in $\mathbb{F}_2^N$ and computed $\lambda/2^{N/2}$ and $\mu/2^{N/2}$ for each of them (24,000 data points for each CPS). The average of these values for CPS's corresponding to different values of κ are given in Fig. 8. As we can see, both $\lambda/2^{N/2}$ and $\mu/2^{N/2}$ are almost equal to $\sqrt{\pi/(8\kappa)}$, which is equivalent to Conjecture 1 being correct.

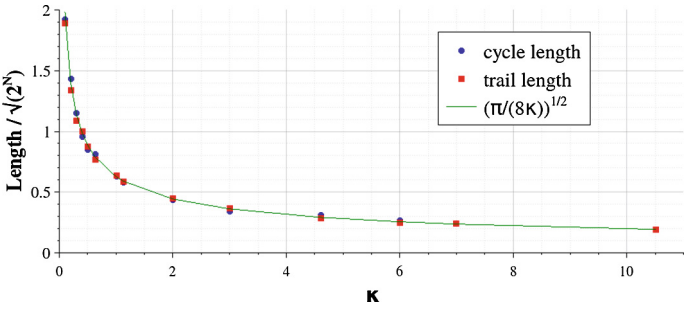


Fig. 8. Average value of $\lambda/2^{N/2}$ and $\mu/2^{N/2}$ for different κ.

A.4 Proof of the Effect of the CPS on a T-sponge

Proof. Our proof is a modified version of the one used by Bertoni et al. in the paper where they introduced the sponge construction [7]. In particular, we use the same terminology: we call the elements of $\mathbb{F}_2^{c+r}$ "nodes" and we partition the space according to the value of the bits in the capacity to obtain 2^c "super-nodes", each containing 2^r nodes. There is an oriented edge from node x to node y if and only if $y = g(x)$. Finding a collision in H boils down to finding two different paths in this graph starting from points in the super-node with capacity zero to an identical super-node.

We shall study the fan-in and the fan-out of these super-nodes, the fan-in of a super-node being the number of edges going to it and the fan-out the number of edges going out of it. In this framework, the fan-out of each super-node is 2^r. However, the number of edges going in each super-node is not constant. Consider some super-node Y made of nodes $y_1, ..., y_{2^r}$. Each y_i has a fan-in $F(y_i)$ so that the $F(y_i)$'s are independent and identically distributed random variables with the distribution

$$\mathbb{P}[F(y_i) = k] = \frac{\mathfrak{c}_k}{k} \text{ if } k \geq 1 \ , \ \mathbb{P}[F(y_i) = 0] = 1 - \frac{1}{\ell_1}$$

which has a mean equal to 1 and a variance equal to κ.

The value of the fan-in of the super-node Y is the sum of the fan-in's of its nodes:

$$F(Y) = \sum_{i=1}^{2^r} F(y_i).$$

We consider that 2^r is large enough to apply the central limit theorem so that $F(Y)$ is normally distributed with mean equal to 2^r and variance equal to $\kappa \cdot 2^r$.

Consider now the set $\mathcal{N}_k$ of all the super-nodes with fan-in equal to k; in other words the set of the super-nodes with exactly k preimages. It has a size equal to $|\mathcal{N}_k| = 2^c G(k)$ where

$$G(k) = \frac{1}{\sqrt{2\pi \cdot \kappa \cdot 2^r}} \cdot \exp\big(-\frac{1}{2} \cdot \frac{(k-2^r)^2}{\kappa \cdot 2^r}\big)$$

and the $\mathcal{N}_k$'s form a partition of the space of the super-nodes. Consider some node x_1: the probability that its image by g is in a super-node of $\mathcal{N}_k$ is

$$\mathbb{P}[g(x_1) \in \mathcal{N}_k] \ = \ \frac{k}{2^{c+r}} \cdot |\mathcal{N}_k| \ = \ \frac{k}{2^r} \cdot G(k)$$

Let V be the super-node $g(x_1)$ is in. The probability that a second element $x_2 \neq x_1$ is such that $g(x_2)$ is in the same super-node as $g(x_1)$ is the probability that x_2 is at the beginning of one of the $k-1$ edges going to V which are not the one starting at x_1. Therefore, the probability that $g(x_1)$ and $g(x_2)$ are in the same super-node V knowing that $V \in \mathcal{N}_k$ is

$$\mathbb{P}[g(x_1), g(x_2) \in V \mid V \in \mathcal{N}_k] = \frac{k-1}{2^{c+r}} \cdot \frac{k}{2^r} \cdot G(k)$$

so that the probability that $g(x_1)$ and $g(x_2)$ have the same capacity bits for x_1 and x_2 chosen uniformly at random is

$$\mathbb{P}[g(x_1), g(x_2) \in V] = \sum_{k=0}^{\infty} \frac{k(k-1)}{2^{c+2r}} \cdot G(k) \approx \frac{(2^r)^2 + \kappa \cdot 2^r - 2^r}{2^{c+2r}}.$$

Therefore, the probability of success of a collision search performed by absorbing Q messages at random until two internal states with the same capacity bits are observed is

$$\mathbb{P}[\text{success of collision search}] \approx \binom{Q}{2} \frac{2^{2r} + 2^r(\kappa - 1)}{2^{c+2r}} \approx \frac{Q^2}{2^{c+1}} \cdot \left(1 + \frac{\kappa - 1}{2^r}\right).$$

□

B Algorithms

Algorithm 1. Enumerating all the solutions of $g(a + \delta) = g(a)$.

$D =$ empty list
$b = 0$
while $b < rw - 1$ **do**
  $F = \mathrm{CNF}(\rho_k^1) + \mathrm{CNF}(\rho_k^2) + \mathrm{CNF}(\rho_k^1(x) = \rho_k^2(y))$
  $F = F + \mathrm{CNF}(x = a) + \mathrm{CNF}(x_b + y_b = 1)$
  for δ in D **do**
    $F = F + \mathrm{CNF}(x + y \neq \delta)$
  end for
  if SAT-solver concludes that F is satisfiable **then**
    Retrieve y from the assignment and append $x + y$ to D
  else
    $b = b + 1$ ▷ We move on only when this bit is exhaustively used
  end if
end while
Return D

Improved All-Subkeys Recovery Attacks on FOX, KATAN and SHACAL-2 Block Ciphers

Takanori Isobe(✉) and Kyoji Shibutani

Sony Corporation, 1-7-1 Konan, Minato-ku, Tokyo 108-0075, Japan
{Takanori.Isobe,Kyoji.Shibutani}@jp.sony.com

Abstract. The all-subkeys recovery (ASR) attack is an extension of the meet-in-the-middle attack, which allows evaluating the security of a block cipher without analyzing its key scheduling function. Combining the ASR attack with some advanced techniques such as the function reduction and the repetitive ASR attack, we show the improved ASR attacks on the 7-round reduced FOX64 and FOX128. Moreover, the improved ASR attacks on the 119-, 105- and 99-round reduced KATAN32, KATAN48 and KATAN64, and the 42-round reduced SHACAL-2 are also presented, respectively. As far as we know, all of those attacks are the best single-key attacks with respect to the number of attacked rounds in literature.

Keywords: Block cipher · Meet-in-the-middle attack · All-subkeys recovery attack

1 Introduction

Since the meet-in-the-middle (MITM) attack was applied to KTANTAN [7], a lot of its improvements have been introduced such as the splice-and-cut technique [4], the initial structure [24], the biclique cryptanalysis [6,19], the internal state guess [10,14], the sieve-in-the-middle technique [9] and the parallel-cut technique [23]. Since the MITM attack basically exploits the weakness in the key scheduling function, it was believed that a block cipher having a strong key scheduling function has enough immunity against the MITM attack.

Isobe and Shibutani proposed the all-subkeys recovery (ASR) approach at SAC 2012 as an extension of the MITM attack [16], and showed several best attacks on block ciphers having relatively complex key scheduling function including CAST-128 [1], SHACAL-2 [13], FOX [18] and KATAN [8]. One of the advantages of the ASR attack compared to the basic MITM attack is that it does not need to take the key scheduling function into account, since it recovers all subkeys instead of the master key. Thus, it has been shown that the MITM attack may be more applicable to block ciphers. Moreover, the ASR approach enables us to evaluate the lower bounds on the security against key recovery attack for a block cipher structure, since the ASR attack is applicable independently from the underlying key scheduling function. For Feistel schemes, such lower bounds were shown by

C. Cid and C. Rechberger (Eds.): FSE 2014, LNCS 8540, pp. 104–126, 2015.
DOI: 10.1007/978-3-662-46706-0_6

Table 1. Summary of attacks on FOX64/128, KATAN32/48/64 and SHACAL-2 (single-key setting)

Algorithm	Attack type	# attacked rounds	Time	Memory	Data	Reference
FOX64	Integral	5	$2^{109.4}$	Not given	2^9 CP	[26][a]
	Impossible Diff.	5	2^{80}	Not given	2^{40} CP	[27][a]
	ASR	**6**	2^{124}	2^{124}	15 CP	This paper
	ASR	**7**	2^{124}	2^{123}	$2^{30.9}$ CP	This paper
FOX128	Integral	5	$2^{205.6}$	Not given	2^9 CP	[26]
	Impossible Diff.	5	2^{144}	Not given	2^{72} CP	[27]
	ASR	5	2^{228}	2^{228}	14 KP	[16]
	ASR	**6**	2^{221}	2^{221}	26 CP	This paper
	ASR	**7**	2^{242}	2^{242}	2^{63} CP	This paper
KATAN32	ASR	110	2^{77}	$2^{75.1}$	138 KP	[16]
	Differential	114	2^{77}	Not given	$2^{31.9}$ CP	[2]
	ASR	**119**	$2^{79.1}$	$2^{79.1}$	144 CP	This paper
KATAN48	ASR	100	2^{78}	2^{78}	128 KP	[16]
	ASR	**105**	$2^{79.1}$	$2^{79.1}$	144 CP	This paper
KATAN64	ASR	94	$2^{77.1}$	$2^{79.1}$	116 KP	[16]
	ASR	**99**	$2^{79.1}$	$2^{79.1}$	142 CP	This paper
SHACAL-2	ASR	41	2^{500}	2^{492}	244 KP	[16]
	ASR	**42**	2^{508}	2^{508}	2^{25} CP	This paper

[a]While the 6- and 7-round attacks on FOX64 were presented in [26,27], the time complexities of both attacks exceed 2^{128}. Similarly, the optimized exhaustive key-searches on the KATAN family were presented in [28].

using the ASR attack with a couple of its improvements such as the function reduction in [17]. For instance, the function reduction reduces the number of subkeys required to compute the matching state by exploiting degrees of freedom of plaintext/ciphertext pairs. Then, the number of attacked rounds can be increased by the ASR attack. Therefore, in order to more precisely evaluate the security of a block cipher against the ASR attack, the following natural question arises: Are those advanced techniques applicable to other structures such as Lai-Massey and LFSR-type schemes?

In this paper, we first apply the function reduction technique to Lai-Massey, LFSR-type and source-heavy generalized Feistel schemes to extend the ASR attacks on those structures. Then, we further improve the attacks on those structures by exploiting structure dependent properties and optimizing data complexity in the function reduction. For instance, the ASR attack with the function reduction on FOX can be improved by using the keyless one-round relation in Lai-Massey scheme. Moreover, combined with the repetitive ASR approach, which optimizes the data complexity when using the function reduction, the attack on FOX can be further improved. Those results are summarized in Table 1. As far as we know, all of the results given by this paper are the best single-key attacks with

respect to the number of attacked rounds in literature[1] We emphasize that our improvements keep the basic concept of the ASR attack, which enables us to evaluate the security of a block cipher without analyzing its key scheduling function. Therefore, our results are considered as not only the best single-key attacks on the specific block ciphers but also the lower bounds on the security of the target block cipher structures independently from key scheduling functions.

The rest of this paper is organized as follows. Section 2 briefly reviews the previously shown techniques including the all-subkeys recovery approach, the function reduction and the repetitive all-subkeys recovery approach. The improved all-subkeys recovery attacks on `FOX64/128`, `KATAN32/48/64` and `SHACAL-2` are presented in Sects. 3, 4 and 5, respectively. Finally, we conclude in Sect. 6.

2 Preliminary

2.1 All-Subkeys Recovery Approach [16]

The all-subkeys recovery (ASR) attack was proposed in [16] as an extension of the meet-in-the-middle (MITM) attack. Unlike the basic MITM attack, the ASR attack is guessing all-subkeys instead of the master key so that the attack can be constructed independently from the underlying key scheduling function.

Let us briefly review the procedure of the ASR attack. First, an attacker determines an s-bit matching state S in a target n-bit block cipher consisting of R rounds. The state S can be computed from a plaintext P and a set of subkey bits $\mathcal{K}_{(1)}$ by a function $\mathcal{F}_{(1)}$ as $S = \mathcal{F}_{(1)}(P, \mathcal{K}_{(1)})$. Similarly, S can be computed from the corresponding ciphertext C and another set of subkey bits $\mathcal{K}_{(2)}$ by a function $\mathcal{F}_{(2)}$ as $S = \mathcal{F}^{-1}_{(2)}(C, \mathcal{K}_{(2)})$. Let $\mathcal{K}_{(3)}$ be a set of the remaining subkey bits, i.e., $|\mathcal{K}_{(1)}|+|\mathcal{K}_{(2)}|+|\mathcal{K}_{(3)}| = R{\cdot}\ell$, where ℓ denotes the size of each subkey. For a plaintext P and the corresponding ciphertext C, the equation $\mathcal{F}_{(1)}(P, \mathcal{K}_{(1)}) = \mathcal{F}^{-1}_{(2)}(C, \mathcal{K}_{(2)})$ holds when the guessed subkey bits $\mathcal{K}_{(1)}$ and $\mathcal{K}_{(2)}$ are correct. Since $\mathcal{K}_{(1)}$ and $\mathcal{K}_{(2)}$ can be guessed independently, we can efficiently filter out the incorrect subkeys from the key candidates. After this process, it is expected that there will be $2^{R\cdot\ell-s}$ key candidates. Note that the number of key candidates can be reduced by parallel performing the matching with additional plaintext/ciphertext pairs. In fact, using N plaintext/ciphertext pairs, the number of key candidates is reduced to $2^{R\cdot\ell-N\cdot s}$, as long as $N \leq (|\mathcal{K}_{(1)}| + |\mathcal{K}_{(2)}|)/s$. Finally, the attacker exhaustively searches the correct key from the remaining key candidates. The required computations (i.e. the number of encryption function calls) of the attack in total C_{comp} is estimated as

$$C_{comp} = \max(2^{|\mathcal{K}_{(1)}|}, 2^{|\mathcal{K}_{(2)}|}) \times N + 2^{R\cdot\ell-N\cdot s}. \quad (1)$$

The number of required plaintext/ciphertext pairs is $\max(N, \lceil (R{\cdot}\ell - N{\cdot}s)/n \rceil)$, where n is the block size of the target cipher. The required memory is about $\min(2^{|\mathcal{K}_{(1)}|}, 2^{|\mathcal{K}_{(2)}|}) \times N$ blocks, which is the cost of the table used for the matching.

[1] In the related-key setting, the attacks on the 174-, 145-, 130- and 44-round reduced `KATAN32`, `KATAN48`, `KATAN64` and `SHACAL-2` were presented, respectively [15,20].

2.2 Improvements on All-Subkeys Recovery Approach

In the ASR attack, the number of the subkeys required to compute the state S from P or C, i.e., $\mathcal{K}_{(1)}$ or $\mathcal{K}_{(2)}$, is usually dominant parameter in the required complexities. Thus, in general, reducing those subkeys $\mathcal{K}_1$ and $\mathcal{K}_2$ will make the ASR attack applicable to more rounds. In the followings, we briefly review and introduce a couple of techniques to reduce such subkeys required to compute the matching state.

Function Reduction Technique. For Feistel ciphers, the *function reduction* technique that directly reduces the number of involved subkeys was introduced in [17]. The basic concept of the function reduction is that fixing some plaintext bits, ciphertext bits or both by exploiting degrees of freedom of a plaintext/ciphertext pair allows an attacker to regard a key dependent variable as a new subkey. As a result, substantial subkeys required to compute the matching state are reduced. By using the function reduction, the lower bounds on the security of several Feistel ciphers against generic key recovery attacks were given in [17]. Note that a similar approach was presented in [11] for directly guessing intermediate state values, while in the function reduction, equivalently transformed key values are guessed.

Suppose that the i-th round state S_i is computed from the $(i-1)$-th round state S_{i-1} XORed with the i-th round subkey K_i by the i-th round function G_i, i.e., $S_i = G_i(K_i \oplus S_{i-1})$. For clear understanding, we divide the function reduction into two parts: a key linearization and an equivalent transform as follows.

- **Key Linearization.** Since the i-th round function G_i is a non-linear function, the i-th round subkey K_i cannot pass through G_i by an equivalent transform. The key linearization technique, which is a part of the function reduction, exploits the degree of freedom of plaintexts/ciphertexts to express S_i as a linear relation of S_{i-1} and K_i, i.e., $S_i = L_i(S_{i-1}, K_i)$, where L_i is a linear function. Once S_i is represented by a linear relation of S_{i-1} and K_i, K_i can be forwardly moved to a next non-linear function by an equivalent transform. Note that, if the splice-and-cut technique [4] is used with the key linearization, K_i can be divided into both forward and backward directions.
- **Equivalent Transform.** After the key linearization, the i-th round subkey K_i is replaced with a new subkey K_i' to pass through a non-linear function. However, in order to reduce the involved subkey bits on the trails to the matching state, all-subkeys on the trails affected by K_i' are also replaced with new variables by an equivalent transform. Consequently, the number of subkeys required to compute the matching state can be reduced. For the Feistel ciphers, it is easily done by replacing all-subkeys in the even numbered rounds K_j with $K_j'(= K_1' \oplus K_j)$, where j is even.

The splice-and-cut technique [4], which was originally presented in the attack of the two-key triple DES [21], was well used in the recent meet-in-the-middle attacks [3,6,7,19,24]. It regards that the first and last rounds are consecutive by exploiting degree of freedom of plaintext/ciphertexts, and thus any round can be

the start point. In general, the splice-and-cut technique is useful to analyze the specific block cipher that key-dependency varies depending on the chunk separation. However, in the ASR approach, the splice-and-cut technique does not work effectively, since the ASR treats all-subkeys as independent variables to evaluate the security independently from the key scheduling function. On the other hand, the function reduction exploits degrees of freedom of plaintexts/ciphertexts to reduce subkey bits required to compute the matching state, and does not use relations among subkeys. Therefore, the function reduction technique is more useful and suitable for the ASR approach than the splice-and-cut technique. However, as mentioned in the description of the key linearization, the combined use of the splice-and-cut and the function reduction in the key linearization is also possible, e.g. the attack on Feistel-1 [17] and the attack on SHACAL-2 in this paper.

Repetitive All-Subkeys Recovery Approach. Since the function reduction exploits the degree of freedom of plaintexts/ciphertexts, it sometimes causes an attack infeasible due to lack of available data. For such cases, we introduce a variant of the all-subkeys recovery approach called *repetitive all-subkeys recovery approach* that repeatedly applies the all-subkeys recovery to detect the correct key. The variant can reduce the required data for each all-subkeys recovery phase, though the total amount of the required data is unchanged. Note that a similar technique, called inner loop technique, was used in [5,23] for reducing the memory requirements. The repetitive all-subkeys recovery approach is described as follows.

1. Mount the ASR attack with N plaintext/ciphertexts, where N is supposed to be less than $(|\mathcal{K}_{(1)}| + |\mathcal{K}_{(2)}|)/s$, then put the remaining key candidates into a table T_1. The number of expected candidates is $|\mathcal{K}_{(1)}| + |\mathcal{K}_{(2)}| - N \cdot s$.
2. Repeatedly mount the ASR attack with *different* N plaintext/ciphertexts. If the remaining candidate match with ones in T_1, such candidates are put into another table T_2. The number of expected candidates is $|\mathcal{K}_{(1)}| + |\mathcal{K}_{(2)}| - 2 \cdot N \cdot s$.
3. Repeat the above processes until the correct key is found, i.e., $M = (|\mathcal{K}_{(1)}| + |\mathcal{K}_{(2)}|)/(N \cdot s)$ times.

When the above procedure is repeated M (≥ 2) times, the computational costs to detect $\mathcal{K}_{(1)}$ and $\mathcal{K}_{(2)}$ are estimated as

$$C_{comp} = (\max(2^{|\mathcal{K}_{(1)}|}, 2^{|\mathcal{K}_{(2)}|}) \times N) \times M + (2^{|\mathcal{K}_{(1)}|+|\mathcal{K}_{(2)}|-N\cdot s}) + \cdots + (2^{|\mathcal{K}_{(1)}|+|\mathcal{K}_{(2)}|-(M-1)\cdot N\cdot s}).$$

While the required data in total is $(|\mathcal{K}_{(1)}| + |\mathcal{K}_{(2)}|)/s$ $(= ((|\mathcal{K}_{(1)}| + |\mathcal{K}_{(2)}|)/(M \cdot s)) \cdot M)$, each ASR approach is done with $N = (|\mathcal{K}_{(1)}| + |\mathcal{K}_{(2)}|)/(M \cdot s)$ data, which is M times less than that required in the basic ASR attack. The required memory is about $\max(2^{|\mathcal{K}_{(1)}|+|\mathcal{K}_{(2)}|-N\cdot s}, \min(2^{|\mathcal{K}_{(1)}|}, 2^{|\mathcal{K}_{(2)}|}) \times N)$ blocks, which is the cost for the table used in the matching. We demonstrate the effectiveness of the proposed variant in the attack on the reduced `FOX` in Sect. 3.

3 Improved All-Subkeys Recovery Attacks on FOX64 and FOX128

In this section, we present the improved ASR attacks using the function reduction and the repetitive ASR approach on the 6- and 7-round reduced FOX64 and FOX128 block ciphers. After short descriptions of FOX64 and FOX128, the function reduction on FOX64 is presented. Then, we show how to construct the attack on the 6-round FOX64, and how to extend it to the 7-round variant by using the repetitive ASR approach. Similarly, the function reduction on FOX128, the attack on the 6-round FOX128, and the attack on the 7-round FOX128 with the repetitive ASR approach are introduced, respectively.

3.1 Descriptions of FOX64 and FOX128

FOX [18], also known as IDEA-NXT, is a family of block ciphers designed by Junod and Vaudenay in 2004. FOX employs a Lai-Massey scheme including two variants referred as FOX64 and FOX128 (see Fig. 1).

FOX64 is a 64-bit block cipher consisting of a 16-round Lai-Massey scheme with a 128-bit key. The i-th round 64-bit input state is denoted as two 32-bit words ($L_{i-1} \,||\, R_{i-1}$). The i-th round function updates the input state using the 64-bit i-th round key K_i as follows:

$$(L_i||R_i) = (\texttt{or}(L_{i-1} \oplus \texttt{f32}(L_{i-1} \oplus R_{i-1}, K_i))||R_{i-1} \oplus \texttt{f32}(L_{i-1} \oplus R_{i-1}, K_i)),$$

where $\texttt{or}(x_0||x_1) = (x_1||(x_0 \oplus x_1))$ for 16-bit x_0, x_1. f32 outputs a 32-bit data from a 32-bit input X and two 32-bit subkeys LK_i and RK_i as ($\texttt{sigma4}(\texttt{mu4}(\texttt{sigma4}(X \oplus LK_i)) \oplus RK_i) \oplus LK_i$), where sigma4 denotes the S-box layer consisting of four 8-bit S-boxes and mu4 denotes the 4×4 MDS matrix. Two 32-bit subkeys LK_i and RK_i are derived from K_i as $K_i = (LK_i||RK_i)$.

FOX128 is a 128-bit block cipher consisting of a 16-round modified Lai-Massey scheme with a 256-bit key. The i-th round 128-bit input state is denoted as four 32-bit words ($LL_{i-1} \,||\, LR_{i-1} \,||\, RL_{i-1} \,||\, RR_{i-1}$). The i-th round function updates the input state using the 128-bit i-th round key K_i as follows:

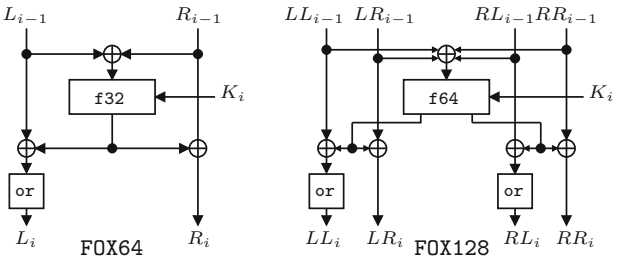


Fig. 1. Round functions of FOX64 and FOX128

$$(LL_i||LR_i) = (\texttt{or}(LL_{i-1} \oplus \phi_L)||LR_{i-1} \oplus \phi_L),$$
$$(RL_i||RR_i) = (\texttt{or}(RL_{i-1} \oplus \phi_R)||RR_{i-1} \oplus \phi_R),$$

where $(\phi_L||\phi_R) = \texttt{f64}((LL_{i-1} \oplus LR_{i-1})||(RL_{i-1} \oplus RR_{i-1}), K_i)$. `f64` outputs a 64-bit data from a 64-bit input X and two 64-bit subkeys LK_i and RK_i as (`sigma8` (`mu8`(`sigma8`$(X \oplus LK_i)) \oplus RK_i) \oplus LK_i$), where `sigma8` denotes the S-box layer consisting of eight 8-bit S-boxes and `mu8` denotes the 8×8 MDS matrix. Two 64-bit subkeys LK_i and RK_i are derived from K_i as $K_i = (LK_i||RK_i)$.

3.2 Function Reduction on FOX64

Key Linearization (Fig. 2). If the value of $L_0 \oplus R_0$ is fixed to a constant CON_1, the input of `f32` is fixed as `f32`(CON_1, K_1). By regarding `f32`(CON_1, K_1) as a 32-bit new key K'_1, K'_1 is XORed to L_0 and R_0. Since `or` is a linear operation, the state after the first round is expressed as $(L_1||R_1) = (\texttt{or}(L_0) \oplus OK'_1)||(R_0 \oplus K'_1)$, where $OK'_1 = \texttt{or}(K'_1)$ (see Fig. 2). This implies that the first round keys linearly affect L_1 and R_1.

Equivalent Transform (Fig. 3). In the second round, OK'_1 and K'_1 are XORed with LK_2 in the first and last operations of `f32` function. Let $LK'_2 = LK_2 \oplus K'_1 \oplus OK'_1$, $K1'' = K1' \oplus LK2$, and $OK1'' = \texttt{or}(OK'1 \oplus LK2)$ be new keys. Then `f32` function contains K'_2 $(= LK'_2||RL_2)$, and K''_1 and $OK1''$ linearly affect outputs of the second round.

In the third round, OK''_1 and K''_1 are also XORed with LK_3 in the first and last operations of `f32` function. Let $LK'_3 = LK_3 \oplus K''_1 \oplus OK''_1$, $K1''' = K1'' \oplus LK2$, and $OK1''' = \texttt{or}(OK''1 \oplus LK2)$ be new keys (see Fig. 3).

Note that the same technique can be applied to the inverse of `FOX64`, because the round function of `FOX64` has the involution property.

3.3 Attack on the 6-Round FOX64

In this attack, we use the following one-round keyless linear relation of the Lai-Massey construction.

$$\texttt{or}^{-1}(L_{i+1}) \oplus R_{i+1} = L_i \oplus R_i.$$

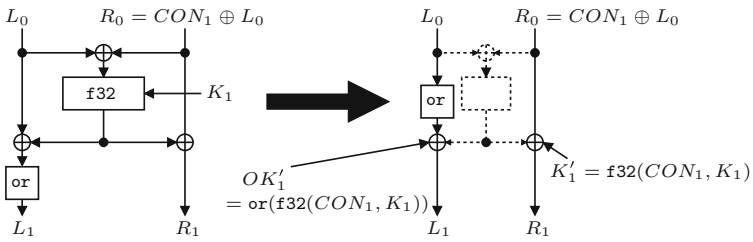


Fig. 2. Key linearization of `FOX64`

From this equation, the 16-bit relation is obtained as follows

$$((L_4^{(1)} \oplus L_4^{(3)})||L_4^{(3)}) \oplus (R_4^{(3)}||R_4^{(1)}) = (L_3^{(3)}||L_3^{(1)}) \oplus (R_3^{(3)}||R_3^{(1)}),$$

where $L_i^{(j)}$ and $R_i^{(j)}$ are the j-th byte of L_i and R_i, respectively, and $L_i^{(3)}$ and $R_i^{(3)}$ are the most significant bytes ,i.e., $L_i = \{L_i^{(3)}||L_i^{(2)}||L_i^{(1)}||L_i^{(0)}\}$ and $R_i = \{R_i^{(3)}||R_i^{(2)}||R_i^{(1)}||R_i^{(0)}\}$.

Forward Computation in $\mathcal{F}_{(1)}$: For given $\{K_2'$, LK_3', $RK_3'^{(3)}$, $RK_3'^{(1)}$, $K_1'''^{(3)}$, $K_1'''^{(1)}, OK_1'''^{(3)}, OK_1'''^{(1)}\}$, $(L_3^{(3)}||L_3^{(1)}) \oplus (R_3^{(3)}||R_3^{(1)})$ is computable. Since $(K_1'''^{(3)}$ $||$ $K_1'''^{(1)})$ and $(OK_1'''^{(3)}||OK_1'''^{(1)})$ linearly affect $(L_3^{(3)}||L_3^{(1)})$ and $(R_3^{(3)}||R_3^{(1)})$, respectively, we can regard $(K_1'''^{(3)}||K_1'''^{(1)}) \oplus (OK_1'''^{(3)}||OK_1'''^{(1)})$ as a new 16-bit key $XORK_1$. Then, $(L_3^{(3)}||L_3^{(1)}) \oplus (R_3^{(3)}||R_3^{(1)})$ is obtained from $112 (= 64 + 32 + 8 + 8)$ bits of the key $\{K_2', LK_3', RK_3'^{(3)}, RK_3'^{(1)}\}$ and linearly-dependent 16-bit key $XORK_1$.

Backward Computation in $\mathcal{F}_{(2)}$: $((L_4^{(1)} \oplus L_4^{(3)})||L_4^{(3)}) \oplus (R_4^{(3)}||R_4^{(1)})$ is obtained from 112 (=64 + 32 + 16) bits of the key $\{K_6, LK_5, RK_5^{(1)}, RK_5^{(3)}\}$. Using the indirect matching technique [3], 8 bits out of 16 bits of $XORK_1$ are moved to the left half of the matching equation. Then, the left and right halves of the equation contains 120 bits of the key, i.e., $|\mathcal{K}_{(1)}| = |\mathcal{K}_{(2)}| = 120$.

Evaluation. When the parameter $N = 15$, the time complexity for finding the involved 240-bit key is estimated as

$$C_{comp} = \max(2^{120}, 2^{120}) \times 15 + 2^{240-15 \cdot 16} = 2^{124}.$$

The required data for the attack is only 15 (=max(15, $\lceil (240 - 15 \cdot 16)/64 \rceil$)) chosen plaintext/ciphertext pairs, and the required memory is estimated as about 2^{124} (=min$(2^{120}, 2^{120}) \times 15$) blocks.

3.4 Attack on the 7-Round FOX64

If the function reduction is applied as well in the backward direction, the 7-round attack is feasible, i.e., the relation of $L_7 \oplus R_7$ is fixed to a constant CON_2. Due to the involution property of the FOX64 round function, $((L_4^{(1)} \oplus L_4^{(3)})$ $||$ $L_4^{(3)})$ $\oplus$ $(R_4^{(3)}||R_4^{(1)})$ is also obtained from $112 (= 64 + 32 + 8 + 8)$ bits of the key and linearly-dependent 16-bit key $XORK_2$. In this attack, we further regard $XORK_1 \oplus XORK_2$ as a 16-bit new key. Then, similar to the attack on the 6-round FOX64, the left and right halves of the equation contain 120 bits of the key, i.e., $|\mathcal{K}_{(1)}| = |\mathcal{K}_{(2)}| = 120$.

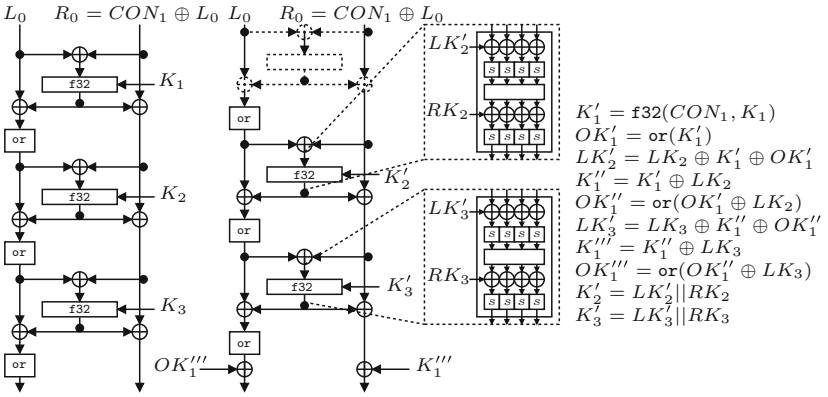


Fig. 3. Function reduction of FOX64

Repetitive ASR Approach. Recall that plaintexts and ciphertexts need to satisfy the 32-bit relations, $L_0 \oplus R_0 = CON_1$ and $L_7 \oplus R_7 = CON_2$. The required data for finding such pairs is equivalently estimated as the game that an attacker finds 32-bit multicollisions by 32-bit-restricted inputs. It has been known that an n-bit t-multicollision is found in $t!^{1/t} \cdot 2^{n \cdot (t-1)/t}$ random data with high probability [25].

In the basic ASR approach, at least $15 (= 240/16)$ multicollisions are necessary to detect the 240-bit involved key. To obtain such pairs with a high probability, it require $2^{32.55} (= 15!^{1/15} \cdot 2^{32 \cdot (14)/15})$ plaintext/ciphertext pairs. However, it is infeasible, since the degree of freedom of plaintexts is only 32 bits.

In order to overcome this problem, we utilize the repetitive all-subkeys recovery approach with $M = 2$ variant. In each all-subkeys recovery phase, the required data is reduced to 8 and 7. Then, such eight 32-bit multicollisions are obtained from $2^{29.9}$ plaintext/ciphertext pairs with a high probability. Thus, we can obtain the required data by exploiting free 32 bits.

Evaluation. The time complexity for finding the involved 240 bits key is estimated as

$$C_{comp} = (\max(2^{120}, 2^{120}) \times 8) \times 2 + (2^{240-8 \cdot 16}) = 2^{124}.$$

The remaining $208 (= 448 - 240)$ bits are obtained by recursively applying all-subkeys recovery attacks. The time complexity for this phase is roughly estimated as $2^{106 (= 208/2+2)}$ using 4 $(= \lceil 208/64 \rceil)$ plaintext/ciphertext pairs.

The required data is $2^{30.9} (= 2^{29.9} \times 2)$ plaintext/ciphertext pairs, and the required memory is about 2^{123} $(= \max(2^{240-128}, \min(2^{120}, 2^{120}) \times 8))$ blocks.

3.5 Function Reduction on FOX128

Key Linearization (Fig. 4). If two 16-bit relations of $LL_0 \oplus LR_0$ and $RL_0 \oplus RR_0$ are fixed to CON_1 and CON_2, respectively, the input of f64 is fixed as f64(CON_1

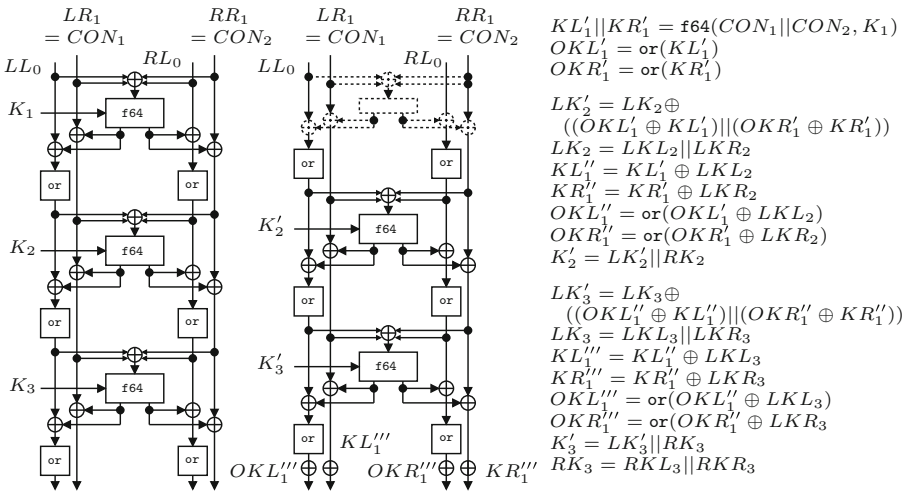


Fig. 4. Function reduction of FOX128

$||\ CON_2, K_1)$. By regarding $\texttt{f64}(CON_1||CON_2, K_1)$ as a 64-bit new key $K'_1 = KL'_1||KR'_1$, KL'_1 and KR'_1 are XORed to $\{LR_0$ and $LR_0\}$ and $\{RR_0$ and $RR_0\}$, respectively. The state after the first round is expressed as follows (see Fig. 4).

$$(LL_1||LR_1||RL_1||RR_1) = (\texttt{or}(LL_0) \oplus OKL'_1)||(LR_0 \oplus KL'_1)||(\texttt{or}(RL_0) \oplus OKR'_1)||(RR_0 \oplus KR'_1),$$

where $OKL'_1 = \texttt{or}(KL'_1)$ and $OKR'_1 = \texttt{or}(KR'_1)$. This implies that the first round keys linearly affect LL_1, LR_1, RL_1 and RR_1.

Equivalent Transform (Fig. 4). The equivalent transform is done similar to FOX64 as shown in Fig. 4.

3.6 Attack on the 6-Round FOX128

We use the following one-round keyless linear relation of the modified Lai-Massey construction,

$$\texttt{or}^{-1}(LL_{i+1}) \oplus LR_{i+1} = LL_i \oplus LR_i.$$

From this equation, the 16-bit relation is obtained as follows:

$$((LL_4^{(1)} \oplus LL_4^{(3)})||LL_4^{(3)}) \oplus (LR_4^{(3)}||LR_4^{(1)}) = (LL_3^{(3)}||LL_3^{(1)}) \oplus (LR_3^{(3)}||LR_3^{(1)}).$$

Forward Computation in $\mathcal{F}_{(1)}$: For given $\{K'_2,\ LK'_3,\ RKL_3'^{(3)},\ RKL_3'^{(1)},\ KL_1'''^{(3)}, KL_1'''^{(1)}, OKL_1'''^{(3)}, OKL_1'''^{(1)}\}$, $(LL_3^{(3)}||LL_3^{(1)}) \oplus (LR_3^{(3)}||LR_3^{(1)})$ is computable. Since $(KL_1'''^{(3)}||KL_1'''^{(1)})$ and $(OKL_1'''^{(3)}||OKL_1'''^{(1)})$ linearly affect the

matching states $(LL_3^{(3)}||LL_3^{(1)})$ and $(LR_3^{(3)}||LR_3^{(1)})$, respectively, we are able to regard $(LK_1'''^{(3)}||LK_1'''^{(1)}) \oplus (OKL_1'''^{(3)}||OKL_1'''^{(1)})$ as a new 16-bit key $XORK_1$. Then, $(LL_3^{(3)}||LL_3^{(1)}) \oplus (LR_3^{(3)}||LR_3^{(1)})$ is obtained from $208(= 128 + 64 + 8 + 8)$ bits of the key $\{K_2', LK_3', RKL_3'^{(3)}, RKL_3'^{(1)}\}$ and linearly-dependent 16 bits key $XORK_1$.

Backward Computation in $\mathcal{F}_{(2)}$: $((LL_4^{(1)} \oplus LL_4^{(3)})||LL_4^{(3)}) \oplus (LR_4^{(3)}||LR_4^{(1)})$ is obtained from $208(= 128 + 64 + 16)$ bits of the key $\{K_6, LK_5, RKL_5^{(1)}, RKL_5^{(3)}\}$. Using the indirect matching technique, 8 bits out of 16-bit $XORK_1$ are moved to the left half of the matching equation. Then, the left and right halves of the equation contain 216 bits of the key, i.e., $|\mathcal{K}_{(1)}| = |\mathcal{K}_{(2)}| = 216$.

Evaluation. When the parameter $N = 26$, the time complexity for the involved 432 bits is estimated as

$$C_{comp} = \max(2^{216}, 2^{216}) \times 26 = 2^{221}.$$

The remaining 352 $(= 768 - 416)$ bits are obtained by recursively applying the all-subkeys recovery attack. The time complexity for determining the remaining subkeys is roughly estimated as $2^{177.6(=352/2+1.6)}$ using 2 $(= \lceil 352/128 \rceil)$ plaintext/ciphertext pairs.

The required data is only 26 chosen plaintext/ciphertext pairs, and the required memory is about $2^{221}(= \min(2^{216}, 2^{216}) \times 26)$ blocks.

3.7 Attack on the 7-Round FOX128

If the function reduction is also used in the backward direction, the 7-round attack is feasible, i.e., two 16-bit relations of $LL_7 \oplus LR_7$ and $RL_7 \oplus RR_7$ are fixed to CON_3 and CON_4, respectively.

Due to the involution property of the FOX128 round function, $((LL_4^{(1)} \oplus LL_4^{(3)}) || LL_4^{(3)}) \oplus (LR_4^{(3)}||LR_4^{(1)})$ is also obtained from $208(= 128 + 64 + 8 + 8)$ bits of the key and linearly-dependent 16 bits key $XORK_2$. In this attack, we further regard $XORK_1 \oplus XORK_2$ as a 16 bit new key. Then, similar to the attack on the 6-round FOX128, the left and right halves of the equation contain 216 bits of the key, i.e., $|\mathcal{K}_{(1)}| = |\mathcal{K}_{(2)}| = 216$.

Repetitive ASR Approach. Recall that plaintexts and ciphertexts need to satisfy 64-bit (32×2) relations, $LL_0 \oplus LR_0$ and $RL_0 \oplus RR_0$, and $LL_7 \oplus LR_7$ and $RL_7 \oplus RR_7$, respectively. The cost is equivalently estimated as the game that an attacker finds 64-bit multicollisions with 64-bit-restricted inputs.

In the basic ASR approach, at least $27(= 432/16)$ multicollisions are needed to detect the 432-bit involved key. To obtain such pairs with a high probability, it requires $2^{65.1}(= 27!^{1/27} \cdot 2^{64 \cdot (26)/27})$ pairs. However, it is infeasible, since the degree of freedom of plaintexts is only 64 bits.

We utilize the repetitive all-subkeys recovery approach with $M = 2$ variant. In each all-subkeys recovery phase, the required data is reduced to 13 and 14. Such 14 64-bit multicollisions are obtained, given $2^{62.0}$ plaintext/ciphertext pairs with high probability.

Evaluation. The time complexity for finding involved 432 bits of the key is estimated as

$$C_{comp} = (\max(2^{216}, 2^{216}) \times 14) \times 2 + 2^{432-16 \cdot 14} = 2^{224}.$$

The remaining $480 (= 896 - 432)$ bits are obtained by recursively applying the all-subkeys recovery attack. The time complexity for this phase is roughly estimated as $2^{242 (=480/2+2)}$ using 4 $(= \lceil 480/128 \rceil)$ plaintext/ciphertext pairs.

The required data is $2^{63.0} (= 2^{62.0} \times 2)$ plaintext/ciphertext pairs, and the required memory is about 2^{242} blocks.

4 Improved All-Subkeys Recovery Attacks on KATAN32/48/64

In this section, we show that the function reduction techniques are applicable to KATAN32/48/64, then we improve the ASR attacks on KATAN32/48/64 block ciphers by 9, 5 and 5 rounds, respectively.

After a short description of KATAN, we show how to apply the function reduction to KATAN32 in detail. Then, the detailed explanation for the attack on the 119-round reduced KATAN32 is given. For KATAN48 and KATAN64, the detailed explanations for applying the function reductions are omitted, since the analysis is done similar to KATAN32.

4.1 Description of KATAN

KATAN [8] family is a feedback shift register-based block cipher consisting of three variants: KATAN32, KATAN48 and KATAN64 whose block sizes are 32-, 48- and 64-bit, respectively. All of the KATAN ciphers use the same key schedule accepting an 80-bit key and 254 rounds. The plaintext is loaded into two shift registers L_1 and L_2. In each round, L_1 and L_2 are shifted by one bit, and the least significant bits of L_1 and L_2 are updated by $f_b(L_2)$ and $f_a(L_1)$, respectively. The bit functions f_a and f_b are defined as follows:

$$f_a(L_1) = L_1[x_1] \oplus L_1[x_2] \oplus (L_1[x_3] \cdot L_1[x_4]) \oplus (L_1[x_5] \cdot IR) \oplus k_{2i},$$
$$f_b(L_2) = L_2[y_1] \oplus L_2[y_2] \oplus (L_2[y_3] \cdot L_2[y_4]) \oplus (L_2[y_5] \cdot L_2[y_6]) \oplus k_{2i+1},$$

where $L[x]$ denotes the x-th bit of L, IR denotes the round constant, and k_{2i} and k_{2i+1} denote the 2-bit i-th round key. Note that for KATAN family, the round number starts from 0 instead of 1, i.e., KATAN family consists of round functions starting from the 0-th round to the 253-th round. L_1^i or L_2^i denote the i-th round

Table 2. Parameters of KATAN family

Algorithm	$\|L_1\|$	$\|L_2\|$	x_1	x_2	x_3	x_4	x_5	y_1	y_2	y_3	y_4	y_5	y_6
KATAN32	13	19	12	7	8	5	3	18	7	12	10	8	3
KATAN48	19	29	18	12	15	7	6	28	19	21	13	15	6
KATAN64	25	39	24	15	20	11	9	38	25	33	21	14	9

registers L_1 or L_2, respectively. Let IR^i be the i-th round constant. For KATAN48 or KATAN64, in each round, the above procedure is iterated twice or three times, respectively. All of the parameters for the KATAN ciphers are listed in Table 2.

The key scheduling function of KATAN ciphers copies the 80-bit user-provided key to $k_0, ..., k_{79}$, where $k_i \in \{0, 1\}$. Then, the remaining 428 bits of the round keys are generated as follows:

$$k_i = k_{i-80} \oplus k_{i-61} \oplus k_{i-50} \oplus k_{i-13} \quad \text{for } i = 80, ..., 507.$$

4.2 Function Reduction on KATAN32

Key Linearization. In the i-th round function of KATAN32, two key bits k_{2i} and k_{2i+1} are linearly inserted into states $L_1^i[0]$ and $L_2^i[0]$, respectively, these states are not updated in the i-th round. Thus, the key linearization technique is not necessary.

Equivalent Transform. Let us consider how linearly-inserted key bits are used in the following round functions. For instance, k_1 is linearly inserted to $L_1^1[0]$, and the updated state $L_1^1[0]$ is described as $(X[0] \oplus k_1)$, where $X[i]$ is defined as

$$X[i] = L_2^0[18 - i] \oplus L_2^0[7 - i] \oplus (L_2^0[12 - i] \cdot L_2^0[10 - i]) \oplus (L_2^0[8 - i] \cdot L_2^0[3 - i]),$$

where $L_2^0[-i] = L_2^i[0]$. For computing $f_a(L_1)$, the state $L_1^1[0] = (X[0] \oplus k_1)$ is used in the following five positions,

- $L_2^4[3] : ((X[0] \oplus k_1) \cdot IR^4)$ is XORed with k_8. If $X[0]$ is fixed to a constant CON_1, a new key k'_8 is defined as $((CON_1 \oplus k_1) \cdot IR^4) \oplus k_8$.
- $L_2^6[5] : ((X[0] \oplus k_1) \cdot L_1^6[8]) = ((X[0] \oplus k_1) \cdot L_1^0[2])$ is XORed with k_{12}. If $L_1^0[2]$ is also fixed to a constant CON_2, a new key k'_{12} is defined as $((CON_1 \oplus k_1) \cdot CON_2) \oplus k_{12}$.
- $L_2^8[7] : (X[0] \oplus k_1)$ is directly XORed with k_{16}. A new key k'_{16} is defined as $(CON_1 \oplus k_1) \oplus k_{16}$.
- $L_2^9[8] : ((X[0] \oplus k_1) \cdot L_1^9[5]) = ((X[0] \oplus k_1) \cdot (X[3] \oplus k_7))$ is XORed with k_{18}. If $X[3]$ is also fixed to a constant CON_3, a new key k'_{18} is defined as $((CON_1 \oplus k_1) \cdot (CON_3 \oplus k_7)) \oplus k_{18}$.
- $L_2^{13}[12] : (X[0] \oplus k_1)$ is directly XORed with k_{26}. A new key k'_{26} is defined as $(CON_1 \oplus k_1) \oplus k_{26}$.

Thus, by fixing $X[0]$, $L_1^0[2]$ and $X[3]$ to constants and defining new key bits k'_8, k'_{12}, k'_{16}, k'_{18} and k'_{26}, we can omit one key bit k_1, i.e., we can compute without k_1 in the forward direction. Note that CON_1, CON_2 and CON_3 are not restricted to constant values. Even if CON_1, CON_2 and CON_3 are expressed by only key bits, we can define new keys in the same manner.

Table 3. Conditions for 8-bit function reductions

Key bit	State bits to be fixed
k_1	$X[0]$, $L_1^0[2]$, $X[3]$
k_3	$X[1]$, $L_1^0[1]$, $X[4]$
k_5	$X[2]$, $L_1^0[0]$, $X[5]$
k_7	$X[3]$, $X[0]$, $X[6]$
k_9	$X[4]$, $X[1]$, $X[7]$
k_{11}	$X[5]$, $X[2]$, $X[8]$
k_{13}	$X[6]$, $X[3]$, $X[9]$
k_{15}	$X[7]$, $X[4]$, $X[10]$

Conditions for Function Reduction. Table 3 shows conditions for 8-bit function reductions. If these 14 bits of $L_1^0[0]$, $L_1^0[2]$, $L_1^0[2]$, $X[0], \ldots, X[10]$ are fixed to constants or expressed by only key bits, then we can eliminate 8 bits of the key, k_1, k_3, k_5, k_7, k_9, k_{11}, k_{13} and k_{15}, in the forward computation of KATAN32.

Let us explain how to control $X[0]$ and $X[10]$ by exploiting the degree of freedom of plaintexts. $X[0]$ to $X[10]$ are expressed as follows:

$$
\begin{aligned}
X[0] &= \underline{L_2^0[18]} \oplus L_2^0[7] \oplus (L_2^0[12] \cdot L_2^0[10]) \oplus (L_2^0[8] \cdot L_2^0[3]),\\
X[1] &= \underline{L_2^0[17]} \oplus L_2^0[6] \oplus (L_2^0[11] \cdot L_2^0[9]) \oplus (L_2^0[7] \cdot L_2^0[2]),\\
X[2] &= \underline{L_2^0[16]} \oplus L_2^0[5] \oplus (L_2^0[10] \cdot L_2^0[8]) \oplus (L_2^0[6] \cdot L_2^0[1]),\\
X[3] &= \underline{L_2^0[15]} \oplus L_2^0[4] \oplus (L_2^0[9] \cdot L_2^0[7]) \oplus (L_2^0[5] \cdot L_2^0[0]),\\
X[4] &= \underline{L_2^0[14]} \oplus L_2^0[3] \oplus (L_2^0[8] \cdot L_2^0[6]) \oplus (\underline{L_2^0[4]} \cdot (Y[0] \oplus k_0)),\\
X[5] &= \underline{L_2^0[13]} \oplus L_2^0[2] \oplus (L_2^0[7] \cdot L_2^0[5]) \oplus (\underline{L_2^0[3]} \cdot (Y[1] \oplus k_2)),\\
X[6] &= \underline{L_2^0[12]} \oplus L_2^0[1] \oplus (L_2^0[6] \cdot L_2^0[4]) \oplus (\underline{L_2^0[2]} \cdot (Y[2] \oplus k_4)),\\
X[7] &= \underline{L_2^0[11]} \oplus L_2^0[0] \oplus (L_2^0[5] \cdot L_2^0[3]) \oplus (\underline{L_2^0[1]} \cdot (Y[3] \oplus k_6)),\\
X[8] &= \underline{L_2^0[10]} \oplus (Y[0] \oplus k_0) \oplus (L_2^0[4] \cdot L_2^0[2]) \oplus (\underline{L_2^0[0]} \cdot (Y[4] \oplus k_8)),\\
X[9] &= \underline{L_2^0[9]} \oplus (Y[1] \oplus k_2) \oplus (L_2^0[3] \cdot L_2^0[1]) \oplus ((\underline{Y[0]} \oplus k_0) \cdot (\underline{Y[5]} \oplus k_{10}),\\
X[10] &= \underline{L_2^0[8]} \oplus (Y[2] \oplus k_4) \oplus (L_2^0[2] \cdot L_2^0[0]) \oplus ((\underline{Y[1]} \oplus k_2) \cdot (\underline{Y[6]} \oplus k_{12}),
\end{aligned}
$$

where

$$
\begin{aligned}
Y[0] &= \underline{L_1^0[12]} \oplus L_1^0[7] \oplus (L_1^0[5] \cdot L_1^0[8]) \oplus (L_1^0[3] \cdot IR^0),\\
Y[1] &= \underline{L_1^0[11]} \oplus L_1^0[6] \oplus (L_1^0[4] \cdot L_1^0[7]) \oplus (L_1^0[2] \cdot IR^1),\\
Y[2] &= \underline{L_1^0[10]} \oplus L_1^0[5] \oplus (L_1^0[3] \cdot L_1^0[6]) \oplus (L_1^0[1] \cdot IR^2),\\
Y[3] &= L_1^0[9] \oplus L_1^0[4] \oplus (L_1^0[2] \cdot L_1^0[5]) \oplus (L_1^0[0] \cdot IR^3),\\
Y[4] &= L_1^0[8] \oplus L_1^0[3] \oplus (L_1^0[1] \cdot L_1^0[4]) \oplus (X[0] \cdot IR^4),\\
Y[5] &= \underline{L_1^0[7]} \oplus L_1^0[2] \oplus (L_1^0[0] \cdot L_1^0[3]) \oplus (X[1] \cdot IR^5),\\
Y[6] &= \underline{L_1^0[6]} \oplus L_1^0[1] \oplus (X[0] \cdot L_1[2]) \oplus (X[2] \cdot IR^6).
\end{aligned}
$$

$X[0], \ldots, X[3]$ are easily fixed to constants by controlling 4 bits of $L_2^0[18]$, $L_2^0[17]$, $L_2^0[16]$ and $L_2^0[15]$ (4-bit condition). $X[4], \ldots, X[8]$ contain key bits in AND operations. If $L_2^0[4] = L_2^0[3] = L_2^0[2] = L_2^0[1] = L_2^0[0] = 0$, those key bits are omitted from the equations (5-bit condition). Then $X[4]$ to $X[7]$ are also fixed to constants by controlling 4 bits of $L_2^0[14]$, $L_2^0[13]$, $L_2^0[12]$ and $L_2^0[11]$ (4 bit condition). In $X[8]$, k_0 is also linearly inserted. If $(L_2^0[10] \oplus Y[0] \oplus (L_2^0[4] \cdot L_2^0[2]))$ is fixed to a constant by

controlling $L_2^0[10]$, then $X[8]$ is expressed as the form of $CON \oplus k_0$, which depends only on key bits, where CON is an arbitrary constant.

In $X[9]$ and $X[10]$, 4 bits of $Y[0]$, $Y[1]$, $Y[5]$ and $Y[6]$ are needed to be fixed. These values are controlled by $L_1^0[12]$, $L_1^0[11]$, $L_1^0[7]$ and $L_1^0[6]$. If the other bits are fixed by $L_2^0[9]$ and $L_2^0[8]$, $X[9]$ and $X[10]$ are expressed by only key bits.

Therefore, if plaintexts satisfy $23(= 3 + 4 + 5 + 4 + 1 + 4 + 2)$ bit conditions described in Table 3, 8 bits of the key are able to be omitted when mounting the ASR attack.

Procedure for Creating Plaintexts. We show how to create plaintexts satisfying the conditions. By using the equations of $X[0]$ to $X[10]$ and $Y[0]$ to $Y[6]$, such plaintexts are easily obtained as follows.

1. Set 18 predetermined values of $L_1^0[0]$, $L_1^0[1]$, $L_1^0[2]$, $X[0], \ldots, X[10]$, $Y[0]$, $Y[1]$, $Y[5]$ and $Y[6]$.
2. Choose values of free 9 bits of $L_2^0[5]$, $L_2^0[6]$, $L_2^0[7]$, $L_1^0[3]$, $L_1^0[4]$, $L_1^0[5]$, $L_1^0[8]$, $L_1^0[9]$ and $L_1^0[10]$.
3. Obtain $L_2^0[8], \ldots, L_2^0[13]$ from equations of $X[5], \ldots, X[10]$, and $L_1^0[6]$ and $L_1^0[7]$ from equations of $Y[5]$ and $Y[6]$, respectively.
4. Obtain $L_2^0[14], \ldots, L_2^0[18]$ from equations of $X[0], \ldots, X[4]$, and $L_1^0[11]$ and $L_1^0[12]$ from equations of $Y[0]$ and $Y[1]$, respectively.
5. Repeat steps 2 to 4 until the required number of plaintexts are obtained.

4.3 Attacks on 119-Round KATAN32

Let us consider the 119-round variant of KATAN32 starting from the first (0-th) round. In this attack, $L_2^{69}[18]$ is chosen as the matching state.

Forward Computation in $\mathcal{F}_{(1)}$: $L_2^{69}[18]$ depends on 83 subkey bits. This implies that $L_2^{69}[18]$ can be computed by a plaintext P and 83 bits of subkeys. More specifically, $L_2^{69}[18] = \mathcal{F}_{(1)}(P, \mathcal{K}_{(1)})$, where $\mathcal{K}_{(1)} \in \{k_0, ..., k_{70}, k_{72}, \ldots, k_{76}, k_{80}, k_{83}, k_{84}, k_{85}, k_{89}, k_{93}, k_{100}\}$ and $|\mathcal{K}_{(1)}| = 83$. If the function reduction technique with the 23-bit condition of plaintexts is used, 8 bits of $\{k_1, k_3, k_5, k_7, k_9, k_{11}, k_{13}, k_{15}\}$ can be omitted in computations of $\mathcal{F}_{(1)}$. Thus, $L_2^{69}[18]$ is computable with $75(= 83 - 8)$ bits. In addition, since 4 bits of $\{k_{68}, k_{75}, k_{85}, k_{100}\}$ linearly affect $L_2^{69}[18]$, we can regard $k_{68} \oplus k_{75} \oplus k_{85} \oplus k_{100}$ as a new key k_f. Thus, $72(= 75 - 4 + 1)$ bits are involved in the forward computation.

Backward Computation in $\mathcal{F}_{(2)}$: In the backward computation starting from the 118-th round, the matching state $L_2^{69}[18]$ is computed as $L_2^{69}[18] = \mathcal{F}_{(2)}^{-1}(C, \mathcal{K}_{(2)})$, where $\mathcal{K}_{(2)} \in \{k_{138}, k_{150}, k_{154}, k_{158}, k_{160}, k_{162}, k_{165}, k_{166}, k_{168}, k_{170}, k_{172}, \ldots k_{237}\}$, and $|\mathcal{K}_{(2)}| = 76$. Since 4 bits of $\{k_{138}, k_{160}, k_{165}, k_{175}\}$ linearly affect $L_2^{69}[18]$, we can regard $k_{138} \oplus k_{160} \oplus k_{165} \oplus k_{175}$ as a new key k_b. Furthermore, by the indirect matching, k_b is moved to the forward computation, then $k_b \oplus k_f$ is regarded as a new key in $\mathcal{F}_{(1)}$. Thus, $72(= 76 - 4)$ bits are involved in the backward computation.

Evaluation. For the 119-round reduced KATAN32, the matching state S is chosen as $L_2^{69}[18]$ (1-bit state). When $N = 144$ ($\leq (72+72)/1$), the time complexity for finding $\mathcal{K}_{(1)}$ and $\mathcal{K}_{(2)}$ is estimated as

$$C_{comp} = \max(2^{72}, 2^{72}) \times 144 = 2^{79.1}.$$

The required data is only 144 chosen plaintext/ciphertext pairs in which the 23 bits of each plaintext satisfy conditions. The required memory is about $2^{79.1}$ blocks.

Finally, we need to find the remaining $94 (= 119 \times 2 - 144)$ bits of subkeys by using the simple MITM approach in the setting where $\mathcal{K}_{(1)}$ and $\mathcal{K}_{(2)}$ are known. The time complexity and the required memory for this process are roughly estimated as $2^{49} (= 2^{48} + 2^{46})$ and 2^{46} blocks, respectively. These costs are obviously much less than those of finding $\mathcal{K}_{(1)}$ and $\mathcal{K}_{(2)}$.

4.4 Function Reduction on KATAN48

Table 4 shows conditions for 4-bit function reductions, where $X'[i]$ is defined as:

$$X'[i] = L_2^0[28-i] \oplus L_2^0[19-i] \oplus (L_2^0[21-i] \cdot L_2^0[13-i]) \oplus (L_2^0[15-i] \cdot L_2^0[6-i]).$$

If these values are fixed to target constants, we can eliminate the key bits in the computation of KATAN48.

Table 4. Conditions for 4-bit function reductions

Key bit	State bits to be fixed
k_1	$X'[0]$, $X'[1]$, $L_1^0[6]$, $L_1^0[7]$, $X'[8]$, $X'[9]$
k_3	$X'[2]$, $X'[3]$, $L_1^0[4]$, $L_1^0[5]$, $X'[10]$, $X'[11]$
k_5	$X'[4]$, $X'[5]$, $L_1^0[2]$, $L_1^0[3]$, $X'[12]$, $X'[13]$
k_7	$X'[6]$, $X'[7]$, $L_1^0[0]$, $L_1^0[1]$, $X'[14]$, $X'[15]$

$X'[0], \ldots, X'[6]$ are fixed by controlling 7 bits of $L_2^0[22], \ldots, L_2^0[28]$ (7 bit condition). $X'[7], \ldots, X'[15]$ contain key bits in AND operations. If $L_2^0[8] = L_2^0[7] = , \ldots, = L_2^0[1] = L_2^0[0] = 0$, these key bits are omitted from these equations (9 bit condition). Then $X'[7], \ldots, X'[15]$ are also fixed by controlling 9 bits of $L_2^0[13], \ldots, L_2^0[21]$ (9 bit condition).

Therefore, if plaintexts satisfy 33 $(= 8 + 7 + 9 + 9)$ bit conditions described in Table 4, 4 bits of the key are able to be omitted when mounting the ASR attack.

4.5 Attacks on 105-Round KATAN48

Let us consider the 105-round variant of KATAN48 starting from the first (0-th) round. In this attack, $L_2^{61}[28]$ is chosen as the matching state.

Forward Computation in $\mathcal{F}_{(1)}$: $L_2^{61}[28]$ depends on 79 subkey bits. This implies that $L_2^{61}[28]$ can be computed by a plaintext P and 79 bits of subkeys. More specifically, $L_2^{61}[28] = \mathcal{F}_{(1)}(P, \mathcal{K}_{(1)})$, where $\mathcal{K}_{(1)} \in \{k_0, ..., k_{68}, k_{70}, k_{71}, k_{72}, k_{75}, k_{77}, k_{78}, k_{81}, k_{85}, k_{87}, k_{92}\}$ and $|\mathcal{K}_{(1)}| = 79$. If the function reduction technique with the 33-bit condition of plaintexts is used, 4 bits of k_1, k_3, k_5, k_7 can be omitted in computations of $\mathcal{F}_{(1)}$. Thus, $L_2^{61}[28]$ is computable with $75(= 79 - 4)$ bits. In addition, 3 bits of $\{k_{75}, k_{81}, k_{92}\}$ linearly affect $L_2^{61}[28]$. Thus, we can regard $k_{75} \oplus k_{81} \oplus k_{92}$ as a new key. By using indirect matching, $k_f = k_{75} \oplus k_{81} \oplus k_{92}$ is moved to $\mathcal{F}_{(2)}$. Then, $72(= 75 - 3)$ bits are involved in the forward computation.

Backward Computation in $\mathcal{F}_{(2)}$: In the backward computation starting from the 104-th round, the matching state $L_2^{61}[28]$ is computed as $L_2^{61}[28] = \mathcal{F}_{(2)}^{-1}(C, \mathcal{K}_{(2)})$, where $\mathcal{K}_{(2)} \in \{k_{122}, k_{128}, k_{130}, k_{134}, k_{136}, k_{138}, k_{140}, k_{141}, k_{142}, k_{144}, \ldots k_{209}\}$, and $|\mathcal{K}_{(2)}| = 75$. 4 bits of $\{k_{122}, k_{130}, k_{140}, k_{141}\}$ linearly affect $L_2^{61}[28]$. Thus, we can regard $k_b = k_{122} \oplus k_{130} \oplus k_{140} \oplus k_{141}$ as a new key. Furthermore, we define $k_f \oplus k_b$ as a new key. Then, $72(= 75-4+1)$ bits are involved in the backward computation.

Evaluation. For the 105-round reduced `KATAN48`, the matching state S is chosen as $L_2^{61}[28]$ (1-bit state). When $N = 144$ ($\leq (72 + 72)/1$), the time complexity for finding $\mathcal{K}_{(1)}$ and $\mathcal{K}_{(2)}$ is estimated as

$$C_{comp} = \max(2^{72}, 2^{72}) \times 144 = 2^{79.1}.$$

The required data is only 144 chosen plaintext/ciphertext pairs. The required memory is about $2^{79.1}$ blocks.

Finally, we need to find the remaining 66 ($= 105 \times 2 - 144$) bits of subkeys by using the simple MITM approach in the setting where $\mathcal{K}_{(1)}$ and $\mathcal{K}_{(2)}$ are known. The time complexity and the required memory for this process are roughly estimated as $2^{34}(= 2^{34} + 2^{32})$ and 2^{32} blocks, respectively. These costs are obviously much less than those of finding $\mathcal{K}_{(1)}$ and $\mathcal{K}_{(2)}$.

4.6 Function Reduction on `KATAN64`

Table 5 shows conditions for 2-bit function reductions, where $X''[i]$ is defined as:

$$X''[i] = L_2^0[38 - i] \oplus L_2^0[25 - i] \oplus (L_2^0[33 - i] \cdot L_2^0[21 - i]) \oplus (L_2^0[14 - i] \cdot L_2^0[9 - i]).$$

If these values are fixed to target constants, we can eliminate the key bits in the computation of `KATAN64`.

Table 5. Conditions for 2-bit function reductions

Key bit	State bits to be fixed
k_1	$X''[0]$, $X''[1]$, $X''[2]$, $L_1^0[6]$, $L_1^0[7]$, $L_1^0[8]$, $X''[9]$, $X''[10]$, $X''[11]$,
k_3	$X''[3]$, $X''[4]$, $X''[5]$, $L_1^0[3]$, $L_1^0[4]$, $L_1^0[5]$, $X''[12]$, $X''[13]$, $X''[14]$,

$X''[0], \ldots, X''[9]$ are fixed by controlling 10 bits of $L_2^0[29], \ldots, L_2^0[38]$ (10 bit condition). $X''[10]$, ..., $X''[14]$ contain key bits in AND operations. If $L_2^0[4] = , \ldots, = L_2^0[0] = 0$, these key bits are omitted from these equations (5 bit condition). Then $X''[10], \ldots, X''[14]$ are also fixed by controlling 5 bits $L_2^0[18]$, ..., $L_2^0[23]$ (5 bit condition).

Therefore, if plaintexts satisfy $29(= 9 + 10 + 5 + 5)$ bit conditions described in Table 5, 2 bits of the key are able to be omitted when mounting the ASR attack.

4.7 Attacks on 99-Round KATAN64

Let us consider the 99-round variant of KATAN64 starting from the first (0-th) round. In this attack, $L_2^{57}[38]$ is chosen as the matching state.

Forward Computation in $\mathcal{F}_{(1)}$: $L_2^{57}[38]$ depends on 74 subkey bits. This implies that $L_2^{57}[38]$ can be computed by a plaintext P and 74 bits of subkeys. More specifically, $L_2^{57}[38] = \mathcal{F}_{(1)}(P, \mathcal{K}_{(1)})$, where $\mathcal{K}_{(1)} \in \{k_0, ..., k_{66}, k_{70}, k_{71}, k_{72}, k_{75}, k_{77}, k_{81}, k_{88}\}$ and $|\mathcal{K}_{(1)}| = 74$. If the function reduction technique with the 29-bit condition of plaintexts is used, 2 bits of k_1, k_3 can be omitted in computations of $\mathcal{F}_{(1)}$. Thus, $L_2^{57}[38]$ is computable with $72(= 74 - 2)$ bits. In addition, 3 bits of $\{k_{71}, k_{77}, k_{88}\}$ linearly affect $L_2^{57}[38]$. Thus, we can regard $k_{71} \oplus k_{77} \oplus k_{88}$ as a new key. Then, $70(= 72 - 3 + 1)$ bits are involved in the forward computation.

Backward Computation in $\mathcal{F}_{(2)}$: In the backward computation starting from the 98-th round, the matching state $L_2^{57}[38]$ is computed as $L_2^{57}[38] = \mathcal{F}_{(2)}^{-1}(C, \mathcal{K}_{(2)})$, where $\mathcal{K}_{(2)} \in \{k_{114}, k_{116}, k_{120}, k_{122}, k_{124}, k_{126}, k_{128}, k_{130}, \ldots k_{197}\}$, and $|\mathcal{K}_{(2)}| = 75$. 3 bits of $\{k_{114}, k_{122}, k_{131}\}$ linearly affect $L_2^{57}[38]$. Thus, we can consider $k_{114} \oplus k_{122} \oplus k_{131}$ as a new key, and move it to the forward computation by the indirect matching. Then, $72(= 75 - 3)$ bits are involved in the backward computation.

Evaluation. For the 99-round reduced KATAN64, the matching state S is chosen as $L_2^{57}[38]$ (1-bit state).

When $N = 142$ $(\leq (72 + 70)/1)$, the time complexity for finding $\mathcal{K}_{(1)}$ and $\mathcal{K}_{(2)}$ is estimated as

$$C_{comp} = \max(2^{72}, 2^{70}) \times 142 = 2^{79.1}.$$

The required data is only 142 chosen plaintext/ciphertext pairs. The required memory is about $2^{77.1}$ blocks.

Finally, we need to find the remaining $56(= 99 \times 2 - 142)$ bits of subkeys by using the simple MITM approach in the setting where $\mathcal{K}_{(1)}$ and $\mathcal{K}_{(2)}$ are known. The time complexity and the required memory for this process are roughly estimated as 2^{28} and 2^{28} blocks, respectively. These costs are obviously much less than those of finding $\mathcal{K}_{(1)}$ and $\mathcal{K}_{(2)}$.

5 Improved All-Subkeys Recovery Attack on SHACAL-2

This section presents the ASR attacks on SHACAL-2 with the function reduction techniques. Then, we propose a 42-round attack on SHACAL-2, based on the 41-round attack on SHACAL-2 [13].

5.1 Description of SHACAL-2

SHACAL-2 [13] is a 256-bit block cipher based on the compression function of SHA-256 [12]. It was submitted to the NESSIE project and selected in the NESSIE portfolio [22].

SHACAL-2 inputs the plaintext to the compression function as the chaining variable, and inputs the key to the compression function as the message block. First, a 256-bit plaintext is divided into eight 32-bit words A_0, B_0, C_0, D_0, E_0, F_0, G_0 and H_0. Then, the state update function updates eight 32-bit variables, A_i, B_i, ..., G_i, H_i in 64 steps as follows:

$$\begin{aligned} T_1 &= H_i \boxplus \Sigma_1(E_i) \boxplus Ch(E_i, F_i, G_i) \boxplus K_i \boxplus W_i, \\ T_2 &= \Sigma_0(A_i) \boxplus Maj(A_i, B_i, C_i), \\ A_{i+1} &= T_1 \boxplus T_2,\ B_{i+1} = A_i,\ C_{i+1} = B_i,\ D_{i+1} = C_i, \\ E_{i+1} &= D_i \boxplus T_1,\ F_{i+1} = E_i,\ G_{i+1} = F_i,\ H_{i+1} = G_i, \end{aligned}$$

where K_i is the i-th step constant, W_i is the i-th step key (32-bit), and the functions Ch, Maj, Σ_0 and Σ_1 are given as follows:

$$\begin{aligned} Ch(X,Y,Z) &= XY \oplus \overline{X}Z, \\ Maj(X,Y,Z) &= XY \oplus YZ \oplus XZ, \\ \Sigma_0(X) &= (X \ggg 2) \oplus (X \ggg 13) \oplus (X \ggg 22), \\ \Sigma_1(X) &= (X \ggg 6) \oplus (X \ggg 11) \oplus (X \ggg 25). \end{aligned}$$

After 64 steps, the function outputs eight 32-bit words $A_{64}, B_{64}, C_{64}, D_{64}, E_{64}, F_{64}$, G_{64} and H_{64} as the 256-bit ciphertext. Hereafter p_i denotes the i-th step state, i.e., $p_i = A_i||B_i||...||H_i$.

The key scheduling function of SHACAL-2 takes a variable length key up to 512 bits as the inputs, then outputs 64 32-bit step keys. First, the 512-bit input key is copied to 16 32-bit words W_0, W_1, ..., W_{15}. If the size of the input key is shorter than 512 bits, the key is padded with zeros. Then, the key scheduling function generates 48 32-bit step keys $(W_{16}, ..., W_{63})$ from the 512-bit key $(W_0, ..., W_{15})$ as follows:

$$W_i = \sigma_1(W_{i-2}) \boxplus W_{i-7} \boxplus \sigma_0(W_{i-15}) \boxplus W_{i-16}, (16 \leq i < 64),$$

where the functions $\sigma_0(X)$ and $\sigma_1(X)$ are defined by

$$\begin{aligned} \sigma_0(X) &= (X \ggg 7) \oplus (X \ggg 18) \oplus (X \gg 3), \\ \sigma_1(X) &= (X \ggg 17) \oplus (X \ggg 19) \oplus (X \gg 10). \end{aligned}$$

5.2 Function Reduction on SHACAL-2

In the round function of SHACAL-2, a round key W_i is inserted to the state T_i by an arithmetic addition operation. We show that the splice and cut framework is applicable by using the partial key linearization technique.

The computation of T_1 is expressed as

$$T_1 = (H_i \boxplus W_i) \boxplus \Sigma_1(E_i) \boxplus Ch(E_i, F_i, G_i) \boxplus K_i.$$

In a straight way, the computation of $(H_i \boxplus W_i)$ is not divided into two parts as $(HL_i \boxplus WL_i)||(HR_i \boxplus WR_i)$ due to the carry bit between these computations, where HL_i and WL_i denote the higher x-bits of H_i and W_i, respectively, and HR_i and WR_i are the lower $(32-x)$-bits of H_i and W_i. If HR_i is fixed to 0, it is equivalent to $(HL_i \boxplus WL_i)||(HL_i \oplus WR_i)$. Then, it allows us to independently compute these two parts without dealing with carry bits. Therefore, by using the splice and cut framework, 32 key bits of one round is divided into forward and backward computations as shown in Fig. 5.

However, we can not reduce the number of involved key bits by using an equivalent transform. It is because that the involved 32-bit key W_i is used at least eight times in the forward and backward directions. In order to fully control values in each state, more than $512(32 \times 8)$ bits of conditions are required.

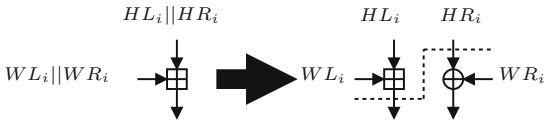


Fig. 5. Splice and cut of SHACAL-2

5.3 Attacks on 42-Round SHACAL-2

We show that the splice and cut framework [4] is applicable to SHACAL-2 by using the key linearization technique. Then we extend the 41-round attack [16] by one more round. In particular, the splice and cut technique is done in the first round, and the higher 15 bits are move to the backward computation, and the lower 17 bits are move to the forward computation. Then we choose the lowest 1 bit of A_{17} as the matching point.

Forward Computation in $\mathcal{F}_{(1)}$: The lowest 1 bit of A_{17} can be computed from the 16-th state p_{16} and the lowest 1 bit of W_{16}, since the other bits of W_{16} are not affected to the lower 1 bit of A_{17}. Thus, the matching state S (the lowest 1 bit of A_{17}) is calculated as $S = \mathcal{F}_{(1)}(P, \mathcal{K}_{(1)})$, where $\mathcal{K}_{(1)} \in \{$the lower 17 bits of $W_0, W_1, ..., W_{15}$, the lowest 1 bit of $W_{16}\}$ and $|\mathcal{K}_{(1)}| = 498 (= 32 \times 15 + 1 + 17)$.

Backward Computation in $\mathcal{F}_{(2)}$: We utilize the following observation [16].

Observation 1. *The lower t bits of A_{j-10} are obtained from the j-th state p_j and the lower t bits of three subkeys W_{j-1}, W_{j-2} and W_{j-3}.*

From Observation 1, the matching state S (the lowest 1 bit of A_{17}) can be computed as $S = \mathcal{F}^{-1}_{(2)}(C, \mathcal{K}_{(2)})$, where $\mathcal{K}_{(2)} \in \{$the higher 15 bits of W_0, $W_{27}, ..., W_{41}$, the lowest 1 bits of W_{24}, W_{25} and $W_{26}\}$. Thus, $|\mathcal{K}_{(2)}| = 498 (= 32 \times 15 + 1 \times 3 + 15)$.

Evaluation. The matching state S is the lowest 1 bit of A_{17}, $|\mathcal{K}_{(1)}| = 498$ and $|\mathcal{K}_{(2)}| = 498$. Thus, using 996 chosen plaintext/ciphertext pairs (i.e. $N = 244 \leq (498 + 498)/1$), the time complexity for finding all-subkeys is estimated as

$$C_{comp} = \max(2^{498}, 2^{498}) \times 996 + 2^{1344-996} = 2^{508}.$$

The required data is $2^{25} (= 996 \times 2^{15})$ chosen plaintext/ciphertext pairs, since 15 bits of plaintext are not controlled in the backward computation when using the splice and cut technique. The required memory is $2^{508} (= \min(2^{498}, 2^{498}) \times 996)$ blocks.

6 Conclusion

The concept of the ASR attack is quite simple, which recovers all-subkeys instead of the master key, but useful to evaluate the security of block cipher structures without analyzing key scheduling functions. Thus, it is valuable to study its improvements to design a secure block cipher. We first observed the function reduction technique, which improved the ASR attack and was originally applied to Feistel schemes. Then, with some improvements such as the repetitive ASR approach, we applied the function reduction to other block cipher structures including Lai-Massey, generalized Lai-Massey, LFSR-type and source-heavy generalized Feistel schemes.

As applications of our approach, we presented the improved ASR attacks on the 7-, 7-, 119-, 105-, 99-, and 42-round reduced `FOX64`, `FOX128`, `KATAN32`, `KATAN48`, `KATAN64` and `SHACAL-2`. All of our results updated the number of attacked rounds by the previously known best attacks. We emphasize that our attacks work independently from the structure of the key scheduling function. In other words, strengthening the key scheduling function does not improve the security against our attack. It implies that our results give the lower bounds on the security of the target structures such as Lai-Massey scheme rather than the specific block ciphers against generic key recovery attack. Therefore, we believe that our results are useful for a deeper understanding the security of the block cipher structures.

References

1. Adams, C.: The CAST-128 encryption algorithm. RFC-2144, May 1997
2. Albrecht, M.R., Leander, G.: An all-in-one approach to differential cryptanalysis for small block ciphers. In: Knudsen, L.R., Wu, H. (eds.) SAC 2012. LNCS, vol. 7707, pp. 1–15. Springer, Heidelberg (2013)
3. Aoki, K., Guo, J., Matusiewicz, K., Sasaki, Y., Wang, L.: Preimages for step-reduced SHA-2. In: Matsui, M. (ed.) ASIACRYPT 2009. LNCS, vol. 5912, pp. 578–597. Springer, Heidelberg (2009)
4. Aoki, K., Sasaki, Y.: Preimage attacks on one-block MD4, 63-step MD5 and more. In: Avanzi, R.M., Keliher, L., Sica, F. (eds.) SAC 2008. LNCS, vol. 5381, pp. 103–119. Springer, Heidelberg (2009)

5. Biham, E., Dunkelman, O., Keller, N., Shamir, A.: New data-efficient attacks on reduced-round IDEA. IACR Cryptology ePrint Archive, vol. 2011, p. 417 (2011)
6. Bogdanov, A., Khovratovich, D., Rechberger, C.: Biclique cryptanalysis of the full AES. In: Lee, D.H., Wang, X. (eds.) ASIACRYPT 2011. LNCS, vol. 7073, pp. 344–371. Springer, Heidelberg (2011)
7. Bogdanov, A., Rechberger, C.: A 3-subset meet-in-the-middle attack: cryptanalysis of the lightweight block cipher KTANTAN. In: Biryukov, A., Gong, G., Stinson, D.R. (eds.) SAC 2010. LNCS, vol. 6544, pp. 229–240. Springer, Heidelberg (2011)
8. De Cannière, C., Dunkelman, O., Knežević, M.: KATAN and KTANTAN — A family of small and efficient hardware-oriented block ciphers. In: Clavier, C., Gaj, K. (eds.) CHES 2009. LNCS, vol. 5747, pp. 272–288. Springer, Heidelberg (2009)
9. Canteaut, A., Naya-Plasencia, M., Vayssière, B.: Sieve-in-the-middle: improved MITM attacks. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013, Part I. LNCS, vol. 8042, pp. 222–240. Springer, Heidelberg (2013)
10. Dinur, I., Dunkelman, O., Shamir, A.: Improved attacks on full GOST. In: Canteaut, A. (ed.) FSE 2012. LNCS, vol. 7549, pp. 9–28. Springer, Heidelberg (2012)
11. Dunkelman, O., Sekar, G., Preneel, B.: Improved meet-in-the-middle attacks on reduced-round DES. In: Srinathan, K., Rangan, C.P., Yung, M. (eds.) INDOCRYPT 2007. LNCS, vol. 4859, pp. 86–100. Springer, Heidelberg (2007)
12. FIPS. Secure Hash Standard (SHS). Federal Information Processing Standards Publication, pp. 180–184
13. Handschuh, H., Naccache, D.: "SHACAL." NESSIE Proposal (updated), October 2001. https://www.cosic.esat.kuleuven.be/nessie/updatedPhase2Specs/SHACAL/shacal-tweak.zip
14. Isobe, T.: A single-key attack on the full GOST block cipher. In: Joux, A. (ed.) FSE 2011. LNCS, vol. 6733, pp. 290–305. Springer, Heidelberg (2011)
15. Isobe, T., Sasaki, Y., Chen, J.: Related-key boomerang attacks on KATAN32/48/64. In: Boyd, C., Simpson, L. (eds.) ACISP. LNCS, vol. 7959, pp. 268–285. Springer, Heidelberg (2013)
16. Isobe, T., Shibutani, K.: All subkeys recovery attack on block ciphers: extending meet-in-the-middle approach. In: Knudsen, L.R., Wu, H. (eds.) SAC 2012. LNCS, vol. 7707, pp. 202–221. Springer, Heidelberg (2013)
17. Isobe, T., Shibutani, K.: Generic key recovery attack on feistel scheme. In: Sako, K., Sarkar, P. (eds.) ASIACRYPT 2013, Part I. LNCS, vol. 8269, pp. 464–485. Springer, Heidelberg (2013)
18. Junod, P., Vaudenay, S.: FOX: a new family of block ciphers. In: Handschuh, H., Hasan, M.A. (eds.) SAC 2004. LNCS, vol. 3357, pp. 114–129. Springer, Heidelberg (2004)
19. Khovratovich, D., Leurent, G., Rechberger, C.: Narrow-bicliques: cryptanalysis of full IDEA. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT 2012. LNCS, vol. 7237, pp. 392–410. Springer, Heidelberg (2012)
20. Lu, J., Kim, J.: Attacking 44 rounds of the SHACAL-2 block cipher using related-key rectangle cryptanalysis. IEICE Trans. **91**–**A**(9), 2588–2596 (2008)
21. Merkle, R.C., Hellman, M.E.: On the security of multiple encryption. Commun. ACM **24**(7), 465–467 (1981)
22. NESSIE consortium, NESSIE portfolio of recommended cryptographic primitives (2003). https://www.cosic.esat.kuleuven.be/nessie/deliverables/decision-final.pdf
23. Nikolic, I., Wang, L., Wu, S.: The parallel-cut meet-in-the-middle attack. IACR Cryptology ePrint Archive, p. 530 (2013)

24. Sasaki, Y., Aoki, K.: Finding preimages in full MD5 faster than exhaustive search. In: Joux, A. (ed.) EUROCRYPT 2009. LNCS, vol. 5479, pp. 134–152. Springer, Heidelberg (2009)
25. Suzuki, K., Tonien, D., Kurosawa, K., Toyota, K.: Birthday paradox for multi-collisions. In: Rhee, M.S., Lee, B. (eds.) ICISC 2006. LNCS, vol. 4296, pp. 29–40. Springer, Heidelberg (2006)
26. Wu, W., Zhang, W., Feng, D.: Integral cryptanalysis of reduced FOX block cipher. In: Won, D.H., Kim, S. (eds.) ICISC 2005. LNCS, vol. 3935, pp. 229–241. Springer, Heidelberg (2006)
27. Wu, Z., Luo, Y., Lai, X., Zhu, B.: Improved cryptanalysis of the FOX block cipher. In: Chen, L., Yung, M. (eds.) INTRUST 2009. LNCS, vol. 6163, pp. 236–249. Springer, Heidelberg (2010)
28. Zhu, B., Gong, G.: Guess-then-meet-in-the-middle attacks on the KTANTAN family of block ciphers. IACR Cryptology ePrint Archive, p. 619 (2011)

Improved Single-Key Attacks on 9-Round AES-192/256

Leibo Li[1], Keting Jia[2], and Xiaoyun Wang[1,3](✉)

[1] Key Laboratory of Cryptologic Technology and Information Security, Ministry of Education, School of Mathematics, Shandong University, Jinan 250100, China
lileibo@mail.sdu.edu.cn

[2] Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China

[3] Institute for Advanced Study, Tsinghua University, Beijing 100084, China
{ktjia,xiaoyunwang}@mail.tsinghua.edu.cn

Abstract. This paper focuses on key-recovery attacks on 9-round AES-192 and AES-256 under single-key model with the framework of the meet-in-the-middle attack. A new technique named *key-dependent sieve* is introduced to further reduce the size of lookup table of the attack, and the 9-round AES-192 is broken with 2^{121} chosen plaintexts, $2^{187.5}$ 9-round encryptions and 2^{185} 128-bit words of memory. If the attack starts from the third round, the complexities would be further reduced by a factor of 16. Moreover, the whole attack is split up into a series of weak-key attacks. Then the memory complexity of the attack is saved significantly when we execute these weak attacks in streaming mode. This method is also applied to reduce the memory complexity of the attack on 9-round AES-256.

Keywords: AES · Block cipher · Meet-in-the-Middle Attack · Differential characteristic

1 Introduction

The block cipher Rijndael was designed by Daemen and Rijmen in 1997, and was selected as the Advanced Encryption Standard (AES) in 2001 by NIST. It is a Substitution-Permutation Network (SPN) with variable key length of 128, 192, 256, which are denoted as AES-128, AES-192 and AES-256, respectively.

For the reason of its importance and popularity, the security of AES has attracted a great amount of attention from worldwide cryptology researchers. Many methods of cryptanalysis were applied to attack AES in previous years, such as impossible differential attack [15,16,19], SQUARE attack [5], collision attack [13], meet-in-the-middle attack [6–9,11,18], biclique attack [4], related-key attack and chosen-key distinguishing [1–3,12]. Although the attacks under

C. Cid and C. Rechberger (Eds.): FSE 2014, LNCS 8540, pp. 127–146, 2015.
DOI: 10.1007/978-3-662-46706-0_7

related-key model, could be used to break the full versions of AES-192 and AES-256 based on exploiting the key schedule [1–3], but such attacks require a very powerful assumption that the adversary can ask to modify the unknown key used in the encryption. So related-key attacks are widely used as an important method to estimate the security of a block cipher, but are not regarded as a real threat to the application of a cipher in practice. For the attacks under single-key model, up to now, the best attacks except the biclique method could reach to 7-round for AES-128, 8-round for AES-192 and 9-round for AES-256. The biclique method was used to attack the full AES with a marginal complexity over exhaustive search by Bogdanov, Khovratovich and Rechberger at ASIACRYPT 2012 [4].

In this paper, we focus on the meet-in-the-middle attack (MITM) in the single-key model, which was deeply researched in recent years, and now may be the most efficient attack on all versions of AES [9]. The meet-in-the-middle attack was first proposed by Diffie and Hellman to attack DES [10]. For AES cipher, this method was introduced by Demirci and Selçuk at FSE 2008 [6] to improve the collision attack proposed by Gilbert and Minier [13]. They constructed a 4-round distinguisher to attack the 7-round and 8-round AES. The attack needs a small data complexity of 2^{34}, but requires a large memory of $2^{25\times 8}$ to set up precomputation table determined by 25 intermediated variable bytes. The number of parameters could be reduced to 24 bytes, if one considers to store the differentials instead of values in precomputation table. Combined with data/time/memory tradeoff, this attack was applied to analyse 7-round AES-192 and 8-round AES-256.

At ASIACRYPT 2010, Dunkelman, Keller and Shamir [11] exploited the differential enumeration and multiset ideas for MITM attacks to reduce the high memory complexity in the precomputation phase. Indeed, they showed that if a pair conforms to a truncated differential characteristic, the number of the desired 24 intermediated variable bytes will be reduced to 16. Since this attack reduces the memory complexity with the expense of increasing the data complexity to make a pair conform to the differential characteristic, it may be seen as a new data/time/memory tradeoff. Furthermore, Derbez, Fouque and Jean presented a significant improvement of Dunkelman *et al.*'s attack at EUROCRYPT 2013 [9]. Using the rebound-like idea, they showed that many values in precomputation table are not reached at all under the constraint of the truncated differential. Actually, the size of precomputation table is determined by 10-byte parameters. Based on the 4-round distinguisher, they gave the most efficient attacks on 7-round AES-128 and 8-round AES-192/256. Besides, they introduced a 5-round distinguisher to analyse 9-round AES-256.

Our Contribution. Although the MITM attack has been improved and perfected a lot by Dunkelman et al. and Derbez et al. [9,11], we notice that some key relations could be further exploited to improve the results of previous attacks.

In this paper, based on the properties of the key schedule, we construct a stronger 5-round distinguisher, which supports us to give more efficient attacks on 9-round AES-192/256.

In [9], Derbez *et al.* proposed a 5-round distinguisher of AES with the memory complexity of 2^{208}, so it seems infeasible for the attack on AES-192. However, by studying the key relationship of the distinguisher, we find that many values in the previous precomputation table could be filtered, and the size of the table is only 2^{192}. Especially, if the attack starts from the third round, the size of precomputation table would be further reduced by a factor of 2^8. Subsequently, combing with the classic data/time/memory trade-off, we present an attack on 9-round AES-192 with about 2^{121} chosen plaintexts, $2^{187.5}$ encryptions and 2^{185} 128-bit storages. For the attack on AES-192 starting from the third round, the data, time and memory complexities are reduced to 2^{117}, $2^{183.5}$ and 2^{181}, respectively. Since the new technique takes advantage of the subkeys involved in distinguisher as the filter conditions to reduce the size of precomputation table, we call it *key-dependent sieve*.

In the second part of the paper, we show that the whole attack is able to be sorted into a series of weak-key attacks by using of the shared key information in the online and offline phases, where every weak-key attack takes an independent sub-table included in the precomputation table. That supports us to reduce the memory complexity of the attack without any cost of the data and time complexities, since we can perform the attack in streaming mode by working on each weak attack independently and releasing the memories afterwards. For 9-round attacks on AES-192 and AES-256, the memory complexities are reduced by 2^8 and 2^{32} times, respectively. Although the data and time complexities are not reduced in such case, it is meaningful for us to save the memory requirement of the attack, specially, for the attack that the memory complexity takes over the dominant term.

Table 1 summaries our results along with some major previous results of AES-192 and AES-256 under single-key model. The rest of this paper is organized as follows. Section 2 gives a brief description of AES and some related works. In Sect. 3, we propose the single-key attacks on 9-round AES-192. Section 4 presents an interesting method to reduce the memory complexity of the attack. Finally, we conclude the paper in Sect. 5.

2 Preliminaries

This section first gives a brief description of AES and denotes some notations and definitions used throughout the paper. Finally, we introduce some related works of AES with meet-in-the-middle attack.

Table 1. Summary of the Attacks on AES-192/256 in the Single-key Model

Cipher	Rounds	Attack Type	Data	Time	Memory	Source
AES-192	8	MITM	2^{113}	2^{172}	2^{129}	[11]
	8	MITM	2^{113}	2^{172}	2^{82}	[9]
	8	MITM	2^{113}	2^{140}	2^{130}	[8]
	9	Bicliques	2^{80}	$2^{188.8}$	2^{8}	[4]
	9	MITM	2^{121}	$2^{187.5}$	2^{185}	Sect. 3.2
	9	MITM	2^{121}	$2^{186.5}$	$2^{177.5}$	Sect. 4.1
	9 (3-11)	MITM	2^{117}	$2^{183.5}$	2^{181}	Sect. 3.3
	9 (3-11)	MITM	2^{117}	$2^{182.5}$	$2^{165.5}$	Sect. 4.1
	Full	Bicliques	2^{80}	$2^{189.4}$	2^{8}	[4]
AES-256	8	MITM	2^{113}	2^{196}	2^{129}	[11]
	8	MITM	2^{113}	2^{196}	2^{82}	[9]
	8	MITM	$2^{102.83}$	2^{156}	$2^{140.17}$	[8]
	9	Bicliques	2^{120}	$2^{251.9}$	2^{8}	[4]
	9	MITM	2^{120}	2^{203}	2^{203}	[9]
	9	MITM	2^{121}	$2^{203.5}$	$2^{169.9}$	Sect. 4.2
	Full	Bicliques	2^{40}	$2^{254.4}$	2^{8}	[4]

2.1 A Brief Description of AES

The advanced encryption standard (AES) [17] is a 128-bit block cipher, which uses variable key sizes and the number of rounds (N_r) depend on the key sizes, i.e., 10 rounds for 128-bit key size, 12 rounds for 192-bit key size and 14 rounds for 256-bit key size. The 128-bit internal state is treated as a byte matrix of size 4×4, and each byte represents a value in $GF(2^8)$. The round function is composed of four basic operations:

- SubBytes (SB) is a nonlinear byte-wise substitution that applies an 8 by 8 S-box to every byte.
- ShiftRows (SR) is a linear operation that rotates on the left of the $i-$th row by i bytes.
- MixColumns (MC) is a matrix multiplication over a finite field applied to each column.
- AddRoundKey (ARK) is an exclusive-or operation with the round subkey.

Before the first round an additional whitening ARK operation is performed, and in the last round the MC operation is omitted.

The key schedule of AES expands the master key to $N_r + 1$ 128-bit subkeys. The subkey array is denoted by $w[0, \cdots, 4 \times N_r + 3]$, where each word $w[\cdot]$ consists

32 bits. Let the number of words for master key is denoted by N_k, e.g., $N_k = 6$ for AES-192. Then the first N_k words of $w[\cdot]$ are filled with the master key. The remaining words are defined as follows:

– For $i = N_k$ to $4 \times N_r + 3$ do the following:
  - If $i \equiv 0 \bmod N_k$, then $w[i] = w[i - N_k] \oplus SB(w[i-1] \lll 8) \oplus Rcon[i/N_k]$,
  - else if $N_k = 8$ and $i \equiv 4 \bmod 8$, then $w[i] = w[i - N_k] \oplus SB(w[i-1])$,
  - Otherwise $w[i] = w[i - N_k] \oplus w[i-1]$.

where $\lll$ represents left rotation, $\oplus$ denotes the bit-wise exclusive OR (XOR) and $Rcon[\cdot]$ is an array of fixed constants. For more details about AES, we refer to [17].

2.2 Notations and Definitions

In this paper, the plaintext and ciphertext are denoted by P and C. The symbols X_i, Y_i, Z_i and W_i denote the internal states before SB, SR, MC and ARK operations in the round-i ($0 \le i \le N_r - 1$), respectively. The subkey of round i is denoted by k_i, the first key (whitening) is denoted by k_{-1}. We use the symbol u_i to represent the equivalent key with $u_i = MC^{-1}(k_i)$.

A 128-bit internal state A is represented as a 4×4 byte matrix. The symbol $A[i]$ is used to express a byte of A, where i is the ordering of bytes ($i = 0, \cdots, 15$). The symbol $A[i, \cdots, j]$ represents the $i-$th byte to the $j-$th byte of A.

As in previous works, the $\delta-$set utilized in this paper is defined as follows.

Definition 1 ($\delta-$set, [5]). *The $\delta-$set is a set of 2^8 AES states that one byte traverses all values (the active byte) and the other bytes are constants (the inactive bytes).*

We denote the $\delta-$set as $(X^0, \cdots, X^{255})$. Usually, we consider to encrypt a $\delta-$set by a function E_K and select the $i-$th byte of the ciphertexts as the output value ($0 \le i \le 15$), then the corresponding 2^8 output bytes form a 2048-bit vector $E_K(X^0)[i] \| \cdots \| E_K(X^{255})[i]$ with ordered arrangement, where $\|$ represents the bit string concatenation. Another important concept is the multiset, which was introduced by Dunkelman *et al.* in [11].

Definition 2 (Multiset of bytes [11]). *A multiset generalizes the set concept by allowing elements to appear more than once. Here, a multiset of 256 bytes can take as many as $\binom{511}{255} \approx 2^{506.7}$ different values.*

Property 1. (Differential property of $S-$box [11]) Given the input and output differences of the SubBytes operation, there exists a pair of actual values on average to satisfy these differences. This property is also applied to the inversion of SubBytes operation.

The time complexity of the attack in this paper is measured with the unit of an equivalent encryption operation of the 9-round AES. The memory complexity is measured with the unit of a block size (128-bit). It is emphasized that we count all operations performed during the attack, in particular, the time and memory requirements in precomputation phase.

2.3 Related Works

In this section, we recall the previously MITM attacks on AES. Firstly, we introduce the Demirci and Selçuk attack. Then two improvements given by Dunkelman *et al.* and Derbez *et al.* are shown briefly.

Demirci and Selçuk Attack. Combining the MITM method, Demirci and Selçuk improved the collision attack [13] on AES. They treated the cipher E as $E_K = E^2_{K_2} \circ E^m \circ E^1_{K_1}$, and built a distinguisher in E^m based on the following 4-round AES property.

Property 2. Consider the encryption of a $\delta-$set through four full AES rounds. For each of the 16 bytes of the state, the ordered sequence of 256 values of that byte in the corresponding ciphertexts is fully determined by just 25 byte parameters. Consequently, for any fixed byte position, there are at most 2^{200} possible sequences when we consider all the possible choices of keys and δ-sets (out of 2^{2048} theoretically value).

These parameters are composed of some intermediate states of a message of the $\delta-$set, which are determined by the positions of the active byte and the corresponding output byte. If the active byte is located in $X_1[0]$ and the output byte is selected in $X_5[0]$, then, as described in Fig. 1, the 25-byte parameter is $X_2[0,1,2,3]||X_3[0,\cdots,15]||X_4[0,5,10,15]||X_5[0]$. When the values in the output sequence were substituted by the corresponding differences, the value $X_5[0]$ could be omitted and the number of parameter reduces to 24.

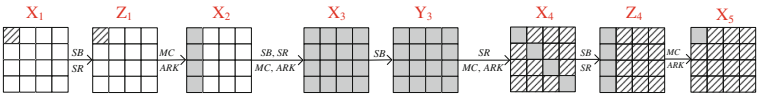


Fig. 1. The 4-round AES distinguisher used in [13], the gray cells represent 25-byte parameter

On the basic of Property 2, they gave the MITM attacks on 7-round and 8-round AES-256, respectively. For the attack on 7-round AES, one round and 2 rounds are extended in the top and bottom of the 4-round E_m, respectively. The attack is divided into two phases, precomputation phase and online phase.

1. Precomputation phase: compute all 2^{200} values of the sequence given in Property 2, and store them in a hash table.
2. Online phase:
   (a) Encrypt a structure of 2^{32} chosen plaintexts such that the main diagonal can take all the 2^{32} possible values and the remaining bytes are constant.
   (b) Guess values of the related subkeys in E_1, and construct a δ-set. Then partially decrypt to get the corresponding 256 plaintexts.
   (c) Obtain the corresponding plaintext-ciphertext pairs from the collection data. Then guess the related subkeys in E_2, and partially decrypt the ciphertexts to get the corresponding 256-byte value of the output sequence of E_m.
   (d) If a sequence value lies in the precomputation table, the guessed related subkeys in E_1 and E_2 may be right key.
   (e) Exhaustive search the remaining subkeys to obtain the right key.

The data complexity of the attack in the online phase is only about 2^{32}, but the memory and time complexities for precomputation phase are too large. So the data/time/memory tradeoff was used in their work to reduce the complexities in the precomputation phase, which makes the attack to apply to 7-round AES-192. However, the time complexity in the online phase is very large, then it is impossible to rebalance for 8-round AES-192 in their work.

Dunkelman *et al.*'s Attack. At ASIACRYPT 2010, Dunkelman, Keller and Shamir [11] proposed some interesting techniques to improve the Demirci and Selçuk attack. Firstly, they proposed to use the multiset to replace the ordered sequence for the output byte, since it is enough to distinguish a proper value from the random sequences. Secondly, a novel idea named differential enumeration technique was introduced to reduce the memory complexity in the precomputation phase, at the expense of increasing the data complexity. The main idea of this technique is to fix some values of intermediate parameters by using of the truncated differential. They showed that if one considers to encrypt a $\delta-$set after four full-rounds of AES, in the case of that a message of the $\delta-$set belongs to a pair conforming the particular 4-round truncated differential characteristic described as in Fig. 2, then the corresponding output value of multiset only takes about 2^{128} possible values. Note that the gray cells in Fig. 2 are active bytes,

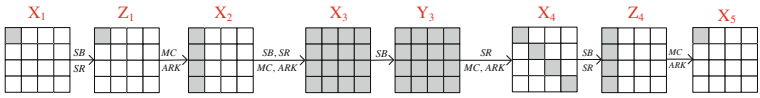


Fig. 2. The truncated differential characteristic of 4-round AES used in [11]

while the white cells are inactive. Indeed, it is obvious that when a pair conforms the truncated differential as in Fig. 2, the state X_3 only takes about 2^{64} different values.

Hence, there are about 2^{128} possible values stored in the precomputation phase. In the online phase, more plaintexts should be chosen to make sure there exists a pair in content with the truncated differential. Thus, the data complexity is 2^{113} chosen plaintexts. This attack procedure is similar to Demirci and Selçuk attack, but a step to look for a pair satisfying the truncated differential is added, and the $\delta-$set is constructed only for such pair. Finally, they gave attacks on the 7-round AES-128 and 8-round AES-192/256. Actually, the attack can be regarded as a special data/time/memory tradeoff.

Derbez *et al.*'s Attack. More recently, Derbez, Fouque and Jean presented a significant improvement to Dunkelman *et al.*'s attack at EUROCRYPT 2013 [9]. Combining with the rebound-like view of the cipher, they showed that the number of possible values of precomputation table in Dunkelman *et al.*'s attack could be further reduced. In their work, if a message of $\delta-$set belongs to a pair conforming the 4-round truncated differential characteristic outlined in Fig. 2, the value of multiset is only determined by 10-byte variables of intermediate state $\Delta Z_1[0]\|X_2[0,1,2,3]\|\Delta X_5[0]\|Z_4[0,1,2,3]$. In other words, there are only about 2^{80} possible sequences of multiset to be stored in precomputation table. Then they improved the attacks on 7-round AES-128 and 8-round AES-192/256. Further, they proposed to use the truncated differential characteristic which contains two active byte in X_1, to balance the time and memory complexities of the attack.

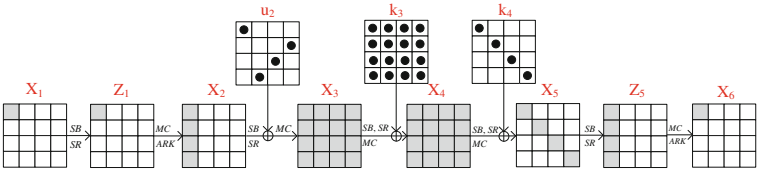


Fig. 3. The truncated differential characteristic of 5-round AES used in [9]

Moreover, they proposed to use a 5-round distinguisher to attack 9-round AES-256. The corresponding truncated differential characteristic is outlined in Fig. 3, where the value of multiset is determined by 26-byte parameters

$$\Delta Z_1[0]\|X_2[0,1,2,3]\|X_3[0,\cdots,15]\|\Delta X_6[0]\|Z_5[0,1,2,3].$$

It is interesting that a value of the 192-bit subkey is known for each value of multiset in precomputation table. For the above differential characteristic, the 192-bit deduced subkey is $u_2[0,7,10,13]\|k_3[0,\cdots,15]\|k_4[0,5,10,15]$.

3 The Improved Attacks on 9-Round AES-192

In this section, we apply the improved 5-round distinguisher to attack 9-round AES-192. It turns out that, the size of hash table in precomputation phase is able to be reduced to 2^{192} from 2^{208}. More importantly, if the 5-round distinguisher starts from the fourth round, the size of hash table would be reduced to 2^{184}.

3.1 Key-Dependent Sieve and 5-Round Distinguisher of AES-192

It is obvious that the memory complexity and time complexity in the precomputation phase are the bottlenecks of the MITM attack on AES. Nevertheless, we find that some key relations are valuable to reduce the complexity in the precomputation phase, where the same key information is deduced by two approaches, the parameters and the key schedule. Then both of them may be not equal since two approaches are absolutely independent. This makes us to filter the redundant values of precomputation table and makes it possible to attack on 9-round AES-192.

We review the 5-round distinguisher proposed by Derbez et al. in [9]. Since the size of lookup table is determined by 26 parameters, it seems infeasible to attack 9-round AES-192. However, by the key schedule of AES-192, it is obviously that the knowledge of k_3 allows to deduce the column 0 and 1 of k_2. That means the value of the equivalent subkey $u_2[0,7]$ is computed by k_3. However, $u_2[0,7]$ is already deduced by the 26-byte parameter for each value of the multiset seen Fig. 3. Thus there exists a contradiction between $u_2[0,7]$ and k_3. In other words, if a possible value of the multiset is correct, the value of $u_2[0,7]$ must be equal to the equivalent value deduced from k_3, which happens with a probability of 2^{-16}. Therefore, the size of look up table is about $\frac{2^{208}}{2^{16}}$ for 5-round distinguisher of AES-192. Because our technique filtering the wrong states is based on the key relationship, so we call it *key-dependent sieve*. Combined with data/time/memory tradeoff, we apply the 5-round distinguisher to attack 9-round AES-192. However, the time complexity of precomputation phase is too large for all possible values of lookup table, that is about $2^{192} \times 2^8$ computations. So we introduce an improved 5-round distinguisher of AES-192 in the sequel.

The 5-round differential characteristic utilized in our attack is described in Fig. 4, where the position of the active byte is defined in $W_0[12]$, and the output value of sequence is located in $Y_6[6]$.

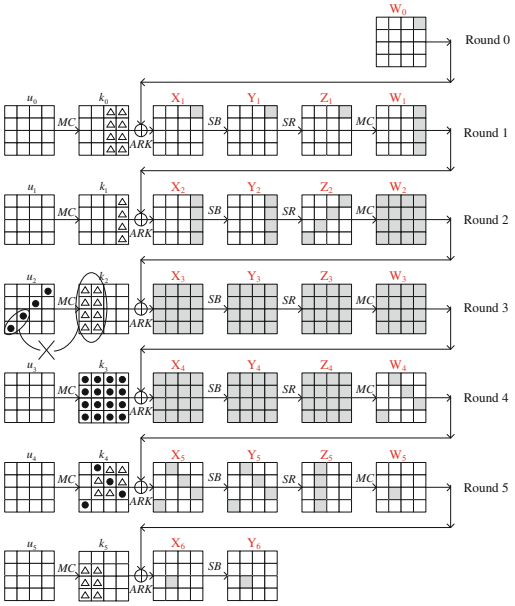


Fig. 4. The truncated differential characteristic of the 5-round AES-192

Proposition 1. *Consider the encryption of the first* 2^5 *values* $(W_0^0, \cdots, W_0^{31})$ *of the* $\delta-$*set through 5-round AES-192, in the case of that a message pair* (W_0, W_0') *of the* $\delta-$*set conforms to the truncated differential characteristic outlined in Fig. 4, then the corresponding 256-bit ordered sequence* $Y_6^0[6]\| \cdots \|Y_6^{31}[6]$ *only takes about* 2^{192} *values (out of* 2^{256} *theoretically value).*

Proof. We give a brief proof of this proposition. For the encryptions of the first 2^5 values of the $\delta-$set, it is easier to conclude that the output sequence $Y_6^0\| \cdots \|Y_6^{31}$ is determined by the 43-byte variable

$$W_0[12]\|X_1[12]\|X_2[12, \cdots, 15]\|X_3[0, \cdots, 15]\|k_3[0, \cdots, 15]\|k_4[3, 4, 9, 14]\|k_5[6].$$

However, if a pair conforms the truncated differential characteristic outlined in Fig. 4, the 40-byte value

$$X_2[12, \cdots, 15]\|X_3[0, \cdots, 15]\|k_3[0, \cdots, 15]\|k_4[3, 4, 9, 14]$$

is determined by the 26-byte variable

$$\Delta Z_1[12]\|X_2[12, 13, 14, 15]\|X_3[0, \cdots, 15]\|Z_5[4, 5, 6, 7]\|\Delta X_6[6].$$

Here, the knowledge of $\Delta Z_1[12]\|X_2[12, 13, 14, 15]\|X_3[0, \cdots, 15]$ supports to compute the intermediate difference ΔX_4. For the backward direction, the knowledge

of $Z_5[4,5,6,7] \| \Delta X_6[6]$ supports to compute ΔY_4. Then according to Property 1, we get one value of intermediate state $X_4 \| Y_4$ on average for the fixed difference $\Delta X_4 \| \Delta Y_4$. Apparently, the 192-bit value of subkey $u_2[3,6,9,12] \| k_3[0,\cdots,15] \| k_4[3,4,9,14]$ is also deduced for every 26-byte variable. According to key schedule of AES-192, we can compute the other value $u_2[3,6]$ form k_3. By the key-dependent sieve, there are 2^{192} possible values for 26-byte parameters.

By the key schedule, the value of subkey denoted by triangles in Fig. 4 are deduced by the value of the 192-bit subkey denoted by blackspot in Fig. 4. Here, we only focus on the subkey $k_0[12] \| k_1[12,13,14,15] \| k_5[6]$. For any 26-byte parameter, $k_1[12,13,14,15]$ is used to compute $X_1[12]$, and $k_0[12]$ is used to compute $W_0[12]$. Besides, the values of $X_6[6]$ and $Y_6[6]$ are computed by the value of $k_5[6]$. Therefore, the whole 43-byte variable is deduced by 26-byte parameter. So there are 2^{192} possible values of 43-byte variables, which means the number of possible sequences $Y_6^0[6] \| \cdots \| Y_6^{31}[6]$ is approximately 2^{192}. □

Note that in Derbez *et al.*'s attack, the multiset technique was used to omit the influence of 16-bit subkey belongs to k_0 and k_5. If the truncated differential characteristic is selected as in Fig. 4, the 16-bit subkey would be $k_0[12] \| k_5[6]$. Here, we prove that such 16-bit information could be deduced by 192-bit subkey $u_2[3,6,9,12] \| k_3[0,\cdots,15] \| k_4[3,4,9,14]$. Then we extend the distinguisher to the W_0 in the forward, and Y_6 in the backward. Thus, we use an ordered sequence instead of the multiset. For the output value of encrypting the $\delta-$set, the ordered sequence includes 2048-bit information, while a multiset only contains about 507-bit information. Indeed, only the first 32-byte value of the $\delta-$set is enough to distinguish a proper sequence with the probability of $\frac{2^{192}}{2^{256}} = 2^{-64}$, and the data and time complexities are reduced by 2^3 times in the attack.

3.2 The Key Recovery Attack on 9-Round AES-192

We propose an attack on 9-round AES-192 by adding one round on the top and three rounds on the bottom of the 5-round distinguisher (Fig. 5). The attack is composed of two phases: precomputation phase and online phase. In the precomputation phase, we get all possible 256-bit sequences described as Proposition 1 by using the rebound-like technique, which is described as follows.

Precomputation Phase. For each 128-bit k_3, do the following steps.

1. Compute the subkey $u_2[3,6] \| k_1[12,13,14,15] \| k_0[12]$ by the key schedule.
2. Traverse $\Delta X_6[2] \| Z_5[4,5,6,7]$ to compute $\Delta X_5[3,4,9,14] \| X_5[3,4,9,14]$, and store $X_5[3,4,9,14]$ in a table T_0 indexed by $\Delta X_5[3,4,9,14]$. There are about 2^8 values of $X_5[3,4,9,14]$ for each index.
3. For all 64-bit value of difference $\Delta Y_2[12,\cdots,15] \| \Delta X_5[3,4,9,14]$, we apply the super-sbox technique [14] to connect the differences $\Delta Y_2[12,\cdots,15]$ and

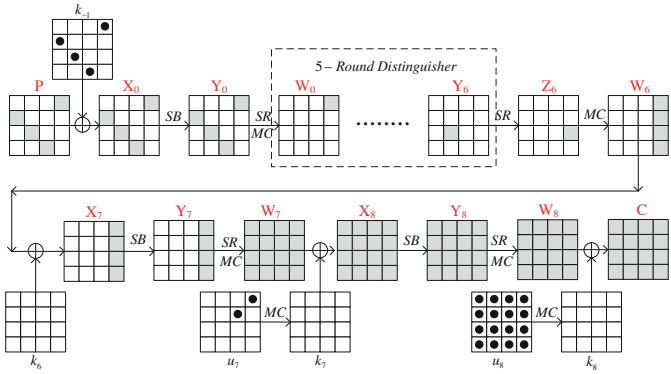


Fig. 5. The attack on 9-round AES-192

$\Delta X_5[3,4,9,14]$, and deduce the intermediate value $X_3 \| W_4$. Then $Y_2[14,15]$ is obtained by X_3 and $u_2[3,6]$. Store these values with the index of 48-bit value $\Delta Y_2[12,\cdots,15] \| Y_2[14,15]$ in a table T_1. There are about 2^{16} values of $\Delta X_5[3,4,9,14] \| X_3 \| W_4[3,4,9,14]$ corresponding to the index $\Delta Y_2[12,\cdots,15] \| Y_2[14,15]$.

4. For each $\Delta Z_1[12] \| X_2[12,13,14,15]$, execute the following substeps.
   (a) Compute the state $X_1[12] \| W_0[12] \| \Delta Y_2[12,13,14,15] \| Y_2[12,13,14,15]$.
   (b) Then look up the table T_1 to get about 2^{16} values $\Delta X_5[3,4,9,14] \| X_3 \| W_4[3,4,9,14]$ by the values of $\Delta Y_2[12,13,14,15] \| Y_2[14,15]$. And get the equivalent subkey $u_2[9,12]$.
   (c) For each value of $\Delta X_5[3,4,9,14] \| X_3 \| W_4[3,4,9,14]$, we get 2^8 values of $X_5[3,4,9,14]$ by accessing the table T_0. Then compute $k_4[3,4,9,14]$ and $k_5[6]$. Here we get 43-byte variable $W_0[12] \| X_1[12] \| X_2[12,\cdots,15] \| X_3[0,\cdots,15] \| k_3[0,\cdots,15] \| k_4[3,4,9,14] \| k_5[6]$.
   (d) Construct the $\delta-$set, and compute the corresponding sequence $Y_6^0[6] \| \cdots \| Y_6^{31}[6]$, and store them in a hash table $\mathcal{H}$.

Online Phase. In the online phase, we need to find at least a pair satisfying the truncated differential characteristic, then construct the $\delta-$set, and obtain the first 32 bytes output value of it. Finally, detect whether it belongs to the precomputation table. The attack procedure is described as follows.

1. Encrypt 2^{81} structures of 2^{32} plaintexts, such that $P[1,6,11,12]$ takes all 32-bit values and other bytes are constants. There are 2^{144} pairs totally.
2. For each pair, do the following substeps.
   (a) Guess the difference value $\Delta Y_7[12,13,14,15]$, and compute the subkey u_8 (or k_8, if the last MC operation is omitted). Then deduce $u_7[3,6]$.
   (b) Compute the difference $\Delta X_7[14,15]$, delete the wrong guesses which don't lead to $\Delta Z_6[12,13,15] = 0$, there are about 2^{24} guesses remaining after this step.

(c) For each remaining guess, deduce subkey $u_7[9, 12]$.
(d) Guess the difference $\Delta W_0[12]$, and compute the subkey $k_{-1}[1, 6, 11, 12]$.

3. For each deduced subkey, select one message of the pair and get the value $W_0[12]$. Then change the value of $W_0[12]$ to be $(0, \cdots, 31)$ and compute plaintexts $(P^0, \cdots, P^{31})$. Query their corresponding ciphertexts, and get the corresponding sequence $Y_6^0[6] \| \cdots \| Y_6^{31}[6]$ by partial decryption. Note that the equivalent subkey $u_6[14]$ is deduced by u_8 in such case.
4. Find the right subkeys by verifying whether the sequence lies in table $\mathcal{H}$. There are about $2^{176} \times 2^{-64}$ subkeys remaining in the end. Then exhaustively search for $u_7[8, 10, 11, 13, 14, 15]$ to find the real key, which needs about 2^{160} encryptions.

Complexity Analysis. In the precomputation phase, each value of the $\delta-$set needs about 2-round AES computations. Then the time complexity of the precomputation phase is about $2^{192} \times 2^5 \times 2^{-2.2} = 2^{194.8}$ 9-round AES encryptions, which also needs about 2^{193} 128-bit words of memory to store all possible sequences. The time complexity of the online phase is dominated by step 3, where the computation of each value in the $\delta-$set needs about 1.5-round AES encryptions. So the time complexity of the online phase is equivalent to $2^{144} \times 2^{32} \times 2^5 \times 2^{-2.6} = 2^{178.4}$ 9-round encryptions. The attack needs about 2^{113} chosen plaintexts.

Data/Time/Memory Tradeoff. With data/time/memory tradeoff, the adversary only needs to precompute a fraction 2^{-8} of possible sequences, then the time complexity is about $2^{184} \times 2^5 \times 2^{-2.2} = 2^{186.8}$ 9-round computations. The memory complexity reduces to $2^{193 \times 2^{-8}} = 2^{185}$. But in the online phase, the adversary will repeat the attack 2^8 times to offset the probability of the failure, that means the attack becomes probabilistic. So the data complexity increases to 2^{121} chosen plaintexts, and the time complexity increase to $2^{178.4} \times 2^8 = 2^{186.4}$ 9-round encryptions. In total, including the precomputation phase, time complexity is approximately $2^{187.5}$.

3.3 The Attack on 9-round AES-192 from the Third Round

We observe that the memory complexity will be reduced again when the 5-round distinguisher is mounted to rounds 4-9 in order to attack the reduced-round AES-192 from rounds 3 to 11. The same truncated differential characteristic outlined in Fig. 4 is used in the attack, except to move all intermediate states after two rounds. Then the 5-round distinguisher is from the state W_2 to the state Y_8. Similar to the Proposition 1, we consider to encrypt a $\delta-$set $(W_2^0, \cdots, W_2^{255})$ after 5-round AES-192. If a message pair of the δ-set satisfies the expected truncated differential characteristic, then there are about 2^{192} possible values of sequence $Y_8^0 \| \cdots \| Y_8^{255}$. Corresponding, the 176-bit subkey

$u_4[9,12]\|k_5[0,\cdots,15]\|k_6[3,4,9,14]$ is deduced for each sequence. Here, $u_4[3,6]$ are omitted, which can be deduced from k_5.

However, as described in Fig. 6, we find that $k_6[4]$ are deduced by the values of $k_5[1]$ and $k_6[9]$, which may be contradicted with the known value $k_6[4]$ for each sequence, where the right half in Fig. 6 is the original key schedule of AES-192, and the left half is its equivalent value $v_i = MC^{-1}(w_i)$. Note that there exist the following relations for the equivalent key v_i if $i \geq 6$.

- If $i \equiv 0 \bmod 6$, then $v[i] = v[i-6] \oplus MC^{-1}(SB(MC(w[i-1]) \lll 8)) \oplus MC^{-1}(Rcon[i/6])$,
- Else $v[i] = v[i-6] \oplus v[i-1]$.

So there are only about 2^{184} possible sequences remaining after eliminating the incorrect states in such case.

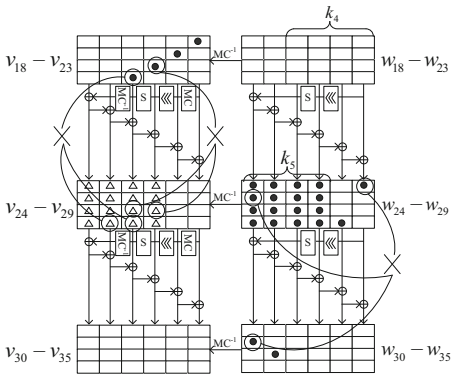


Fig. 6. The key relationship of the attack on AES-192 starting from the third round

The attack procedure is similar to the attack of subsect. 3.2. The precomputation table is constructed as follows. For each 128-bit difference ΔX_6, do the following steps.

1. Traverse the 40-bit difference $\Delta Y_3[12]\|\Delta Y_4[12,13,14,15]$ to deduce the states $X_4[12,13,14,15]\|X_5\|W_5\|u_4[3,6,9,12]$. Store these states in a hash table T_1 with the 24-bit index

$$(Z_5[7] \oplus Z_5[11] \oplus u_4[3])\|(Z_5[10] \oplus Z_5[14] \oplus u_4[6])\|W_5[1].$$

2. Traverse the 40-bit difference $\Delta X_8[6]\|\Delta X_7[3,4,9,14]$ to deduce the intermediate states $X_7[3,4,9,14]\|X_6\|W_6\|k_6[3,4,9,14]$. Compute the 24-bit value $(MC(X_6)[7] \oplus MC(X_6)[11])\|(MC(X_6)[10] \oplus MC(X_6)[14])\|(X_6[1] \oplus k_6[9] \oplus S(k_6[4]) \oplus Rcon[4][1])$. Then access the hash table T_1 with 24-bit value to get

the states $X_4[12,13,14,15]\|X_5\|W_5\|u_4[3,6,9,12]$. There are about 2^{16} states in table T_1 for each index. In total, we collect 2^{56} correct states which satisfy

$$\begin{cases} u_4[3] = u_5[7] \oplus u_5[11], \\ u_4[6] = u_5[10] \oplus u_5[14], \\ k_5[1] = k_6[9] \oplus S(k_6[4]) \oplus Rcon[4][1]. \end{cases}$$

3. Construct the $\delta-$set, compute the corresponding sequence $Y_8^0[6]\| \cdots \|Y_8^{31}[6]$, and store them in a hash table.

Complexity Analysis. The time complexity of this phase is equivalent to $2^{184} \times 2^5 \times 2^{-2.2} = 2^{186.8}$ 9-round encryptions, the memory complexity is about $2^{184} \times 2 = 2^{185}$ 128-bit storages. The online phase is exactly the same procedure of the attack in Sect. 3.2, which needs about 2^{113} chosen plaintexts and $2^{178.4}$ 9-round encryptions.

Data/Time/Memory Tradeoff. We precompute a fraction 2^{-4} possible sequences, then the time complexity of the attack in the online phase is about $2^{178.4} \times 2^4 = 2^{182.4}$ 9-round encryptions. The memory complexity decreases to $2^{185} \times 2^{-4} = 2^{181}$ 128-bit, and the data complexity increases to 2^{117} chosen plaintexts. In additional, we need $2^{182.8}$ 9-round encryptions to compute all possible sequences in precomputation phase, then the time complexity including the precomputation is about $2^{183.5}$ 9-round encryptions.

4 Reducing the Memory Complexity with Weak-Key Attacks

It is known that there exists a subkey k' for every sequence in precomputation table. In other view, such a value k' could be regarded as an extensional characteristic of the sequence. In Sect. 3, we use the property of self-contradictory phenomenon of the k' to reduce the number of possible sequences. In this section, by investigating more properties of this information, we show that the memory complexity could be further reduced without increasing the data and time complexities.

We denote the subkey guessed in the online phase as $\widehat{k}$. It is obvious there exist some linear relations in k' and $\widehat{k}$. Assuming m bits value $\widetilde{k} \subset (k' \cap \widehat{k})$, we first split the precomputation table with the index of $\widetilde{k}$ into 2^m sub-tables. Thus, in the online phase, for each guessed subkey $\widehat{k}$ and its sequence, instead of checking all precomputation table, we only need to detect a sub-table in the line with the index value $\widetilde{k}$. Furthermore, we also split the sequences computed in the online phase to 2^m subsets with the same index $\widetilde{k}$. Then for all sequences belong to a subset, we only need to detect a sub-table, and it is meaningless to check whether they belong to other sub-tables.

Thus, the whole attack could be sorted into 2^m sub-attacks. Each sub-attack contains a sub-table of precomputation, and all of these attacks are independent each other. Since each sub-attack is worked under a fixed value of m-bit key information, which is also seen as a weak-key attack. Assuming $\mathcal{C}$ is the time (or memory) complexity of the whole attack, then it is evident to see that the time (or memory) complexity for every weak-key attack is $\mathcal{C}/2^m$, but the data and time complexities of the whole attack don't change at all. Nevertheless, if all weak-key attacks are worked in the streaming model, the memory complexity could be reduced by 2^m times since the storages could be reused for each weak-key attack. However, the whole precomputation table could not be reused in such case.

4.1 Reducing the Memory Complexity for Attacks on 9-Round AES-192

We take into account the application of this method to the attack on 9-round AES-192, where k' is 176-bit subkey $u_2[9,12]\|k_3[0,\cdots,15]\|k_4[3,4,9,14]$, and $\widehat{k}$ is the 176-bit subkey $u_8[0,\cdots,15]\|u_7[9,12]\|k_{-1}[1,6,11,12]$. It is not difficult to see that the set $k' \cup \widehat{k}$ contains the whole information of 192-bit master key, and the set $k' \cap \widehat{k}$ contains 160-bit information. We use 8-bit information $k_{-1}[6]$ as the index to split the attack to 2^8 weak-key attacks, where

$$k_{-1}[6] = SB(k_3[1] \oplus k_3[5]) \oplus k_3[10] \oplus k_3[14] \oplus Rcon[2][2].$$

The attack procedure is very simple. We first split 2^{128} possible value of k_3 to 2^8 subsets with the index value $k_{-1}[6]$. Then for each weak-key attack with a fixed value $k_{-1}[6]$, do as follows.

1. For the corresponding subset of k_3, do as described in Sect. 3.2 to construct the sub-table $\mathcal{H}'$.
2. For 2^{113} plaintexts, guess 24-bit subkey $k_{-1}[1,11,12]$, and collect all pairs satisfying $\Delta W_0[13,14,15] = 0$.
3. For every pair guess the difference $\Delta Y_6[6]$.
4. Guess $\Delta Y_7[14,15]$ to deduce the subkey $u_7[3,6]\|u_8[0,1,4,7,10,11,13,14]$, and only keep the value which satisfies $u_7[3] = u_8[7] \oplus u_8[11]$ and $u_7[6] = u_8[10] \oplus u_8[14]$. There is one value of $\Delta Y_7[14,15]$ along with its subkey remain on average for every $\Delta Y_6[6]$.
5. Guess $\Delta Y_7[12,13]$ to deduce the subkey $u_7[9,12]\|u_8[2,3,5,6,8,9,12,15]$.
6. Construct the $\delta-$set, compute the corresponding sequence, and check whether they belongs to $\mathcal{H}'$.

Complexity Analysis. For each weak-key attack, the time complexity of step 1 is about $2^{184} \times 2^5 \times 2^{-2.2} = 2^{186.8}$, the memory complexity is about 2^{185} 128-bit. For step 2 to step 6, the time complexity is about $2^{168} \times 2^5 \times 2^{-2.6} = 2^{170.4}$. By data/time/memory tradeoff, we precompute a fraction 2^{-8} possible sequences, the time complexity is about $2^{179.5}$ 9-round encryptions, the memory complexity could be reduced to 2^{177} 128-bit spaces. In total, the complexity is still $2^{179.5} \times 2^8$ 9-round encryptions, approximately.

Reducing the Time Complexity by a Half. By investigating the information of $k' \cap \widehat{k}$, we learn that the 64-bit information is identified out of 160-bit information, that is $k_{-1}[6] \| k_{-1}[11] \| u_1[12] \| u_3[1] \| k_4[4,5,6] \| k_5[11]$. Then each sequence of the distinguisher is represented by the first 16 bytes of the $\delta-$set along with above 64-bit information and 176-bit k'. Hence, for each sequence computed in the online phase, we get k' by the 192-bit index $Y_6^0[6] \| \cdots \| Y_6^{15}[6] \| k_{-1}[6,11] \| u_1[12] \| u_3[1] \| k_4[4,5,6] \| k_5[11]$, and sieve the right key by verifying the consistency of k' and $\widehat{k}$. The probability of this filter is about $\frac{2^{64}}{2^{160}} = 2^{-96}$. Thus, the time complexity of the attack is reduced by a half, but the memory complexity is increased to $2^{192} \times 2^{1.5} = 2^{193.5}$, which is used to store 368-bit information $Y_6^0[6] \| \cdots \| Y_6^{15}[6] \| k_{-1}[6,11] \| u_1[12] \| u_3[1] \| k_4[4,5,6] \| k_5[11] \| k'$ in such case. Combined with data/time/memory tradeoff and weak-key method, the time complexity of the attack is about $2^{186.5}$, the memory complexity is about $2^{177.5}$.

The Attack Starting from the Third Round. For the attack starting from the third round, the 16-bit shared information $k_1[6,11]$ could be used as the index to convert the attack to 2^{16} weak-key attacks, where $k_1[6] = k_5[2] \oplus k_5[6] \oplus k_5[14]$ and $k_1[11] = k_5[7] \oplus k_5[11] \oplus k_6[3]$. The attack procedure is similar to above attack, in use of the data/time/memory tradeoff, the memory complexity of the attack is reduced to 2^{165} 128-bit spaces. If we use the information $k' \cap \widehat{k}$ instead of partial value of the sequence, the time complexity would be reduced to $2^{182.5}$, and the memory complexity is about $2^{165.5}$.

4.2 Reducing the Memory Complexity for the Attack on AES-256

This attack is based on the 5-round distinguisher, where the active byte of the $\delta-$set is defined in $W_0[3]$, and the output value is located in $Y_6[7]$.

Proposition 2. *If one encrypts the first 32 values* $(W_0^0, \cdots, W_0^{31})$ *of the* $\delta-$*set through 5-round AES-256, assuming a pair of the* $\delta-$*set satisfying the expected truncated differential characteristic, then the sequence* $Y_6^0[6] \| \cdots \| Y_6^{31}[6]$ *along with a 192-bit subkey* $u_2[1,4,11,14] \| k_3[0, \cdots, 15] \| k_4[3,4,9,14]$ *takes about* 2^{208} *values.*

According to the generic attack, we need to precompute all possible values of sequence $Y_6^0[6]\| \cdots \|Y_6^{31}[6]$. Then in the online phase, we collect 2^{144} pairs, and find a pair satisfying the differential path for each 192-bit subkey $k_{-1}[0,5,10,15]$ $\|k_8\|u_7[2,5,8,15]$. After that, construct the $\delta-$set, compute the sequence $Y_6^0[6]\| \cdots \|Y_6^{31}[6]$, and check whether it belongs to precomputation table. Finally, detect the consistency of $u_2[1,4,11,14]\|k_3[0,\cdots,15]\|k_4[3,4,9,14]$ and $k_{-1}[0,5,10,15]\|k_8\|u_7[2,5,8,15]$ to retrieve the correct key. Then the time complexity in precomputation phase is about $2^{208} \times 2^5 \times 2^{-2.2} = 2^{210.8}$. The memory complexity is about $2^{209.9}$ 128-bit words of memory, where we need to store 448-bit information. In online phase, the time complexity is about $2^{192} \times 2^5 \times 2^{-2.6} = 2^{194.4}$. The data complexity is about 2^{113} chosen plaintexts. By data/time/memory tradeoff, we precompute a fraction 2^{-8} possible sequences, then the data, time and memory complexities are 2^{121}, $2^{203.5}$ and $2^{201.9}$, respectively. Here, we consider to use the 32-bit information $k_{-1}[10,15]\|k_4[9,14]$ to convert the attack to 2^{32} weak-key attacks, then the memory complexity reduces to $2^{169.9}$. Note that the subkey $k_{-1}[10,15]$ and $k_4[9,14]$ are linear dependent on k_3 and k_8, respectively.

5 Conclusion

In this paper, we take advantage of some subkey relations in the truncated differential to reduce the memory complexity of meet-in-the-middle attack, which is the bottleneck of this kind of attack. For 9-round AES-192, the 16-bit subkey conditions are obtained in the construction of the δ-set sequence. Based on this, we propose the 9-round attack on AES-192. In particular, when the 9-round attack starts from the third round of AES, the time complexity is $2^{182.4}$ 9-round encryption, the data complexity is 2^{117} chosen plaintexts, and the memory complexity is 2^{181} blocks. Moreover, combining the key relations between the inline phase and online phase, we introduce an interesting method to decompose the whole attack into a series of weak-key attacks, which helps to reduce the memory complexity of the attack without increasing the data and time complexities. To the best of our knowledge, these attacks are the most efficient results in single-key model for 9-round AES-192 and AES-256.

Acknowledgments. We would like to thank anonymous reviewers for their very helpful comments on the paper. This work is supported by 973 Program (No. 2013CB834205), and the National Natural Science Foundation of China (No. 61133013, 61373142 and 61272035).

References

1. Biryukov, A., Dunkelman, O., Keller, N., Khovratovich, D., Shamir, A.: Key recovery attacks of practical complexity on AES-256 variants with up to 10 rounds. In: Gilbert, H. (ed.) EUROCRYPT 2010. LNCS, vol. 6110, pp. 299–319. Springer, Heidelberg (2010)
2. Biryukov, A., Khovratovich, D.: Related-key cryptanalysis of the full AES-192 and AES-256. In: Matsui, M. (ed.) ASIACRYPT 2009. LNCS, vol. 5912, pp. 1–18. Springer, Heidelberg (2009)
3. Biryukov, A., Khovratovich, D., Nikolić, I.: Distinguisher and related-key attack on the full AES-256. In: Halevi, S. (ed.) CRYPTO 2009. LNCS, vol. 5677, pp. 231–249. Springer, Heidelberg (2009)
4. Bogdanov, A., Khovratovich, D., Rechberger, C.: Biclique cryptanalysis of the full AES. In: Lee, D.H., Wang, X. (eds.) ASIACRYPT 2011. LNCS, vol. 7073, pp. 344–371. Springer, Heidelberg (2011)
5. Daemen, J., Rijmen, V.: AES proposal: Rijndael. In: First Advanced Encryption Standard (AES) Conference (1998)
6. Demirci, H., Selçuk, A.A.: A meet-in-the-middle attack on 8-round AES. In: Nyberg, K. (ed.) FSE 2008. LNCS, vol. 5086, pp. 116–126. Springer, Heidelberg (2008)
7. Demirci, H., Taşkın, I., Çoban, M., Baysal, A.: Improved meet-in-the-middle attacks on AES. In: Roy, B., Sendrier, N. (eds.) INDOCRYPT 2009. LNCS, vol. 5922, pp. 144–156. Springer, Heidelberg (2009)
8. Derbez, P., Fouque, P.-A.: Exhausting Demirci-Selçuk Meet-in-the-Middle attacks against reduced-round AES. In: Moriai, S. (ed.) FSE 2013. LNCS, vol. 8424, pp. 541–560. Springer, Heidelberg (2014)
9. Derbez, P., Fouque, P.-A., Jean, J.: Improved key recovery attacks on reduced-round `AES` in the single-key setting. In: Johansson, T., Nguyen, P.Q. (eds.) EUROCRYPT 2013. LNCS, vol. 7881, pp. 371–387. Springer, Heidelberg (2013)
10. Diffie, W., Hellman, M.E.: Special feature exhaustive cryptanalysis of the NBS data encryption standard. Computer **10**, 74–84 (1977)
11. Dunkelman, O., Keller, N., Shamir, A.: Improved single-key attacks on 8-round AES-192 and AES-256. In: Abe, M. (ed.) ASIACRYPT 2010. LNCS, vol. 6477, pp. 158–176. Springer, Heidelberg (2010)
12. Fouque, P.-A., Jean, J., Peyrin, T.: Structural evaluation of `AES` and chosen-key distinguisher of 9-round `AES-128`. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013, Part I. LNCS, vol. 8042, pp. 183–203. Springer, Heidelberg (2013)
13. Gilber, H., Minier, M.: A collision attack on 7 rounds of Rijndael. In: AES Candidate Conference, pp. 230–241 (2000)
14. Gilbert, H., Peyrin, T.: Super-Sbox cryptanalysis: improved attacks for AES-like permutations. In: Hong, S., Iwata, T. (eds.) FSE 2010. LNCS, vol. 6147, pp. 365–383. Springer, Heidelberg (2010)

15. Lu, J., Dunkelman, O., Keller, N., Kim, J.-S.: New impossible differential attacks on AES. In: Chowdhury, D.R., Rijmen, V., Das, A. (eds.) INDOCRYPT 2008. LNCS, vol. 5365, pp. 279–293. Springer, Heidelberg (2008)
16. Mala, H., Dakhilalian, M., Rijmen, V., Modarres-Hashemi, M.: Improved impossible differential cryptanalysis of 7-round AES-128. In: Gong, G., Gupta, K.C. (eds.) INDOCRYPT 2010. LNCS, vol. 6498, pp. 282–291. Springer, Heidelberg (2010)
17. National Institute of Standards and Technology. ADVANCED ENCRYPTION STANDARD. In: IPS PUB 197, Federal Information Processing Standards Publication (2001)
18. Wei, Y., Lu, J., Hu, Y.: Meet-in-the-middle attack on 8 rounds of the AES block cipher under 192 key bits. In: Bao, F., Weng, J. (eds.) ISPEC 2011. LNCS, vol. 6672, pp. 222–232. Springer, Heidelberg (2011)
19. Zhang, W., Wu, W., Feng, D.: New results on impossible differential cryptanalysis of reduced AES. In: Nam, K.-H., Rhee, G. (eds.) ICISC 2007. LNCS, vol. 4817, pp. 239–250. Springer, Heidelberg (2007)

Authenticated Encryption

CLOC: Authenticated Encryption for Short Input

Tetsu Iwata[1(✉)], Kazuhiko Minematsu[2], Jian Guo[3], and Sumio Morioka[4]

[1] Nagoya University, Nagoya, Japan
iwata@cse.nagoya-u.ac.jp
[2] NEC Corporation, Tokyo, Japan
k-minematsu@ah.jp.nec.com
[3] Nanyang Technological University, Singapore, Singapore
ntu.guo@gmail.com
[4] NEC Europe Ltd., London, UK
s-morioka@ak.jp.nec.com

Abstract. We define and analyze the security of a blockcipher mode of operation, CLOC, for provably secure authenticated encryption with associated data. The design of CLOC aims at optimizing previous schemes, CCM, EAX, and EAX-prime, in terms of the implementation overhead beyond the blockcipher, the precomputation complexity, and the memory requirement. With these features, CLOC is suitable for handling short input data, say 16 bytes, without needing precomputation nor large memory. This property is especially beneficial to small microprocessors, where the word size is typically 8 bits or 16 bits, and there are significant restrictions in the size and the number of registers. CLOC uses a variant of CFB mode in its encryption part and a variant of CBC MAC in the authentication part. We introduce various design techniques in order to achieve the above mentioned design goals. We prove CLOC secure, in a reduction-based provable security paradigm, under the assumption that the blockcipher is a pseudorandom permutation. We also present our preliminary implementation results.

Keywords: CLOC · Blockcipher · Authenticated encryption with associated data · Security analysis · Efficiency analysis

1 Introduction

Background. An authenticated encryption with associated data scheme (AEAD) is a symmetric key cryptographic primitive that provides both confidentiality and integrity of plaintexts, and integrity of associated data. There are several ways of designing AEADs, and we focus on a design based on a blockcipher. CCM [39] was proposed by Whiting, Housley, and Ferguson for use within the IEEE 802.11 standard for Wireless LANs. It is adopted as NIST recommendation [16], and is broadly used in practice [9,20,21]. The mode is 2-pass, meaning that we run two algorithms, one for encryption and one for authentication. It is provably secure [25],

C. Cid and C. Rechberger (Eds.): FSE 2014, LNCS 8540, pp. 149–167, 2015.
DOI: 10.1007/978-3-662-46706-0_8

but CCM suffers from a number of limitations, most notably it is not on-line; the encryption process cannot be started until knowing the whole input data. There are other issues in CCM [35], and EAX was proposed by Bellare, Rogaway, and Wagner to overcome these limitations [13]. EAX is included in ISO 19772 [9], and it has a number of attractive features; it is simple as it uses CMAC and CTR mode in a black-box manner, and it was designed by taking provable security into consideration. However, it has several implementation costs, and EAX-prime was designed by Moise, Beroset, Phinney, and Burns [31] to reduce the costs. It was designed to reduce the number of blockcipher calls both in precomputation and in processing the input data, to eliminate the key dependent constants, also called masks, to reduce memory requirement to store them, and to unify the associated data and the nonce, which contributes to reduce the memory requirement and the number of blockcipher calls as well. However, a practical attack was pointed out against EAX-prime [30], showing that it is not a secure AEAD. Later, Minematsu, Lucks, and Iwata proposed a variant of EAX called EAX$^+$, which has similar complexity as EAX-prime and is provably secure as EAX [29].

Presumably, though not clearly stated in the document [31], the most significant advantage of EAX-prime over original EAX (and CCM) is its efficient handling of *short input data* with small memory. As EAX-prime needs only one blockcipher call in precomputation whereas EAX needs three calls, EAX-prime gains the performance for short (say 16 bytes) input data, in particular if precomputation is difficult due to a limited amount of memory, or frequent key changes, or both. The performance for short input data is important for many practical applications, most notably for low-power wireless sensor networks, since messages are typically short to suppress the energy consumption of sensor nodes, which are usually battery-powered. For example, Zigbee [8] limits the maximum message length to be 127 bytes, and Bluetooth low energy limits the length to 47 bytes [4]. Another example is Electronic Product Code (EPC), which is a replacement of bar-code using RFID tags, and it typically has 96 bits [5].

Our Contributions. In this paper, we present a mode of operation, CLOC (which stands for Compact Low-Overhead CFB, and is pronounced as "clock"), to meet the demand. The design of CLOC aims at optimizing previous schemes, CCM, EAX, and EAX-prime, in terms of the implementation overhead beyond the blockcipher, the precomputation complexity, and the memory requirement. CLOC is sequential and its asymptotic performance (i.e. for long input data) is comparable to CCM, EAX, and EAX-prime. However, CLOC has a unique feature in its low overhead computation. CLOC works *without* any precomputation beyond the key scheduling of the blockcipher. Specifically, we do not need any blockcipher calls nor generating a key dependent table. This contributes to the improvement of the performance for short input data. For example, when the input data consists of 1-block nonce, 1-block associated data, and 1-block plaintext, CLOC needs 4 blockcipher calls, while we need 5 or 6 calls in CCM, 7 calls (where 3 out of 7 can be precomputed) in EAX, and 5 calls (where 1 out of 5 can be precomputed) in EAX-prime. We focus on provably secure schemes, but for comparison, there are lightweight AE schemes including ALE [15] and FIDES [14], where ALE needs

44 AES rounds which amount to 4.4 AES calls (10 out of 44 AES rounds can be precomputed), and FIDES needs 33 round function calls, where the round function is similar to that of AES but has larger state. This property of CLOC is particularly beneficial for embedded devices since the internal blockcipher is relatively slow due to limited computing power. Moreover, CLOC can be implemented using only two state blocks, i.e. the working memory of $2n$ bits with an n-bit blockcipher, except those needed for interfacing and blockcipher invocations. We do not aware of any provably secure AE mode with on-line capability to work with such a small amount of memory, and this property makes CLOC even suitable for small processors.

Important properties of CLOC can be summarized as follows.

1. It is a nonce-based authenticated encryption with associated data (AEAD).
2. It uses only the encryption of the blockcipher both for encryption and decryption.
3. It makes $\lceil |N|/n \rceil + \lceil |A|/n \rceil + 2\lceil |M|/n \rceil$ blockcipher calls for a nonce N, associated data A, and a plaintext M, when $|A| \geq 1$, where $|X|$ is the length of X in bits and n is the block length in bits of the blockcipher. No precomputation is needed. We note that in CLOC, $1 \leq |N| \leq n-1$ holds (hence we always have $\lceil |N|/n \rceil = 1$), and when $|A| = 0$, it needs $\lceil |N|/n \rceil + 1 + 2\lceil |M|/n \rceil$ blockcipher calls.
4. It works with two state blocks (i.e. $2n$ bits).

We introduce various design techniques in order to achieve the above mentioned design goals. We introduce *tweak functions* which are used to update the internal state at several points in the encryption and the decryption. While bit-wise operations, such as a constant multiplication over $\mathrm{GF}(2^n)$, are often employed in majority of previous schemes, considering the performance for small devices, we completely eliminate bit-wise operations. Instead, our tweak functions consist of word-wise permutations and xor's. As a result, each tweak function can be described by using a 4×4 binary matrix.

The use of word-wise permutations and xor's to update a mask or a key dependent constant was discussed in [22,29], and the approach was applied on CMAC and EAX. Here we use them directly to update the internal state, instead of updating a key dependent constant and xoring it to the state. This was employed for example in designs of MACs [32,40] using bit shift operations. The techniques introduced here seem to be worth for other areas, e.g., in designing MACs, and thus it may be of independent interest.

We also introduce bit-fixing functions. CFB mode leaks input and output pairs of the underlying blockcipher, which may result in the loss of security. We use the functions to logically separate the encryption part and the authentication part of CLOC.

With these techniques, we prove CLOC secure, in a reduction-based provable security paradigm, under the assumption that the blockcipher is a pseudorandom permutation. For security notions, CLOC fulfills the standard security notions for nonce-based AEADs, i.e., the privacy and the authenticity under nonce-respecting adversaries [34]. Furthermore, we prove that the authenticity notion holds even for

Table 1. Comparison of AE modes, for a-block associated data and m-block plaintext with one-block nonce, where $a \geq 1$

Property∘	CCM [16]	GCM [17]	EAX [13]	EAX-prime [31]	OCB3 [26]	CLOC
Calls	$a + 2m + 2$†	$m + 1$‡	$a + 2m + 1$	$a + 2m + 1$§	$a + m + 1$†	$a + 2m + 1$
Setup	0	1	3	1	1	0
On-line	No	Yes	Yes	Yes	Yes	Yes
Static AD	No	Yes	Yes	Yes	Yes	Yes
Parallel	No	Yes	No	No	Yes	No
Primitive	E	E, GHASH	E	E	E, D	E
PRIV/AUTH⋆	$O(2^{n/2})$[25]	$O(2^{n/2})$[24]	$O(2^{n/2})$[13]	$O(1)$[30]	$O(2^{n/2})$[26]	$O(2^{n/2})$
N-AUTH⋄	$\ll 2^{n/2}$[18,19]	$O(1)$[18]	$O(1)$[18]	$O(1)$[30]	$O(1)$[18]	$O(2^{n/2})$

∘ "Setup" shows the number of blockcipher calls for setup, "Static AD" shows if efficient handling of static associated data is possible, "Parallel" shows if the blockcipher calls are parallelizable, and "Primitive" shows the components of the mode. E is the encryption of the blockcipher and D is the decryption.
† May have additional one call
‡ Plus $a + m$ multiplications over GF(2^n) for GHASH
§ Nonce and associated data are concatenated to form a 2-block "cleartext"
⋆ Attack workload of nonce-respecting adversaries to break the privacy notion or the authenticity notion
⋄ Attack workload of nonce-reusing adversaries to break the authenticity notion

nonce-reusing adversaries, where only a small number of schemes achieve this goal, and most of known modes do fail to provide [18]. See Table 1 for a brief comparison of CLOC to other AEADs.

2 Preliminaries

Let $\{0,1\}^*$ be the set of all finite bit strings, including the empty string ε. For an integer $\ell \geq 0$, let $\{0,1\}^\ell$ be the set of all bit strings of ℓ bits. For $X, Y \in \{0,1\}^*$, we write $X \,\|\, Y$, (X, Y), or simply XY to denote their concatenation. For $\ell \geq 0$, we write $0^\ell \in \{0,1\}^\ell$ to denote the bit string that consists of ℓ zeros, and $1^\ell \in \{0,1\}^\ell$ to denote the bit string that consists of ℓ ones. For $X \in \{0,1\}^*$, $|X|$ is its length in bits, and for $\ell \geq 1$, $|X|_\ell = \lceil |X|/\ell \rceil$ is the length in ℓ-bit blocks. For $X \in \{0,1\}^*$ and $\ell \geq 0$ such that $|X| \geq \ell$, $\mathsf{msb}_\ell(X)$ is the most significant (the leftmost) ℓ bits of X. For instance we have $\mathsf{msb}_1(1100) = 1$ and $\mathsf{msb}_3(1100) = 110$. For $X \in \{0,1\}^*$ and $\ell \geq 1$, we write its partition into ℓ-bit blocks as $(X[1], \ldots, X[x]) \xleftarrow{\ell} X$, which is defined as follows. If $X = \varepsilon$, then $x = 1$ and $X[1] \xleftarrow{\ell} X$, where $X[1] = \varepsilon$. Otherwise $X[1], \ldots, X[x] \in \{0,1\}^*$ are unique bit strings such that $X[1] \,\|\, \cdots \,\|\, X[x] = X$, $|X[1]| = \cdots = |X[x-1]| = \ell$, and $1 \leq |X[x]| \leq \ell$. For a finite set $\mathcal{X}$, $X \xleftarrow{\$} \mathcal{X}$ means that X is chosen uniformly random from $\mathcal{X}$.

In what follows, we fix a block length n and a blockcipher $E : \mathcal{K}_E \times \{0,1\}^n \to \{0,1\}^n$, where $\mathcal{K}_E$ is a non-empty set of keys. Let $\mathrm{Perm}(n)$ be the set of all permutations over $\{0,1\}^n$. We write $E_K \in \mathrm{Perm}(n)$ for the permutation specified by $K \in \mathcal{K}_E$, and $C = E_K(M)$ for the ciphertext of plaintext $M \in \{0,1\}^n$ under key $K \in \mathcal{K}_E$.

3 Specification of CLOC

CLOC takes three parameters, a blockcipher $E : \mathcal{K}_E \times \{0,1\}^n \to \{0,1\}^n$, a nonce length ℓ_N, and a tag length τ. We require $1 \leq \ell_N \leq n-1$ and $1 \leq \tau \leq n$. We also

require that $n/4$ is an integer. We write $\mathrm{CLOC}[E, \ell_N, \tau]$ for CLOC that is parameterized by E, ℓ_N, and τ, and we often omit the parameters if they are irrelevant or they are clear from the context. $\mathrm{CLOC}[E, \ell_N, \tau] = (\text{CLOC-}\mathcal{E}, \text{CLOC-}\mathcal{D})$ consists of the encryption algorithm CLOC-$\mathcal{E}$ and the decryption algorithm CLOC-$\mathcal{D}$.

CLOC-$\mathcal{E}$ and CLOC-$\mathcal{D}$ have the following syntax.

$$\begin{cases} \text{CLOC-}\mathcal{E} : \mathcal{K}_{\mathrm{CLOC}} \times \mathcal{N}_{\mathrm{CLOC}} \times \mathcal{A}_{\mathrm{CLOC}} \times \mathcal{M}_{\mathrm{CLOC}} \to \mathcal{CT}_{\mathrm{CLOC}} \\ \text{CLOC-}\mathcal{D} : \mathcal{K}_{\mathrm{CLOC}} \times \mathcal{N}_{\mathrm{CLOC}} \times \mathcal{A}_{\mathrm{CLOC}} \times \mathcal{CT}_{\mathrm{CLOC}} \to \mathcal{M}_{\mathrm{CLOC}} \cup \{\perp\} \end{cases}$$

$\mathcal{K}_{\mathrm{CLOC}} = \mathcal{K}_E$ is the key space, which is identical to the key space of the underlying blockcipher, $\mathcal{N}_{\mathrm{CLOC}} = \{0,1\}^{\ell_N}$ is the nonce space, $\mathcal{A}_{\mathrm{CLOC}} = \{0,1\}^*$ is the associated data space, $\mathcal{M}_{\mathrm{CLOC}} = \{0,1\}^*$ is the plaintext space, $\mathcal{CT}_{\mathrm{CLOC}} = \mathcal{C}_{\mathrm{CLOC}} \times \mathcal{T}_{\mathrm{CLOC}}$ is the ciphertext space, where $\mathcal{C}_{\mathrm{CLOC}} = \{0,1\}^*$ and $\mathcal{T}_{\mathrm{CLOC}} = \{0,1\}^\tau$ is the tag space, and $\perp \notin \mathcal{M}_{\mathrm{CLOC}}$ is the distinguished reject symbol. We write $(C,T) \leftarrow \text{CLOC-}\mathcal{E}_K(N,A,M)$ and $M \leftarrow \text{CLOC-}\mathcal{D}_K(N,A,C,T)$ or $\perp \leftarrow \text{CLOC-}\mathcal{D}_K(N,A,C,T)$, where $(C,T) \in \mathcal{CT}_{\mathrm{CLOC}}$ is a ciphertext, and we also call $C \in \mathcal{C}_{\mathrm{CLOC}}$ a ciphertext.

CLOC-$\mathcal{E}$ and CLOC-$\mathcal{D}$ are defined in Fig. 1. In these algorithms, we use four subroutines, HASH, PRF, ENC, and DEC. They have the following syntax.

$$\begin{cases} \mathsf{HASH} : \mathcal{K}_{\mathrm{CLOC}} \times \mathcal{N}_{\mathrm{CLOC}} \times \mathcal{A}_{\mathrm{CLOC}} \to \{0,1\}^n \\ \mathsf{PRF} : \mathcal{K}_{\mathrm{CLOC}} \times \{0,1\}^n \times \mathcal{C}_{\mathrm{CLOC}} \to \mathcal{T}_{\mathrm{CLOC}} \\ \mathsf{ENC} : \mathcal{K}_{\mathrm{CLOC}} \times \{0,1\}^n \times \mathcal{M}_{\mathrm{CLOC}} \to \mathcal{C}_{\mathrm{CLOC}} \\ \mathsf{DEC} : \mathcal{K}_{\mathrm{CLOC}} \times \{0,1\}^n \times \mathcal{C}_{\mathrm{CLOC}} \to \mathcal{M}_{\mathrm{CLOC}} \end{cases}$$

These subroutines are defined in Fig. 2, and illustrated in Figs. 3, 4, and 5. In the figures, i is the identity function, and $\mathsf{i}(X) = X$ for all $X \in \{0,1\}^n$. In the subroutines, we use the one-zero padding function $\mathsf{ozp} : \{0,1\}^* \to \{0,1\}^*$, the bit-fixing functions $\mathsf{fix0}, \mathsf{fix1} : \{0,1\}^* \to \{0,1\}^*$, and five tweak functions f_1, f_2, g_1, g_2, and h, which are functions over $\{0,1\}^n$.

The one-zero padding function ozp is used to adjust the length of an input string so that the total length becomes a positive multiple of n bits. For $X \in \{0,1\}^*$, $\mathsf{ozp}(X)$ is defined as $\mathsf{ozp}(X) = X$ if $|X| = \ell n$ for some $\ell \geq 1$, and $\mathsf{ozp}(X) = X \,\|\, 10^{n-1-(|X| \bmod n)}$ otherwise. We note that $\mathsf{ozp}(\varepsilon) = 10^{n-1}$, and we also note that, in general, the function is not invertible.

The bit-fixing functions $\mathsf{fix0}$ and $\mathsf{fix1}$ are used to fix the most significant bit of an input string to zero and one, respectively. For $X \in \{0,1\}^*$, $\mathsf{fix0}(X)$ is defined as $\mathsf{fix0}(X) = X \wedge 01^{|X|-1}$, and $\mathsf{fix1}(X)$ is defined as $\mathsf{fix1}(X) = X \vee 10^{|X|-1}$, where $\wedge$ and $\vee$ are the bit-wise AND operation, and the bit-wise OR operation, respectively.

The tweak function h is used in HASH if the most significant bit of $\mathsf{ozp}(A[1])$ is zero. We use f_1 and f_2 in HASH and PRF, where f_1 is used if the last input block is full (i.e., if $|A[a]| = n$ or $|C[m]| = n$) and f_2 is used otherwise. We use g_1 and g_2 in PRF, where we use g_1 if the second argument of the input is the empty string (i.e., $|C| = 0$), and otherwise we use g_2. Now for $X \in \{0,1\}^n$, let $(X[1], X[2], X[3], X[4]) \xleftarrow{n/4} X$. Then f_1, f_2, g_1, g_2, and h are defined as follows.

Algorithm CLOC-$\mathcal{E}_K(N, A, M)$	**Algorithm** CLOC-$\mathcal{D}_K(N, A, C, T)$
1. $V \leftarrow \mathsf{HASH}_K(N, A)$	1. $V \leftarrow \mathsf{HASH}_K(N, A)$
2. $C \leftarrow \mathsf{ENC}_K(V, M)$	2. $T^* \leftarrow \mathsf{PRF}_K(V, C)$
3. $T \leftarrow \mathsf{PRF}_K(V, C)$	3. **if** $T \neq T^*$ **then return** $\perp$
4. **return** (C, T)	4. $M \leftarrow \mathsf{DEC}_K(V, C)$
	5. **return** M

Fig. 1. Pseudocode of the encryption and the decryption algorithms of CLOC

$$\begin{cases} \mathsf{f}_1(X) = (X[1,3], X[2,4], X[1,2,3], X[2,3,4]) \\ \mathsf{f}_2(X) = (X[2], X[3], X[4], X[1,2]) \\ \mathsf{g}_1(X) = (X[3], X[4], X[1,2], X[2,3]) \\ \mathsf{g}_2(X) = (X[2], X[3], X[4], X[1,2]) \\ \mathsf{h}(X) = (X[1,2], X[2,3], X[3,4], X[1,2,4]) \end{cases}$$

Here $X[a, b]$ stands for $X[a] \oplus X[b]$ and $X[a, b, c]$ stands for $X[a] \oplus X[b] \oplus X[c]$.

Alternatively the tweak functions can be specified by a matrix. Let

$$\mathbf{M} = \begin{pmatrix} 0\,0\,0\,1 \\ 1\,0\,0\,1 \\ 0\,1\,0\,0 \\ 0\,0\,1\,0 \end{pmatrix} \tag{1}$$

be a 4×4 binary matrix, and let $\mathbf{M}^i$ for $i \geq 0$ be exponentiations of $\mathbf{M}$, where $\mathbf{M}^0$ denotes the identity matrix. Then we have $\mathsf{f}_1(X) = X \cdot \mathbf{M}^8$, $\mathsf{f}_2(X) = X \cdot \mathbf{M}$, $\mathsf{g}_1(X) = X \cdot \mathbf{M}^2$, $\mathsf{g}_2(X) = X \cdot \mathbf{M}$, and $\mathsf{h}(X) = X \cdot \mathbf{M}^4$, where $X = (X[1], X[2], X[3], X[4])$ is interpreted as a vector.

The design rationale for the tweak functions is explained in Sect. 4.

4 Design Rationale

Overall Structure. At abstract level CLOC is a straightforward combination of CFB mode and CBC MAC, where CBC MAC is called twice for processing associated data and a ciphertext, and CFB mode is called once to generate a ciphertext. However, when we want to achieve low-overhead computation and small memory consumption, we found that any other combination of a basic encryption mode and a MAC mode did not work. For instance, we could not use CTR mode or OFB mode, as they require one state block in processing a plaintext to hold a counter value or a blockcipher output. We then realized that combining CFB mode and CBC MAC was not an easy task. Since we avoid using two keys or using blockcipher pre-calls, such as $L = E_K(0^n)$ used in EAX, we could not computationally separate CFB mode and CBC MAC via input masking, such as Galois-field doubling ($2^i L$ for the i-th block, where $2L$ denotes the multiplication of 2 and L in GF(2^n)) [13,33]. This implies that CFB mode leaks input and output pairs of

Algorithm $\mathsf{HASH}_K(N, A)$

1. $(A[1], \ldots, A[a]) \xleftarrow{n} A$
2. $S_{\mathsf{H}}[1] \leftarrow E_K(\mathsf{fix0}(\mathsf{ozp}(A[1])))$
3. **if** $\mathsf{msb}_1(\mathsf{ozp}(A[1])) = 1$ **then**
4. $\quad S_{\mathsf{H}}[1] \leftarrow \mathsf{h}(S_{\mathsf{H}}[1])$
5. **if** $a \geq 2$ **then**
6. $\quad$ **for** $i \leftarrow 2$ **to** $a-1$ **do**
7. $\quad\quad S_{\mathsf{H}}[i] \leftarrow E_K(S_{\mathsf{H}}[i-1] \oplus A[i])$
8. $\quad S_{\mathsf{H}}[a] \leftarrow E_K(S_{\mathsf{H}}[a-1] \oplus \mathsf{ozp}(A[a]))$
9. **if** $|A[a]| = n$ **then**
10. $\quad V \leftarrow \mathsf{f}_1(S_{\mathsf{H}}[a] \oplus \mathsf{ozp}(N))$
11. **else** // $0 \leq |A[a]| \leq n-1$
12. $\quad V \leftarrow \mathsf{f}_2(S_{\mathsf{H}}[a] \oplus \mathsf{ozp}(N))$
13. **return** V

Algorithm $\mathsf{PRF}_K(V, C)$

1. **if** $|C| = 0$ **then**
2. $\quad T \leftarrow \mathsf{msb}_\tau(E_K(\mathsf{g}_1(V)))$
3. $\quad$ **return** T
4. $(C[1], \ldots, C[m]) \xleftarrow{n} C$
5. $S_{\mathsf{P}}[0] \leftarrow E_K(\mathsf{g}_2(V))$
6. **for** $i \leftarrow 1$ **to** $m-1$ **do**
7. $\quad S_{\mathsf{P}}[i] \leftarrow E_K(S_{\mathsf{P}}[i-1] \oplus C[i])$
8. **if** $|C[m]| = n$ **then**
9. $\quad S_{\mathsf{P}}[m] \leftarrow E_K(\mathsf{f}_1(S_{\mathsf{P}}[m-1] \oplus C[m]))$
10. **else** // $1 \leq |C[m]| \leq n-1$
11. $\quad S_{\mathsf{P}}[m] \leftarrow E_K(\mathsf{f}_2(S_{\mathsf{P}}[m-1] \oplus \mathsf{ozp}(C[m])))$
12. $T \leftarrow \mathsf{msb}_\tau(S_{\mathsf{P}}[m])$
13. **return** T

Algorithm $\mathsf{ENC}_K(V, M)$

1. **if** $|M| = 0$ **then**
2. $\quad C \leftarrow \varepsilon$
3. $\quad$ **return** C
4. $(M[1], \ldots, M[m]) \xleftarrow{n} M$
5. $S_{\mathsf{E}}[1] \leftarrow E_K(V)$
6. **for** $i \leftarrow 1$ **to** $m-1$ **do**
7. $\quad C[i] \leftarrow S_{\mathsf{E}}[i] \oplus M[i]$
8. $\quad S_{\mathsf{E}}[i+1] \leftarrow E_K(\mathsf{fix1}(C[i]))$
9. $C[m] \leftarrow \mathsf{msb}_{|M[m]|}(S_{\mathsf{E}}[m]) \oplus M[m]$
10. $C \leftarrow (C[1], \ldots, C[m])$
11. **return** C

Algorithm $\mathsf{DEC}_K(V, C)$

1. **if** $|C| = 0$ **then**
2. $\quad M \leftarrow \varepsilon$
3. $\quad$ **return** M
4. $(C[1], \ldots, C[m]) \xleftarrow{n} C$
5. $S_{\mathsf{D}}[1] \leftarrow E_K(V)$
6. **for** $i \leftarrow 1$ **to** $m-1$ **do**
7. $\quad M[i] \leftarrow S_{\mathsf{D}}[i] \oplus C[i]$
8. $\quad S_{\mathsf{D}}[i+1] \leftarrow E_K(\mathsf{fix1}(C[i]))$
9. $M[m] \leftarrow \mathsf{msb}_{|C[m]|}(S_{\mathsf{D}}[m]) \oplus C[m]$
10. $M \leftarrow (M[1], \ldots, M[m])$
11. **return** M

Fig. 2. Subroutines used in the encryption and decryption algorithms of CLOC

the blockcipher calls, which can be freely used to guess or fake the internal chaining value of CBC MAC, leading to a break of the scheme. Lucks [28] proposed an AEAD scheme based on CFB mode, called CCFB. However, the problem is not relevant to CCFB due to the difference in the global structure. To overcome this obstacle in composition, we introduced the bit-fixing functions. Their role is to absolutely separate the input blocks of the blockcipher in CFB mode and *the first input block* of CBC MAC. This imposes the most significant one bit of the input of CBC MAC being fixed to 0, implying one-bit input loss. The set of five tweak functions, which is another tool we introduced in this paper, is used to compensate for this information loss. It also works to compensate the information loss caused by padding functions applied to the last input block to CBC MAC. A similar technique can be found in literature [32,40], however, the previous works only considered MACs and the tweak functions required bit operations.

In the following we explain the specific requirements for the tweak functions.

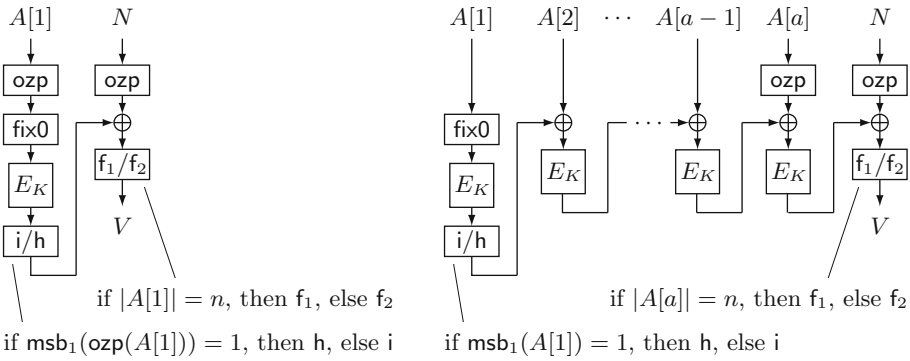


Fig. 3. $V \leftarrow \mathsf{HASH}_K(N, A)$ for $0 \le |A| \le n$ (left) and $|A| \ge n + 1$ (right)

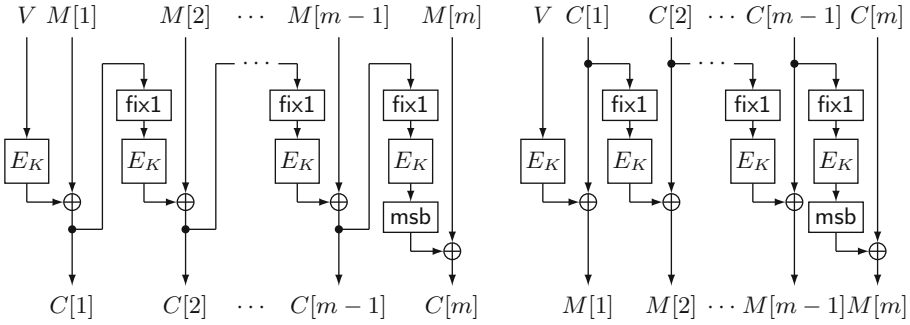


Fig. 4. $C \leftarrow \mathsf{ENC}_K(V, M)$ for $|M| \ge 1$ (left), and $\mathsf{DEC}_K(V, C)$ for $|C| \ge 1$ (right)

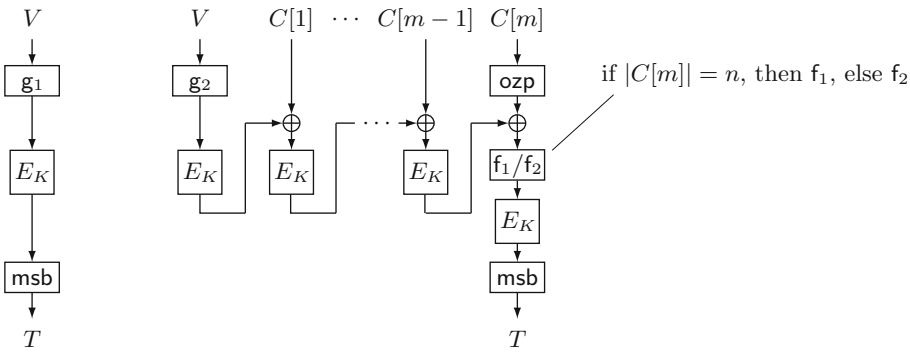


Fig. 5. $T \leftarrow \mathsf{PRF}_K(V, C)$ for $|C| = 0$ (left) and $|C| \ge 1$ (right)

$\mathsf{i} \oplus \mathsf{f}_1$	$\mathsf{i} \oplus \mathsf{f}_2\mathsf{h}$	$\mathsf{f}_1 \oplus \mathsf{f}_2\mathsf{h}$	$\mathsf{h} \oplus \mathsf{g}_2\mathsf{f}_1$	$\mathsf{g}_2\mathsf{f}_1 \oplus \mathsf{g}_1\mathsf{f}_2\mathsf{h}$
$\mathsf{i} \oplus \mathsf{g}_1\mathsf{f}_1$	$\mathsf{i} \oplus \mathsf{h}$	$\mathsf{f}_2 \oplus \mathsf{g}_1\mathsf{f}_1$	$\mathsf{h} \oplus \mathsf{f}_2$	$\mathsf{g}_2\mathsf{f}_1 \oplus \mathsf{f}_2\mathsf{h}$
$\mathsf{i} \oplus \mathsf{g}_1\mathsf{f}_1\mathsf{h}$	$\mathsf{i} \oplus \mathsf{g}_1$	$\mathsf{f}_2 \oplus \mathsf{g}_1\mathsf{f}_1\mathsf{h}$	$\mathsf{h} \oplus \mathsf{g}_1\mathsf{f}_2$	$\mathsf{g}_1\mathsf{f}_2 \oplus \mathsf{g}_2\mathsf{f}_1$
$\mathsf{i} \oplus \mathsf{g}_2\mathsf{f}_1$	$\mathsf{i} \oplus \mathsf{g}_2$	$\mathsf{f}_2 \oplus \mathsf{g}_2\mathsf{f}_1$	$\mathsf{h} \oplus \mathsf{g}_2\mathsf{f}_2$	$\mathsf{g}_1\mathsf{f}_2 \oplus \mathsf{g}_2\mathsf{f}_1\mathsf{h}$
$\mathsf{i} \oplus \mathsf{g}_2\mathsf{f}_1\mathsf{h}$	$\mathsf{f}_1 \oplus \mathsf{g}_1\mathsf{f}_1\mathsf{h}$	$\mathsf{f}_2 \oplus \mathsf{g}_2\mathsf{f}_1\mathsf{h}$	$\mathsf{g}_1\mathsf{f}_1 \oplus \mathsf{f}_1\mathsf{h}$	$\mathsf{g}_1\mathsf{f}_2 \oplus \mathsf{f}_1\mathsf{h}$
$\mathsf{i} \oplus \mathsf{f}_1\mathsf{h}$	$\mathsf{f}_1 \oplus \mathsf{g}_2\mathsf{f}_1\mathsf{h}$	$\mathsf{f}_2 \oplus \mathsf{f}_1\mathsf{h}$	$\mathsf{g}_1\mathsf{f}_1 \oplus \mathsf{g}_2\mathsf{f}_1\mathsf{h}$	$\mathsf{g}_1\mathsf{f}_2 \oplus \mathsf{g}_2\mathsf{f}_2\mathsf{h}$
$\mathsf{i} \oplus \mathsf{f}_2$	$\mathsf{f}_1 \oplus \mathsf{f}_2$	$\mathsf{f}_2 \oplus \mathsf{g}_1\mathsf{f}_2\mathsf{h}$	$\mathsf{g}_1\mathsf{f}_1 \oplus \mathsf{g}_2\mathsf{f}_2$	$\mathsf{g}_1\mathsf{f}_2 \oplus \mathsf{f}_2\mathsf{h}$
$\mathsf{i} \oplus \mathsf{g}_1\mathsf{f}_2$	$\mathsf{f}_1 \oplus \mathsf{g}_1\mathsf{f}_2$	$\mathsf{f}_2 \oplus \mathsf{g}_2\mathsf{f}_2\mathsf{h}$	$\mathsf{g}_1\mathsf{f}_1 \oplus \mathsf{g}_2\mathsf{f}_2\mathsf{h}$	$\mathsf{g}_2\mathsf{f}_2 \oplus \mathsf{g}_1\mathsf{f}_1\mathsf{h}$
$\mathsf{i} \oplus \mathsf{g}_1\mathsf{f}_2\mathsf{h}$	$\mathsf{f}_1 \oplus \mathsf{g}_1\mathsf{f}_2\mathsf{h}$	$\mathsf{g}_1 \oplus \mathsf{g}_2$	$\mathsf{g}_1\mathsf{f}_1 \oplus \mathsf{f}_2\mathsf{h}$	$\mathsf{g}_2\mathsf{f}_2 \oplus \mathsf{f}_1\mathsf{h}$
$\mathsf{i} \oplus \mathsf{g}_2\mathsf{f}_2$	$\mathsf{f}_1 \oplus \mathsf{g}_2\mathsf{f}_2$	$\mathsf{h} \oplus \mathsf{f}_1$	$\mathsf{g}_2\mathsf{f}_1 \oplus \mathsf{g}_1\mathsf{f}_1\mathsf{h}$	$\mathsf{g}_2\mathsf{f}_2 \oplus \mathsf{g}_1\mathsf{f}_2\mathsf{h}$
$\mathsf{i} \oplus \mathsf{g}_2\mathsf{f}_2\mathsf{h}$	$\mathsf{f}_1 \oplus \mathsf{g}_2\mathsf{f}_2\mathsf{h}$	$\mathsf{h} \oplus \mathsf{g}_1\mathsf{f}_1$	$\mathsf{g}_2\mathsf{f}_1 \oplus \mathsf{f}_1\mathsf{h}$	$\mathsf{g}_2\mathsf{f}_2 \oplus \mathsf{f}_2\mathsf{h}$

Fig. 6. Differential probability constraints of $\mathsf{f}_1, \mathsf{f}_2, \mathsf{g}_1, \mathsf{g}_2$, and h

Definition of $\mathsf{f}_1, \mathsf{f}_2, \mathsf{g}_1, \mathsf{g}_2$, *and* h. These functions are defined to meet the following properties. First, they have the additive property. That is, for any $\mathsf{z} \in \{\mathsf{f}_1, \mathsf{f}_2, \mathsf{g}_1, \mathsf{g}_2, \mathsf{h}\}$, we have $\mathsf{z}(X \oplus X') = \mathsf{z}(X) \oplus \mathsf{z}(X')$ for all $X, X' \in \{0,1\}^n$. Next, these functions are invertible over $\{0,1\}^n$. For any $\mathsf{z} \in \{\mathsf{f}_1, \mathsf{f}_2, \mathsf{g}_1, \mathsf{g}_2, \mathsf{h}\}$, we have $\mathsf{z} \in \mathrm{Perm}(n)$. Finally, they satisfy the differential probability constraints specified in Fig. 6. Let z be a function in Fig. 6. Then we require that, for any $Y \in \{0,1\}^n$, $\Pr[\mathsf{z}(K) = Y] = 1/2^n$, where the probability is taken over $K \xleftarrow{\$} \{0,1\}^n$. When z is of the form $\mathsf{z} = \mathsf{z}' \oplus \mathsf{z}''$, then $\mathsf{z}(K)$ stands for $\mathsf{z}'(K) \oplus \mathsf{z}''(K)$. When z is of the form $\mathsf{z} = \mathsf{z}'\mathsf{z}''$, then $\mathsf{z}(K)$ stands for $\mathsf{z}'(\mathsf{z}''(K))$. Recall that we define i as $\mathsf{i}(K) = K$.

Choosing Tweak Functions. Finding simple and word-wise tweak functions fulfilling all properties is not a trivial task. We start with matrix $\mathbf{M}$ of (1), which is invertible and has order 15 (i.e. $\mathbf{M}^{15} = \mathbf{M}^0$), and test all combinations of the form $(\mathsf{f}_1, \mathsf{f}_2, \mathsf{g}_1, \mathsf{g}_2, \mathsf{h}) = (i_1, \ldots, i_5) \in \{1, \ldots, 14\}^5$, where $i_1 = 2$ means $\mathsf{f}_1(X) = X \cdot \mathbf{M}^2$, using a computer. There are 864 candidates out of 537,824 fulfilling the differential probability constraints of Fig. 6. The complexity increases as the index of $\mathbf{M}$ grows, when we implement the tweak function by iterating $\mathbf{M}$, which seems suitable for hardware. For software we would directly implement $\mathbf{M}^i$ using a word-wise permutation and xor, and in this case we observe slight irregular, but similar phenomena (e.g. $\mathbf{M}^1$ needs one xor while $\mathbf{M}^3$ needs three xor's). Figure 7 shows $\mathbf{M}^i$ and the Feistel-like implementations using a word-wise permutation and xor. It shows that, except for $\mathbf{M}^5$ and $\mathbf{M}^{10}$, we have a simple implementation using at most four xor's. Based on these observations, we simply define the cost of computing $\mathbf{M}^i$ as i for $1 \le i \le 7$ and $15 - i$ for $8 \le i \le 14$, and define $f_{\mathrm{cost}}(i_1, \ldots, i_5)$ as

$$\left(i_1 \times \frac{1}{16} + i_2 \times \frac{15}{16}\right) \times 2 + i_4 + i_5 \times \frac{1}{2}.$$

This corresponds to the expected total cost for given $(i_1, \ldots, i_5)$, where associated data and a plaintext are assumed to be non-empty byte strings of random lengths (as we expect the standard use of CLOC is AEAD, not MAC), and we also assume that the most significant bit of the associated data is random. Then there remains

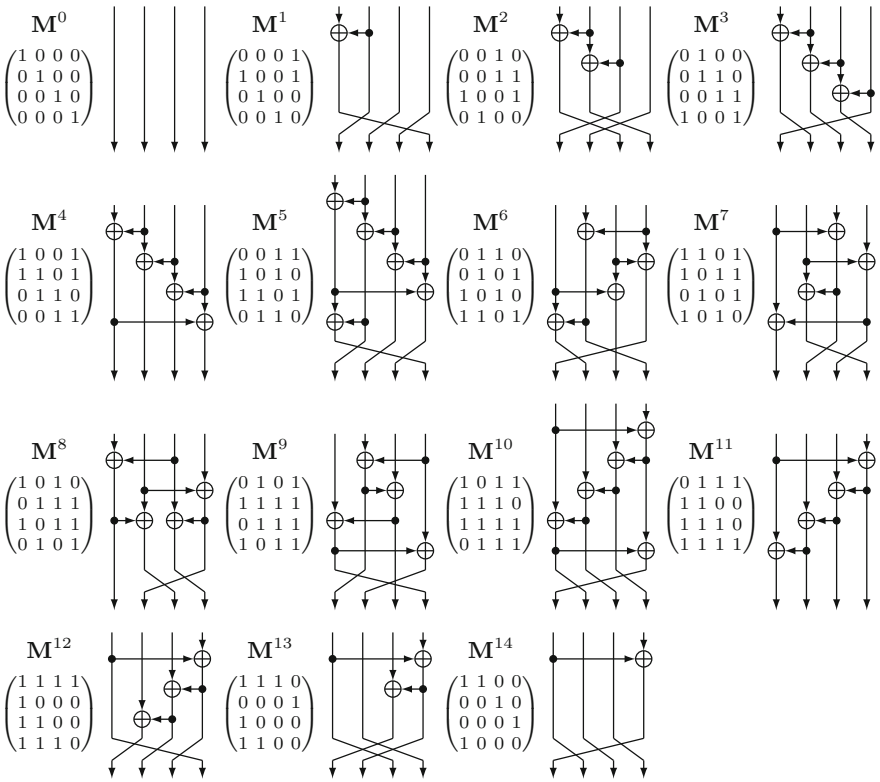


Fig. 7. Matrix exponentiations for the tweak functions

only two candidates giving the minimum value of f_{cost}, which are $(i_1, \ldots, i_5) = (8, 1, 2, 1, 4)$ and $(8, 1, 6, 1, 4)$. As smaller i_3 is better, we choose the former as the sole winner. We also tested other matrices, say the one replacing the forth column of $\mathbf{M}$ by the transposition of $(1, 0, 1, 0)$, but no better solution was found.

We note that $\mathbf{M}^8 = \mathbf{M}^2 \oplus \mathbf{M}^0$ and $\mathbf{M}^4 = \mathbf{M}^1 \oplus \mathbf{M}^0$ hold, implying that we have $\mathsf{f}_1(X) = \mathsf{g}_1(X) \oplus X$ and $\mathsf{h}(X) = \mathsf{f}_2(X) \oplus X = \mathsf{g}_2(X) \oplus X$, which may be useful in some implementations.

5 Security of CLOC

In this section, we define the security notions of a blockcipher and CLOC, and present our security theorems.

PRP Notion. We assume that the blockcipher $E : \mathcal{K}_E \times \{0,1\}^n \to \{0,1\}^n$ is a pseudo-random permutation, or a PRP [27]. We say that P is a random permutation if $P \xleftarrow{\$} \mathrm{Perm}(n)$, and define

$$\mathbf{Adv}^{\mathrm{prp}}_E(\mathcal{A}) \stackrel{\mathrm{def}}{=} \Pr\left[\mathcal{A}^{E_K(\cdot)} \Rightarrow 1\right] - \Pr\left[\mathcal{A}^{P(\cdot)} \Rightarrow 1\right],$$

where the first probability is taken over $K \xleftarrow{\$} \mathcal{K}_E$ and the randomness of $\mathcal{A}$, and the last is over $P \xleftarrow{\$} \mathrm{Perm}(n)$ and $\mathcal{A}$. We write CLOC[Perm$(n), \ell_N, \tau$] for CLOC that uses P as E_K, and the encryption and decryption algorithms are written as CLOC-$\mathcal{E}_P$ and CLOC-$\mathcal{D}_P$. We also consider CLOC that uses a random function as E_K, which is naturally defined as the invertibility of E_K is irrelevant in the definition of CLOC. Let Rand(n) be the set of all functions from $\{0,1\}^n$ to $\{0,1\}^n$, and we say that R is a random function if $R \xleftarrow{\$} \mathrm{Rand}(n)$. We write CLOC[Rand$(n)$, ℓ_N, τ] for CLOC that uses R as E_K, and its encryption and decryption algorithms are written as CLOC-$\mathcal{E}_R$ and CLOC-$\mathcal{D}_R$.

Privacy Notion. We define the privacy notion for CLOC$[E, \ell_N, \tau]$ = (CLOC-$\mathcal{E}$, CLOC-$\mathcal{D}$). This notion captures the indistinguishably of a nonce-respecting adversary in a chosen plaintext attack setting [34]. We consider an adversary $\mathcal{A}$ that has access to the CLOC encryption oracle, or a random-bits oracle. The encryption oracle takes $(N, A, M) \in \mathcal{N}_{\mathrm{CLOC}} \times \mathcal{A}_{\mathrm{CLOC}} \times \mathcal{M}_{\mathrm{CLOC}}$ as input and returns $(C, T) \leftarrow$ CLOC-$\mathcal{E}_K(N, A, M)$. The random-bits oracle, \$-oracle, takes $(N, A, M) \in \mathcal{N}_{\mathrm{CLOC}} \times \mathcal{A}_{\mathrm{CLOC}} \times \mathcal{M}_{\mathrm{CLOC}}$ as input and returns a random string $(C, T) \xleftarrow{\$} \{0,1\}^{|M|+\tau}$. We define the privacy advantage as

$$\mathbf{Adv}^{\mathrm{priv}}_{\mathrm{CLOC}[E,\ell_N,\tau]}(\mathcal{A}) \stackrel{\mathrm{def}}{=} \Pr\left[\mathcal{A}^{\text{CLOC-}\mathcal{E}_K(\cdot,\cdot,\cdot)} \Rightarrow 1\right] - \Pr\left[\mathcal{A}^{\$(\cdot,\cdot,\cdot)} \Rightarrow 1\right],$$

where the first probability is taken over $K \xleftarrow{\$} \mathcal{K}_{\mathrm{CLOC}}$ and the randomness of $\mathcal{A}$, and the last is over the random-bits oracle and $\mathcal{A}$. We assume that $\mathcal{A}$ in the privacy game is nonce-respecting, that is, $\mathcal{A}$ does not make two queries with the same nonce.

Privacy Theorem. Let $\mathcal{A}$ be an adversary that makes q queries, and suppose that the queries are $(N_1, A_1, M_1), \ldots, (N_q, A_q, M_q)$. Then we define the total associated data length as $a_1 + \cdots + a_q$, and the total plaintext length as $m_1 + \cdots + m_q$, where $(A_i[1], \ldots, A_i[a_i]) \xleftarrow{n} A_i$ and $(M_i[1], \ldots, M_i[m_i]) \xleftarrow{n} M_i$. We have the following information theoretic result.

Theorem 1. *Let* Perm(n), ℓ_N, *and* τ *be the parameters of* CLOC. *Let* $\mathcal{A}$ *be an adversary that makes at most* q *queries, where the total associated data length is at most* σ_A, *and the total plaintext length is at most* σ_M. *Then we have* $\mathbf{Adv}^{\mathrm{priv}}_{\mathrm{CLOC}[\mathrm{Perm}(n),\ell_N,\tau]}(\mathcal{A}) \le 5\sigma^2_{\mathrm{priv}}/2^n$, *where* $\sigma_{\mathrm{priv}} = q + \sigma_A + 2\sigma_M$.

A proof overview is given in Sect. 6, and a complete proof is presented in [23, Appendix A]. If we use a blockcipher E, which is secure in the sense of the PRP notion, instead of Perm(n), then the corresponding complexity theoretic result can be shown by a standard argument. See e.g. [11]. We note that the privacy of CLOC is broken if the nonce is reused.

Authenticity Notion. We next define the authenticity notion, which captures the unforgeability of an adversary in a chosen ciphertext attack setting [34]. We consider a strong adversary that can repeat the same nonce multiple times. Let $\mathcal{A}$ be an adversary that has access to the CLOC encryption oracle and the CLOC

decryption oracle. The encryption oracle is defined as above. The decryption oracle takes $(N, A, C, T) \in \mathcal{N}_{\mathrm{CLOC}} \times \mathcal{A}_{\mathrm{CLOC}} \times \mathcal{C}_{\mathrm{CLOC}} \times \mathcal{T}_{\mathrm{CLOC}}$ as input and returns $M \leftarrow \text{CLOC-}\mathcal{D}_K(N, A, C, T)$ or $\perp \leftarrow \text{CLOC-}\mathcal{D}_K(N, A, C, T)$. The authenticity advantage is defined as

$$\mathbf{Adv}^{\mathrm{auth}}_{\mathrm{CLOC}[E,\ell_N,\tau]}(\mathcal{A}) \stackrel{\mathrm{def}}{=} \Pr\left[\mathcal{A}^{\text{CLOC-}\mathcal{E}_K(\cdot,\cdot,\cdot),\text{CLOC-}\mathcal{D}_K(\cdot,\cdot,\cdot,\cdot)} \text{ forges}\right],$$

where the probability is taken over $K \stackrel{\$}{\leftarrow} \mathcal{K}_{\mathrm{CLOC}}$ and the randomness of $\mathcal{A}$, and the adversary forges if the decryption oracle returns a bit string (other than $\perp$) for a query (N, A, C, T), but (C, T) was not previously returned to $\mathcal{A}$ from the encryption oracle for a query (N, A, M). The adversary $\mathcal{A}$ in the authenticity game is not necessarily nonce-respecting, and $\mathcal{A}$ can make two or more queries with the same nonce. Specifically, $\mathcal{A}$ can repeat using the same nonce for encryption queries, a nonce used for encryption queries can be used for decryption queries and vice-versa, and the same nonce can be repeated for decryption queries. Without loss of generality, we assume that $\mathcal{A}$ does not make trivial queries, i.e., if the encryption oracle returns (C, T) for a query (N, A, M), then $\mathcal{A}$ does not make a query (N, A, C, T) to the decryption oracle, and $\mathcal{A}$ does not repeat a query.

Authenticity Theorem. Let $\mathcal{A}$ be an adversary that makes q encryption queries and q' decryption queries. Let $(N_1, A_1, M_1), \ldots, (N_q, A_q, M_q)$ be the encryption queries, and $(N'_1, A'_1, C'_1, T'_1), \ldots, (N'_{q'}, A'_{q'}, C'_{q'}, T'_{q'})$ be the decryption queries. Then we define the total associated data length in encryption queries as $a_1 + \cdots + a_q$, the total plaintext length as $m_1 + \cdots + m_q$, the total associated data length in decryption queries as $a'_1 + \cdots + a'_{q'}$, and the total ciphertext length as $m'_1 + \cdots + m'_{q'}$, where $(A_i[1], \ldots, A_i[a_i]) \stackrel{n}{\leftarrow} A_i$, $(M_i[1], \ldots, M_i[m_i]) \stackrel{n}{\leftarrow} M_i$, $(A'_i[1], \ldots, A'_i[a'_i]) \stackrel{n}{\leftarrow} A'_i$, and $(C'_i[1], \ldots, C'_i[m'_i]) \stackrel{n}{\leftarrow} C'_i$. We have the following information theoretic result.

Theorem 2. *Let* $\mathrm{Perm}(n)$*,* ℓ_N*, and* τ *be the parameters of* CLOC*. Let* $\mathcal{A}$ *be an adversary that makes at most* q *encryption queries and at most* q' *decryption queries, where the total associated data length in encryption queries is at most* σ_A*, the total plaintext length is at most* σ_M*, the total associated data length in decryption queries is at most* $\sigma_{A'}$*, and the total ciphertext length is at most* $\sigma_{C'}$*. Then we have* $\mathbf{Adv}^{\mathrm{auth}}_{\mathrm{CLOC}[\mathrm{Perm}(n),\ell_N,\tau]}(\mathcal{A}) \leq 5\sigma^2_{\mathrm{auth}}/2^n + q'/2^\tau$*, where* $\sigma_{\mathrm{auth}} = q + \sigma_A + 2\sigma_M + q' + \sigma_{A'} + \sigma_{C'}$*.*

A proof overview is given in Sect. 6, and a complete proof is presented in [23, Appendix A]. As in the privacy case, if we use a blockcipher E secure in the sense of the PRP notion, then we obtain the corresponding complexity theoretic result by a standard argument in, e.g., [11].

6 Overview of Security Proofs

PRP/PRF Switching. The first step is to replace the random permutation P in $\mathrm{CLOC}[\mathrm{Perm}(n), \ell_N, \tau]$ with a random function R, and use the PRP/PRF switching lemma [12] to obtain the followingdifferences.

$$\begin{cases} \mathbf{Adv}^{\mathrm{priv}}_{\mathrm{CLOC}[\mathrm{Perm}(n),\ell_N,\tau]}(\mathcal{A}) - \mathbf{Adv}^{\mathrm{priv}}_{\mathrm{CLOC}[\mathrm{Rand}(n),\ell_N,\tau]}(\mathcal{A}) \\ \mathbf{Adv}^{\mathrm{auth}}_{\mathrm{CLOC}[\mathrm{Perm}(n),\ell_N,\tau]}(\mathcal{A}) - \mathbf{Adv}^{\mathrm{auth}}_{\mathrm{CLOC}[\mathrm{Rand}(n),\ell_N,\tau]}(\mathcal{A}) \end{cases}$$

Defining $Q_1, \ldots, Q_{26}$ *and* CLOC2. We define twenty six functions $Q_1, \ldots, Q_{26} : \{0,1\}^n \to \{0,1\}^n$ based on R, K_1, K_2, and K_3, where $K_1, K_2, K_3 \xleftarrow{\$} \{0,1\}^n$ are three independent random n-bit strings. We also define a modified version of $\mathrm{CLOC}[\mathrm{Rand}(n), \ell_N, \tau]$ called $\mathrm{CLOC2}[\ell_N, \tau]$, which uses $Q = (Q_1, \ldots, Q_{26})$ as oracles. Q and CLOC2 are designed so that CLOC-$\mathcal{E}_R$ and CLOC2-$\mathcal{E}_Q$ are the same algorithms, CLOC-$\mathcal{D}_R$ and CLOC2-$\mathcal{D}_Q$ are the same algorithms (except that CLOC2-$\mathcal{D}_Q$ is used for the verification only, and it does not output a plaintext even if the verification succeeds), and $Q_1, \ldots, Q_{26}$ are indistinguishable from $F_1, \ldots, F_{26}$, which are independent random functions. We then have

$$\begin{cases} \mathbf{Adv}^{\mathrm{priv}}_{\mathrm{CLOC}[\mathrm{Rand}(n),\ell_N,\tau]}(\mathcal{A}) = \mathbf{Adv}^{\mathrm{priv}}_{\mathrm{CLOC2}[\ell_N,\tau]}(\mathcal{A}), \\ \mathbf{Adv}^{\mathrm{auth}}_{\mathrm{CLOC}[\mathrm{Rand}(n),\ell_N,\tau]}(\mathcal{A}) = \mathbf{Adv}^{\mathrm{auth}}_{\mathrm{CLOC2}[\ell_N,\tau]}(\mathcal{A}), \end{cases}$$

and we show the distinguishing probability of $Q = (Q_1, \ldots, Q_{26})$ and $F = (F_1, \ldots, F_{26})$ in [23, Lemma 1]. However, the indistinguishability does not hold for arbitrary adversaries. We formalize an input-respecting adversary, and our indistinguishability result in [23, Lemma 1] holds only for these adversaries.

The three random strings, K_1, K_2, and K_3, are secret keys from the adversary's perspective, and we introduce them to show the indistinguishability between Q and F. For instance we know that the input $\mathsf{fix0}(\mathsf{ozp}(A[1]))$ to produce $S_{\mathsf{H}}[1]$ in $\mathsf{HASH}_K(N, A)$ (The 2nd line of $\mathsf{HASH}_K(N, A)$ in Fig. 2) never collides with the input $\mathsf{fix1}(C[i])$ to produce $S_{\mathsf{E}}[i+1]$ in $\mathsf{ENC}_K(V, M)$ (The 8th line of $\mathsf{ENC}_K(V, M)$ in Fig. 2), and hence we can safely assume that they are independent. Likewise, we show that the collision probability between $\mathsf{fix0}(\mathsf{ozp}(A[1]))$ and, say, $S_{\mathsf{H}}[i-1] \oplus A[i]$ in $\mathsf{HASH}_K(N, A)$ (The 7th line of $\mathsf{HASH}_K(N, A)$ in Fig. 2) is low, and the three random strings are introduced to help this argument.

Defining CLOC3. We define another version of $\mathrm{CLOC}[\mathrm{Rand}(n), \ell_N, \tau]$ called $\mathrm{CLOC3}[\ell_N, \tau]$. It uses $F = (F_1, \ldots, F_{26})$ as oracles, and the encryption algorithm CLOC3-$\mathcal{E}_F$ and the decryption algorithm CLOC3-$\mathcal{D}_F$ are obtained from CLOC2-$\mathcal{E}_Q$ and CLOC2-$\mathcal{D}_Q$ by replacing $Q_1, \ldots, Q_{26}$ with $F_1, \ldots, F_{26}$, respectively. We use [23, Lemma 1] to obtain the following differences.

$$\begin{cases} \mathbf{Adv}^{\mathrm{priv}}_{\mathrm{CLOC2}[\ell_N,\tau]}(\mathcal{A}) - \mathbf{Adv}^{\mathrm{priv}}_{\mathrm{CLOC3}[\ell_N,\tau]}(\mathcal{A}) \\ \mathbf{Adv}^{\mathrm{auth}}_{\mathrm{CLOC2}[\ell_N,\tau]}(\mathcal{A}) - \mathbf{Adv}^{\mathrm{auth}}_{\mathrm{CLOC3}[\ell_N,\tau]}(\mathcal{A}) \end{cases}$$

The simulations work with input-respecting adversaries, and hence [23, Lemma 1] is sufficient for our purpose.

Indistinguishability of $(\mathsf{HASH3}, \mathsf{HASH3}', \mathsf{HASH3}'')$. We then consider three subroutines $\mathsf{HASH3}$, $\mathsf{HASH3}'$, and $\mathsf{HASH3}''$ in $\mathrm{CLOC3}[\ell_N, \tau]$. $\mathsf{HASH3}$ roughly corresponds to a function that computes $S_{\mathsf{E}}[1]$ from (N, A) in $\mathrm{CLOC}[E, \ell_N, \tau]$, i.e.,

$E_K(\mathsf{HASH}_K(N, A))$. $\mathsf{HASH3}'$ computes the tag T when $|C| = 0$, i.e., this function roughly corresponds to $\mathsf{msb}_\tau(E_K(\mathsf{g}_1(\mathsf{HASH}_K(N, A))))$. $\mathsf{HASH3}''$ computes $S_\mathsf{P}[0]$ from (N, A), which is used when $|C| \geq 1$, i.e., $E_K(\mathsf{g}_2(\mathsf{HASH}_K(N, A)))$. Then in [23, Lemma 2], we show that these functions are indistinguishable from three independent random functions $\mathsf{HASH4}$, $\mathsf{HASH4}'$, and $\mathsf{HASH4}''$.

Defining CLOC4. We define another version of CLOC[Rand$(n), \ell_N, \tau$], called CLOC4[ℓ_N, τ]. This is obtained by replacing $\mathsf{HASH3}$, $\mathsf{HASH3}'$, and $\mathsf{HASH3}''$ in CLOC3 with $\mathsf{HASH4}$, $\mathsf{HASH4}'$, and $\mathsf{HASH4}''$, respectively. We use [23, Lemma 2] to obtain the following differences.

$$\begin{cases} \mathbf{Adv}^{\mathrm{priv}}_{\mathrm{CLOC3}[\ell_N,\tau]}(\mathcal{A}) - \mathbf{Adv}^{\mathrm{priv}}_{\mathrm{CLOC4}[\ell_N,\tau]}(\mathcal{A}) \\ \mathbf{Adv}^{\mathrm{auth}}_{\mathrm{CLOC3}[\ell_N,\tau]}(\mathcal{A}) - \mathbf{Adv}^{\mathrm{auth}}_{\mathrm{CLOC4}[\ell_N,\tau]}(\mathcal{A}) \end{cases}$$

Indistinguishability of $\mathsf{PRF4}$. We then consider a subroutine called $\mathsf{PRF4}$ in CLOC4. This function outputs a tag T from (N, A, C), and internally uses $\mathsf{HASH4}'$, $\mathsf{HASH4}''$, F_{24}, F_{25}, and F_{26}. We show in [23, Lemma 3] that this function is indistinguishable from a random function $\mathsf{PRF5}$.

Defining CLOC5. We define our final version of CLOC[Rand$(n), \ell_N, \tau$], called CLOC5[ℓ_N, τ], which is obtained from CLOC4 by replacing $\mathsf{PRF4}$ with $\mathsf{PRF5}$. This function is used in both encryption and decryption, and we obtain the following differences from [23, Lemma 3].

$$\begin{cases} \mathbf{Adv}^{\mathrm{priv}}_{\mathrm{CLOC4}[\ell_N,\tau]}(\mathcal{A}) - \mathbf{Adv}^{\mathrm{priv}}_{\mathrm{CLOC5}[\ell_N,\tau]}(\mathcal{A}) \\ \mathbf{Adv}^{\mathrm{auth}}_{\mathrm{CLOC4}[\ell_N,\tau]}(\mathcal{A}) - \mathbf{Adv}^{\mathrm{auth}}_{\mathrm{CLOC5}[\ell_N,\tau]}(\mathcal{A}) \end{cases}$$

Privacy and Authenticity of CLOC5. Finally, we analyze the privacy and the authenticity of CLOC5 in [23, Lemma 4]. The privacy result shows the upper bound on $\mathbf{Adv}^{\mathrm{priv}}_{\mathrm{CLOC5}[\ell_N,\tau]}(\mathcal{A})$, and the proof is reduced to bounding the collision probability among the input values of the random function which is used to encrypt plaintexts. The authenticity result shows the upper bound on $\mathbf{Adv}^{\mathrm{auth}}_{\mathrm{CLOC5}[\ell_N,\tau]}(\mathcal{A})$, and its proof is simple and the result is obtained from the fact that the adversary, even if the nonce is reused, has to guess the output of a random function $\mathsf{PRF5}$ for the input that was not queried before.

We finally obtain the proofs of Theorems 1 and 2 by combining the above differences between advantage functions.

7 Software Implementation

We first tested CLOC on a general-purpose CPU. It is interesting to note that the encryption process and tag generation can be done in parallel, which could speed up the overall computation by a factor close to 2 for long plaintexts, then the final speed could be close to that of encryption only in serial mode. To show that, we implemented CLOC instantiated with AES-128 using the AES new instruction set, and

tested against Intel processor, Core i5-3427U 1.80GHz (Ivy Bridge) [6]. It is known that Intel's AES instruction allows fast parallel processing (up to 4 or 8 blocks), and we used this technique for two parallel inputs to AES. The tested speed for long plaintexts (more than 2^{20} blocks) is around 4.9 cycles per byte (cpb), while AES-128 encrypts at a speed of 4.3 cpb in serial mode. In Table 2, we provide the test vectors.

We then tested CLOC on embedded software. We used an 8-bit microprocessor, Atmel AVR ATmega128 [2]. For comparison we also implemented EAX [13], EAX-prime [31], and OCB3 [26]. For OCB3 we used a byte-oriented code from [7]. OCB3 needs relatively large precomputation for GF doublings, but we modify the code so that the doublings are on-line, since large precomputation may not be suitable to handle short input data for microprocessors. We also considered GCM for comparison, however, recent studies show that GCM does not perform well on constrained devices (see e.g. [10,38]), hence we decided not to include it. All modes are written in C and combined with AES-128. Our AES code is taken from [3], which is written in assembler. AES runs at 156.7 cpb for encryption, 196.8 cpb for decryption, both without key scheduling, and the key scheduling runs at 1,979 cycles. Our codes are complied with Atmel Studio 6 available from [2]. Cycles counts are measured on the simulator of Atmel Studio 6. Table 3 shows the implementation result. ROM denotes the object size in bytes. The speed is measured based on the scenario of non-static associated data, i.e., we excluded key setup and other computations before processing associated data and a nonce, defined as "Init", and figures for Data b denote cycles per byte to process a b-byte plaintext. In EAX, "Init" includes the computation of $E_K(0^n)$, $E_K(0^{n-1}1)$, and $E_K(0^{n-2}10)$. The length of associated data is fixed to 16 bytes except for EAX-prime, and for EAX-prime, we use 32-byte "cleartext," which can be regarded as the combination of associated data and a nonce [31]. For OCB3 we also measured the decryption performance, whereas those of CLOC, EAX, and EAX-prime are almost the same as encryption, since CLOC, EAX and EAX-prime require only forward direction of the underlying blockcipher. The result shows a superior performance of CLOC for short input data, up to around 128 bytes, which would be sufficiently long for

Table 2. Test vector of CLOC instantiated with AES-128

	Length (bytes)	Value (in hex)
Key	16	`000102030405060708090a0b0c0d0e0f`
Associated data	14	`ff000102030405060708090a0b0c0d`
Nonce	12	`001122334455667788 99aabb`
Plaintext	30	`86012204ccebf09ad5305ea8967aebd0`
		`0dd9c05cbde9407ff1ef52f043a2`
Ciphertext	30	`ebd908c23eac555dee406434fb2cffd4`
		`e1bee4401002063e2d13cdf9df3b`
Tag	16	`6621dae27674aa6fbc303426824b2c05`

Table 3. Software implementation on ATmega128

	ROM (bytes)	RAM (bytes)	Init (cycles)	Speed (cycles/byte)					
				Data 16	32	64	96	128	256
CLOC	2980	362	1999	750.1	549.0	448.4	414.9	398.2	373.0
EAX	2772	402	12996	913.6	632.5	490.8	443.6	419.9	384.5
EAX-prime	2588	421	5102	908.7	638.7	496.6	449.3	425.6	390.1
OCB-E	5010	971	4956	1217.5	736.1	495.5	412.2	375.1	315.0
OCB-D	5010	971	4955	1252.2	773.4	534.0	451.2	414.3	354.4

low-power wireless networks, as we mentioned in Sect. 1. We also measure the RAM usage of the AVR implementations, using a public tool [41], based on data of 16 bytes. It is clear to see that CLOC requires much less RAM than OCB3.

8 Hardware Implementation

Although the primary focus of CLOC is embedded software, we also implemented CLOC on hardware to see basic performance figures. We used Altera FPGA, Cyclone IV GX (EP4CGX110DF31C7) [1], and implemented CLOC using AES-128. AES implementation is round-based, and the S-box of AES is based on a composite field [37]. For reference we also wrote EAX for the same device, using the same AES. Both CLOC and EAX use one AES core for encryption and authentication. In EAX implementation, all input masks are stored to registers. Table 4 shows the results. The size is measured by the number of logic elements (LEs). Our implementation is not optimized. Still, these figures show that CLOC has slightly smaller size and faster speed than EAX. Table 4 lacks other important modes, in particular OCB. A more comprehensive comparison and optimized implementation for short input data are interesting future topics.

Table 4. Hardware implementation. Throughput figures of CLOC and EAX are measured for 8-block plaintexts with one-block associated data.

	Size (LE)	Max. Freq. (MHz)	Throughput (Mbit/sec)
CLOC	5628	82.1	400.7
EAX	6453	61.3	342.2
AES Enc.	3175	98.7	971.7

9 Conclusions

We presented a blockcipher mode of operation called CLOC for authenticated encryption with associated data. It uses a variant of CFB mode in its encryption part and a variant of CBC MAC in the authentication part. The scheme efficiently

handles short input data without heavy precomputation nor large memory, and it is suitable for use in microprocessors. We proved CLOC secure, in a reduction-based provable security paradigm, under the assumption that the blockcipher is a pseudorandom permutation. We also presented our preliminary implementation results.

It would be interesting to see improved implementation results using possibly lightweight blockciphers.

Acknowledgments. The authors would like to thank the anonymous FSE 2014 reviewers for helpful comments. The work by Tetsu Iwata was carried out in part while visiting Nanyang Technological University, Singapore. The work by Jian Guo was supported by the Singapore National Research Foundation Fellowship 2012 (NRF-NRFF2012-06).

References

1. Altera Corporation. http://www.altera.com/
2. ATMEL Corporation. http://www.atmel.com/
3. AVR-Crypto-Lib. http://www.das-labor.org/wiki/AVR-Crypto-Lib/en/
4. Bluetooth low energy. http://www.bluetooth.com/Pages/Low-Energy.aspx/
5. Electronic Product Code (EPC) Tag Data Standard (TDS). http://www.epcglobalinc.org/standards/tds/
6. Intel Corporation. http://www.intel.com/
7. OCB News and Code. http://www.cs.ucdavis.edu/~rogaway/ocb/news/
8. ZigBee Alliance. http://www.zigbee.org/
9. Information Technology – Security Techniques – Authenticated Encryption, ISO/IEC 19772:2009. International Standard ISO/IEC 19772 (2009)
10. Bauer, G.R., Potisk, P., Tillich, S.: Comparing Block Cipher Modes of Operation on MICAz Sensor Nodes. In: Baz, D.E., Spies, F., Gross, T. (eds.) PDP, pp. 371–378. IEEE Computer Society (2009)
11. Bellare, M., Kilian, J., Rogaway, P.: The Security of the Cipher Block Chaining Message Authentication Code. J. Comput. Syst. Sci. **61**(3), 362–399 (2000)
12. Bellare, M., Rogaway, P.: The Security of Triple Encryption and a Framework for Code-Based Game-Playing Proofs. In: Vaudenay, S. (ed.) EUROCRYPT 2006. LNCS, vol. 4004, pp. 409–426. Springer, Heidelberg (2006)
13. Bellare, M., Rogaway, P., Wagner, D.: The EAX Mode of Operation. In: Roy and Meier [36], pp. 389–407
14. Bilgin, B., Bogdanov, A., Knežević, M., Mendel, F., Wang, Q.: FIDES: Lightweight Authenticated Cipher with Side-Channel Resistance for Constrained Hardware. In: Bertoni, G., Coron, J.-S. (eds.) CHES 2013. LNCS, vol. 8086, pp. 142–158. Springer, Heidelberg (2013)
15. Bogdanov, A., Mendel, F., Regazzoni, F., Rijmen, V., Tischhauser, E.: ALE: AES-Based Lightweight Authenticated Encryption. In: Moriai, S. (ed.) FSE 2013. LNCS, vol. 8424, pp. 447–466. Springer, Heidelberg (2014)
16. Dworkin, M.: Recommendation for Block Cipher Modes of Operation: The CCM Mode for Authentication and Confidentiality, NIST Special Publication 800-38C (2004)
17. Dworkin, M.: Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC, NIST Special Publication 800-38D (2007)

18. Fleischmann, E., Forler, C., Lucks, S.: McOE: A Family of Almost Foolproof On-Line Authenticated Encryption Schemes. In: Canteaut, A. (ed.) FSE 2012. LNCS, vol. 7549, pp. 196–215. Springer, Heidelberg (2012)
19. Fouque, P.-A., Martinet, G., Valette, F., Zimmer, S.: On the Security of the CCM Encryption Mode and of a Slight Variant. In: Bellovin, S.M., Gennaro, R., Keromytis, A.D., Yung, M. (eds.) ACNS 2008. LNCS, vol. 5037, pp. 411–428. Springer, Heidelberg (2008)
20. Housley, R.: Using Advanced Encryption Standard (AES) CCM Mode with IPsec Encapsulating Security Payload (ESP). IETF RFC 4309 (2005)
21. Housley, R.: Using AES-CCM and AES-GCM Authenticated Encryption in the Cryptographic Message Syntax (CMS). IETF RFC 5084 (2007)
22. Iwata, T., Minematsu, K.: Generating a Fixed Number of Masks with Word Permutations and XORs. DIAC: Directions in Authenticated Ciphers (2013). http://2013.diac.cr.yp.to/
23. Iwata, T., Minematsu, K., Guo, J., Morioka, S.: CLOC: Authenticated Encryption for Short Input. Cryptology ePrint Archive, Report 2014/157 (2014). Full version of this paper http://eprint.iacr.org/
24. Iwata, T., Ohashi, K., Minematsu, K.: Breaking and Repairing GCM Security Proofs. In: Safavi-Naini, R., Canetti, R. (eds.) CRYPTO 2012. LNCS, vol. 7417, pp. 31–49. Springer, Heidelberg (2012)
25. Jonsson, J.: On the Security of CTR + CBC-MAC. In: Nyberg, K., Heys, H. (eds.) SAC 2002. LNCS, vol. 2595, pp. 76–93. Springer, Heidelberg (2003)
26. Krovetz, T., Rogaway, P.: The Software Performance of Authenticated-Encryption Modes. In: Joux, A. (ed.) FSE 2011. LNCS, vol. 6733, pp. 306–327. Springer, Heidelberg (2011)
27. Luby, M., Rackoff, C.: How to Construct Pseudorandom Permutations from Pseudorandom Functions. SIAM J. Comput. **17**(2), 373–386 (1988)
28. Lucks, S.: Two-Pass Authenticated Encryption Faster than Generic Composition. In: Gilbert, H., Handschuh, H. (eds.) FSE 2005. LNCS, vol. 3557, pp. 284–298. Springer, Heidelberg (2005)
29. Minematsu, K., Lucks, S., Iwata, T.: Improved Authenticity Bound of EAX, and Refinements. In: Susilo, W., Reyhanitabar, R. (eds.) ProvSec 2013. LNCS, vol. 8209, pp. 184–201. Springer, Heidelberg (2013)
30. Minematsu, K., Lucks, S., Morita, H., Iwata, T.: Attacks and Security Proofs of EAX-Prime. In: Moriai, S. (ed.) FSE 2013. LNCS, vol. 8424, pp. 327–347. Springer, Heidelberg (2014). http://eprint.iacr.org/2012/018
31. Moise, A., Beroset, E., Phinney, T., Burns, M.: EAX' Cipher Mode, NIST Submission (May 2011). http://csrc.nist.gov/groups/ST/toolkit/BCM/documents/proposedmodes/eax-prime/eax-prime-spec.pdf
32. Nandi, M.: Fast and Secure CBC-Type MAC Algorithms. In: Dunkelman, O. (ed.) FSE 2009. LNCS, vol. 5665, pp. 375–393. Springer, Heidelberg (2009)
33. 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)
34. Rogaway, P.: Nonce-Based Symmetric Encryption. In: Roy and Meier [36], pp. 348–359
35. Rogaway, P., Wagner, D.: A Critique of CCM (2003). http://csrc.nist.gov/groups/ST/toolkit/BCM/documents/comments/800-38_Series-Drafts/CCM/RW_CCM_comments.pdf
36. Roy, B.K., Meier, W. (eds.): FSE 2004. LNCS, vol. 3017. Springer, Heidelberg (2004)

37. Satoh, A., Morioka, S., Takano, K., Munetoh, S.: A Compact Rijndael Hardware Architecture with S-Box Optimization. In: Boyd, C. (ed.) ASIACRYPT 2001. LNCS, vol. 2248, pp. 239–254. Springer, Heidelberg (2001)
38. Simplício Jr., M.A., de Oliveira, B.T., Barreto, P.S.L.M., Margi, C.B., Carvalho, T.C.M.B., Näslund, M.: Comparison of Authenticated-Encryption Schemes in Wireless Sensor Networks. In: Chou, C.T., Pfeifer, T., Jayasumana, A.P. (eds.) LCN, pp. 450–457. IEEE (2011)
39. Whiting, D., Housley, R., Ferguson, N.: Counter with CBC-MAC. Submission to NIST (2002). http://csrc.nist.gov/groups/ST/toolkit/BCM/modes_development.html
40. Zhang, L., Wu, W., Zhang, L., Wang, P.: CBCR: CBC MAC with Rotating Transformations. Sci. CHINA Inf. Sci. **54**(11), 2247–2255 (2011)
41. Zharkov, E.: EZSTACK: A tool to measure the RAM usage of AVR implementations http://home.comcast.net/~ezstack/ezstack.c

APE: Authenticated Permutation-Based Encryption for Lightweight Cryptography

Elena Andreeva[1], Begül Bilgin[1,2], Andrey Bogdanov[3], Atul Luykx[1], Bart Mennink[1(✉)], Nicky Mouha[1], and Kan Yasuda[1,4]

[1] Department of Electrical Engineering, ESAT/COSIC, KU Leuven and iMinds, Ghents, Belgium
{elena.andreeva,begul.bilgin,atul.luykx, bart.mennink,nicky.mouha}@esat.kuleuven.be
[2] Faculty of EEMCS, DIES, UTwente, Enschede, Netherlands
[3] Department of Mathematics, Technical University of Denmark, Lyngby, Denmark
anbog@dtu.dk
[4] NTT Secure Platform Laboratories, Tokyo, Japan
yasuda.kan@lab.ntt.co.jp

Abstract. The domain of lightweight cryptography focuses on cryptographic algorithms for extremely constrained devices. It is very costly to avoid nonce reuse in such environments, because this requires either a hardware source of randomness, or non-volatile memory to store a counter. At the same time, a lot of cryptographic schemes actually require the nonce assumption for their security. In this paper, we propose APE as the first permutation-based authenticated encryption scheme that is resistant against nonce misuse. We formally prove that APE is secure, based on the security of the underlying permutation. To decrypt, APE processes the ciphertext blocks in reverse order, and uses inverse permutation calls. APE therefore requires a permutation that is both efficient for forward and inverse calls. We instantiate APE with the permutations of three recent lightweight hash function designs: Quark, Photon, and Spongent. For any of these permutations, an implementation that supports both encryption and decryption requires less than 1.9 kGE and 2.8 kGE for 80-bit and 128-bit security levels, respectively.

Keywords: APE · Authenticated encryption · Sponge function · Online · Deterministic · Permutation-based · Misuse resistant

1 Introduction

In constrained environments, conventional solutions to cryptographic problems are prohibitively expensive to implement. Lightweight cryptography deals with cryptographic algorithms within the stringent requirements imposed by devices such as low-cost smart cards, sensor networks, and electronic body implants where energy, power, or hardware area consumption can be heavily restricted.

Although symmetric-key cryptography predominantly makes use of solutions based on block ciphers, recently permutation-based constructions [9] are gaining

C. Cid and C. Rechberger (Eds.): FSE 2014, LNCS 8540, pp. 168–186, 2015.
DOI: 10.1007/978-3-662-46706-0_9

traction for a wide range of platforms, and on lightweight devices in particular. Lightweight permutation-based hash functions include Gluon [6], Photon [19], Quark [2,3], and Spongent [11].

Lightweight applications in practice require not only hash functions but also secret-key cryptographic functions, such as authenticated encryption (AE). AE is a cryptographic primitive that guarantees two security goals: privacy and integrity. The prevalent solutions in this direction are block cipher based [24,28,31]. Permutation-based AE schemes were only recently proposed, such as the deterministic key-wrap scheme [21] of Khovratovich and SpongeWrap [7,10] of the Keccak team.

These two constructions unfortunately have their limitations. With the key-wrap scheme [21], the message length is restricted to one block by design. While sufficient for key wrapping [29], this construction cannot handle arbitrary-length data and is therefore not a full AE scheme. SpongeWrap [7,10] can encrypt messages of varying lengths but relies on the uniqueness of the nonce value: failure to ensure so makes it possible to reuse the keystream of the encryption. For example, if a pair of plaintexts share a common prefix, the XOR of the first pair of plaintext blocks after this common prefix is leaked.

In Rogaway's security formalism of nonce-based encryption [26,27], the nonce is considered to be unique for every evaluation. While this approach has theoretical merits, in practice it is challenging to ensure that a nonce is never reused. This is especially the case in lightweight cryptography, as a nonce is realized either by keeping a state (and correctly updating it) or by providing a hardware source of randomness. Indeed, nonce misuse is a security threat in plenty of practical applications, not necessarily limited to the lightweight setting. Examples include flawed implementations of nonces [13,15,22,23,32], bad management of nonces by the user, and backup resets or virtual machine clones when the nonce is stored as a counter.

Nonce *misuse resistance* has become an important criterion in the design of AE schemes. The CAESAR competition [14] considers misuse resistance in detail for their selection of a portfolio of AE algorithms. The problem of nonce misuse has also been addressed by the recent deterministic AE scheme SIV [29], by the online AE scheme McOE [18], and in part by the aforementioned deterministic key-wrap scheme [21]. However, there are currently no permutation-based AE schemes that are resistant to nonce misuse.

Our Contributions. In this work we introduce APE (Authenticated Permutation-based Encryption). APE is the first permutation-based *and* nonce misuse resistant authenticated encryption scheme. APE is inspired by SpongeWrap [7,10], but differs in several fundamental aspects in order to achieve misuse resistance. Most importantly, in APE the ciphertexts are extracted from the state, whereas SpongeWrap generates a keystream to perform the encryption. APE encryption processes data in an online manner, whereas decryption is done backwards using the inverse of the permutation. APE is formally introduced in Sect. 3. Here, we initially focus on associated data and messages of an integral

number of blocks. In Appendix A, we show how APE can be generalized at almost no extra cost to handle fractional associated data and message blocks.

We prove that APE achieves privacy and integrity up to about $2^{c/2}$ queries, where c is the capacity parameter of APE. This result is proven in two different models: first, in Sect. 4, we prove security of APE in the ideal permutation model, where the underlying permutation is assumed to behave perfectly random. Next, in Sect. 5, we consider APE *with block ciphers*, which is a generalization of APE where the permutation with surrounding key XORs is replaced with a block cipher call. We use the results from the ideal model to prove that APE with block ciphers is secure in the standard model up to roughly $2^{c/2}$ queries as well.

APE is designed to be well suited for lightweight applications. However, APE decrypts in inverse direction and requires an efficiently invertible permutation. In Sect. 6, we implement APE in less than 1.9 kGE and 2.8 kGE for 80-bit and 128-bit security respectively with the permutations of Quark, Photon, and Spongent. The results indicate that including the inverses of these permutations only leads to a marginal increase of the size of the implementation when compared to the cost of providing a hardware source of randomness to generate nonces.

2 Notation

Set $\mathbf{R} := \{0,1\}^r$ and $\mathbf{C} := \{0,1\}^c$. Given two strings A and B, we use $A\|B$ and AB interchangeably, so for example $AB = A\|B \in \mathbf{R} \times \mathbf{C} \cong \{0,1\}^{r+c}$. Given $X \in \mathbf{R} \times \mathbf{C}$, X_r denotes its projection onto $\mathbf{R}$, also known as its rate part, and X_c denotes its projection onto $\mathbf{C}$, or capacity part. We write $0 \in \mathbf{R}$ for a shorthand for $00\cdots0 \in \mathbf{R}$ and $1 \in \mathbf{C}$ for $00\cdots01 \in \mathbf{C}$. The symbol $\oplus$ denotes the bitwise XOR operation of two (or more) strings.

An element of $\mathbf{R}$ is called a block. Let $\mathbf{R}^*$ denote the set of strings whose length is a multiple of r, with at most $2^{c/2}$ blocks. This explicit bound of $2^{c/2}$ is needed in order to define a random function as being sampled over a finite set of functions. Note that the bounds we prove become trivial for queries of length $2^{c/2}$ blocks. Similarly, let $\mathbf{R}^+$ denote the set of strings whose length is a positive multiple of r, with at most $2^{c/2}$ blocks. Given $M \in \mathbf{R}^+$, we divide it into blocks and write $M[1]M[2]\cdots M[w] \leftarrow M$, where each $M[i]$ is a block and w the block length of the string M.

Let $\mathcal{A}$ be some class of adversaries. For convenience, we use the notation

$$\Delta_{\mathcal{A}}[f,g] := \sup_{A\in\mathcal{A}} \left|\Pr[A^f = 1] - \Pr[A^g = 1]\right|$$

to denote the supremum of the distinguishing advantages over all adversaries distinguishing f and g. Providing access to multiple algorithms is denoted with

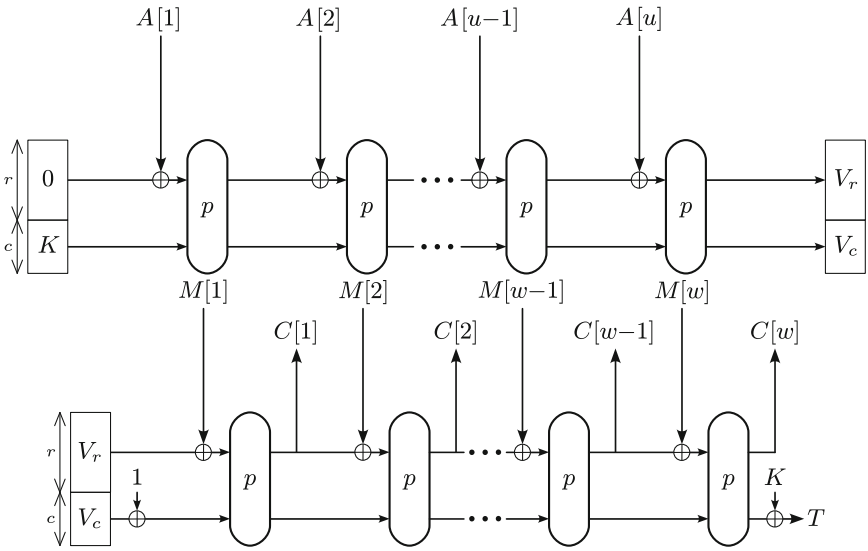


Fig. 1. The APE mode of operation (encryption). If there is no associated data ($A = \varnothing$), we have $V_r := 0$ and $V_c := K$.

a comma, e.g. $\Delta[f_1, f_2\,;\, g_1, g_2]$ denotes distinguishing the combination of f_1 and f_2 from the combination of g_1 and g_2.

3 APE Authenticated Encryption Mode

We now define our APE mode for the case of plaintexts and associated data of length a multiple of the block size. We refer to Sect. A for the generalization of APE to fractional data blocks. APE iterates a fixed permutation $p : \mathbf{R} \times \mathbf{C} \to \mathbf{R} \times \mathbf{C}$ in a way similar to the sponge construction. The permutation p is the only underlying cryptographic primitive used by APE. A diagram of APE is given in Fig. 1 and an algorithmic description in Fig. 2.

The encryption algorithm $\mathcal{E}$ takes as input a key $K \in \mathcal{K} = \mathbf{C}$, associated data $A \in \mathbf{R}^*$, and a message $M \in \mathbf{R}^+$, and returns a ciphertext $C \in \mathbf{R}^+$ and a tag $T \in \mathbf{C}$, as $(C, T) \leftarrow \mathcal{E}_K(A, M)$. On the other hand, $\mathcal{D}$ takes as input a key $K \in \mathbf{C}$, associated data $A \in \mathbf{R}^*$, a ciphertext $C \in \mathbf{R}^+$, and a tag $T \in \mathbf{C}$, and returns either a message $M \in \mathbf{R}^+$ or the reject symbol $\perp$, as $M/\perp \leftarrow \mathcal{D}_K(A, C, T)$. The two functionalities are sound, in the sense that whenever we encrypt a message as $(C, T) \leftarrow \mathcal{E}_K(A, M)$, we always get the message back, not $\perp$, via the decryption process $M \leftarrow \mathcal{D}_K(A, C, T)$.

In APE, the inclusion of a nonce is optional. If a nonce is required, it can be included as part of the associated data. Care must be taken when allowing nonces of varying lengths, as the nonce and the associated data should be clearly distinguishable.

Algorithm 1. $\mathcal{E}_K(A, M)$

Input: $K \in \mathbf{C}$, $A \in \mathbf{R}^*$, $M \in \mathbf{R}^+$
Output: $C \in \mathbf{R}^+$, $T \in \mathbf{C}$

$V \leftarrow (0, K) \in \mathbf{R} \times \mathbf{C}$
if $A \neq \varnothing$ **then**
  $A[1]A[2] \cdots A[u] \leftarrow A$
  for $i = 1$ **to** u **do**
    $V \leftarrow p\big(A[i] \oplus V_r, V_c\big)$
  end
end
$V \leftarrow V \oplus (0, 1)$
$M[1]M[2] \cdots M[w] \leftarrow M$
for $i = 1$ **to** w **do**
  $V \leftarrow p\big(M[i] \oplus V_r, V_c\big)$
  $C[i] \leftarrow V_r$
end
$C \leftarrow C[1]C[2] \cdots C[w]$
$T \leftarrow V_c \oplus K$
return (C, T)

Algorithm 2. $\mathcal{D}_K(A, C, T)$

Input: $K \in \mathbf{C}$, $A \in \mathbf{R}^*$, $C \in \mathbf{R}^+$, $T \in \mathbf{C}$
Output: $M \in \mathbf{R}^+$ or $\perp$

$V \leftarrow (0, K) \in \mathbf{R} \times \mathbf{C}$
if $A \neq \varnothing$ **then**
  $A[1]A[2] \cdots A[u] \leftarrow A$
  for $i = 1$ **to** u **do**
    $V \leftarrow p\big(A[i] \oplus V_r, V_c\big)$
  end
end
$C[1]C[2] \cdots C[w] \leftarrow C$
$C[0] \leftarrow V_r$
$V \leftarrow (C[w], K \oplus T)$
for $i = w$ **to** 1 **do**
  $V \leftarrow p^{-1}(V)$
  $M[i] \leftarrow C[i-1] \oplus V_r$
  $V \leftarrow C[i-1] \| V_c$
end
$M \leftarrow M[1]M[2] \cdots M[w]$
if $V_c = V_c \oplus 1$ **then**
  return M
else
  return $\perp$
end

Fig. 2. The encryption $\mathcal{E}_K(A, M)$ and decryption $\mathcal{D}_K(A, C, T)$ algorithms of APE.

4 Privacy and Integrity of APE

We prove that APE satisfies privacy under chosen plaintext attacks (CPA) and integrity security up to about $c/2$ bits. Before doing so, in Sect. 4.1 we present the security model, where we formalize the notion of an ideal online function, and where we introduce the CPA and integrity security definitions. Then, privacy is proven in Sect. 4.2 and integrity in Sect. 4.3. The security results in this section assume that the underlying permutation p is ideal. Later, in Sect. 5, we consider the security of APE with block ciphers in the standard model.

4.1 Security Model

Let $\mathrm{Perm}(n)$ be the set of all permutations on n bits. By $\perp$, we denote a function that returns $\perp$ on every input. When writing $x \xleftarrow{\$} \mathbf{X}$ for some finite set $\mathbf{X}$ we mean that x is sampled uniformly from $\mathbf{X}$. To avoid confusion, for $X \in \mathbf{R} \times \mathbf{C}$ we sometimes write $[X]_c := X_c$ to denote the projection of X onto $\mathbf{C}$.

Online functions were first introduced in [4]. We deviate slightly from their approach by explicitly defining our ideal online function in terms of random functions.

Definition 1 (Ideal Online Function). *Let* $g : \mathbf{R}^* \times \mathbf{R}^* \to \mathbf{R}$ *and* $g' : \mathbf{R}^* \times \mathbf{R}^* \to \mathbf{C}$ *be random functions. Then, on input of* (A, M) *with* $w = |M|/r$*, we define* $\$: \mathbf{R}^* \times \mathbf{R}^+ \to \mathbf{R}^+ \times \mathbf{C}$ *as*

$$\$(A, M[1]\|M[2]\|\cdots\|M[w]) = (C[1]\|C[2]\|\cdots\|C[w], T),$$

where

$$\begin{aligned} C[j] &= g(A, M[1]\|\cdots\|M[j]) \text{ for } j = 1, \ldots, w, \\ T &= g'(A, M). \end{aligned}$$

Notice that the above function is actually online: prefixes of outputs remain the same if prefixes of the inputs remain constant. Furthermore, if the associated data in the ideal online function is unique for each invocation of the function, then we achieve full privacy. This is because the inputs to g and g' will then be unique for each invocation, and since g and g' are random functions, we get outputs that are independent and uniformly distributed. By comparing our scheme to the above ideal online function we will automatically achieve the notions of CPA security and integrity from Rogaway and Zhang [30] and Fleischmann et al. [18].

Definition 2. *Let* $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ *denote an AE scheme. The CPA advantage of a distinguisher* D *is defined as*

$$\mathbf{Adv}^{\mathrm{cpa}}_{\Pi}(D) = \left| \begin{array}{r} \Pr\left[p \xleftarrow{\$} \mathrm{Perm}(r+c)\,,\, K \xleftarrow{\$} \mathcal{K} \;:\; D^{\mathcal{E}_K, \perp, p, p^{-1}} = 1\right] - \\ \Pr\left[p \xleftarrow{\$} \mathrm{Perm}(r+c) \;:\; D^{\$, \perp, p, p^{-1}} = 1\right] \end{array} \right|.$$

By $\mathbf{Adv}^{\mathrm{cpa}}_{\Pi}(q, m)$ *we denote the supremum taken over all distinguishers making* q *queries of total length* m *blocks.*

Definition 3. *Let* $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ *denote an AE scheme. The integrity advantage of a distinguisher* D *is defined as*

$$\mathbf{Adv}^{\mathrm{int}}_{\Pi}(D) = \left| \begin{array}{r} \Pr\left[p \xleftarrow{\$} \mathrm{Perm}(r+c)\,,\, K \xleftarrow{\$} \mathcal{K} \;:\; D^{\mathcal{E}_K, \mathcal{D}_K, p, p^{-1}} = 1\right] - \\ \Pr\left[p \xleftarrow{\$} \mathrm{Perm}(r+c)\,,\, K \xleftarrow{\$} \mathcal{K} \;:\; D^{\mathcal{E}_K, \perp, p, p^{-1}} = 1\right] \end{array} \right|.$$

We assume that the distinguisher does not make a decryption query (A, C, T) *if it ever obtained* $(C, T) \leftarrow \mathcal{E}_K(A, M)$ *for some* M*. By* $\mathbf{Adv}^{\mathrm{int}}_{\Pi}(q, m)$ *we denote the supremum taken over all distinguishers making* q *queries of total length* m *blocks.*

4.2 Privacy

In this section, we present a privacy security proof for APE.

Theorem 1. *Let $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be the APE construction. Then,*

$$\mathbf{Adv}^{\mathrm{cpa}}_{\Pi}(q, m) \leq \frac{m^2}{2^{r+c}} + \frac{m(m+1)}{2^c}.$$

Proof. We consider the strongest possible type of distinguishers: let D be any information-theoretic distinguisher which has unbounded computational power and whose complexity is measured solely by the number of queries it makes to its oracles. Without loss of generality, we restrict ourselves to distinguishers that do not ask "trivial" queries, queries to which it knows the answer in advance.

As a first step, we make a PRP-PRF switch [5]: we move from random permutation (p, p^{-1}) to a primitive (f, f^{-1}) defined as follows. This primitive maintains an initially empty list of responses, $\mathcal{F}$, and we denote its domain/range by $\mathsf{dom}(\mathcal{F})/\mathsf{rng}(\mathcal{F})$. Now, on a non-trivial forward query $f(x)$, the response y is randomly drawn from $\mathbf{R} \times \mathbf{C}$. The primitive aborts if y happens to be in $\mathsf{rng}(\mathcal{F})$ already; otherwise, the fresh tuple (x, y) is added to $\mathcal{F}$. Similarly for inverse queries to f^{-1}. Clearly, (p, p^{-1}) and (f, f^{-1}) behave identically as long as the latter does not abort. Given that the distinguisher makes at most q queries of total length m blocks (each block corresponds to a new (f, f^{-1})-query), such an abort happens with probability at most $\binom{m}{2}/2^{r+c} \leq m^2/2^{r+c+1}$. We apply this PRP-PRF switch to both the ideal and the real world, and hence we find

$$\Delta_D(\mathcal{E}_K, \perp, p, p^{-1}; \$, \perp, p, p^{-1}) \leq \frac{m^2}{2^{r+c}} + \Delta_D(\mathcal{E}_K, \perp, f, f^{-1}; \$, \perp, f, f^{-1}). \quad (1)$$

In the remainder, we consider D to have oracle access to one of the two worlds: (F, f, f^{-1}), where $F \in \{\mathcal{E}_K, \$\}$ (without loss of generality we can drop the $\perp$).

If f is called by D then we call this a direct f-query, and similar for direct f^{-1}-queries. A call of f by $\mathcal{E}_K$ (as a result of D calling $\mathcal{E}_K$) is called an indirect f-query. When we do not specify whether an f-query is indirect or direct, we mean that it could be either. Note that indirect queries do not occur in the random world $(\$, f, f^{-1})$. Every indirect f-query has a sequence of associated data blocks and message blocks leading up to it (from the $\mathcal{E}_K$-query calling it); we call this sequence the message chain associated to the indirect f-query.

Let $\mathcal{Q}_i$ denote the set of all prefixes of all queries made by D to its F-oracle before the ith (f, f^{-1})-query, where a query (A, M) results in prefixes $\{A[1], A[1]\|A[2], \ldots, A\|M\}$. Regarding all *direct* queries before the ith query, we denote by X_i^{dir} the set of all capacity values input to f-queries or output of f^{-1}-queries. For example, a direct forward query $y \leftarrow f(x)$ adds $[x]_c$ to X_i^{dir} and a direct inverse query $x \leftarrow f^{-1}(y)$ adds $[x]_c$ to X_i^{dir}. Similarly, by X_i^{ind} we denote the set of all capacity values input to *indirect* f-queries before the ith f-query. We write $X_i = X_i^{\mathrm{dir}} \cup X_i^{\mathrm{ind}}$, and initialize $X_0^{\mathrm{ind}} = \{K\}$.

We define event $\mathsf{E}_i = \mathsf{E}_i^{\text{dir-}X} \cup \mathsf{E}_i^{\text{ind-}X}$, where

$\mathsf{E}_i^{\text{dir-}X}$: direct query $y \leftarrow f(x)$ or $x \leftarrow f^{-1}(y)$ satisfies $[x]_c \in X_i^{\text{ind}} \cup X_i^{\text{ind}} \oplus 1$,

$\mathsf{E}_i^{\text{ind-}X}$: indirect query $f(x)$ with message chain $(A, M) \notin \mathcal{Q}_i$ satisfies $[f(x)]_c \in X_i \cup X_i \oplus 1$.

We furthermore define

$$\hat{\mathsf{E}}_i := \mathsf{E}_i \cap \bigcap_{j=1}^{i-1} \overline{\mathsf{E}_j}, \text{ and } \mathsf{E} := \bigcup_{i=1}^{m} \hat{\mathsf{E}}_i, \tag{2}$$

where $\overline{\mathsf{E}_j}$ is the complement of E_j.

Now, the remainder of the proof is divided as follows. In Lemma 1 we will prove that $(\mathcal{E}_K, f, f^{-1})$ and $(\$, f, f^{-1})$ are indistinguishable as long as E does not occur. From (1) and the fundamental lemma of game playing [5] we find

$$\mathbf{Adv}_{\Pi}^{\text{cpa}}(q, m) \leq \frac{m^2}{2^{r+c}} + \Pr[D^{\mathcal{E}_K, f, f^{-1}} \text{ sets } \mathsf{E}].$$

Then, in Lemma 2, we will prove that $\Pr[D^{\mathcal{E}_K, f, f^{-1}} \text{ sets } \mathsf{E}] \leq \frac{m(m+1)}{2^c}$, which completes the proof. □

Lemma 1. *Given that* E *does not occur,* $(\mathcal{E}_K, f, f^{-1})$ *and* $(\$, f, f^{-1})$ *are indistinguishable.*

Proof. Note that in the ideal world, each direct f-query is new, and is answered with a uniformly randomly drawn response. Now, consider a direct query $f(x)$ in the real world. As the distinguisher does not make trivial queries, it does not coincide with any previous direct query. Additionally, if $[x]_c \in X_i^{\text{ind}} \cup X_i^{\text{ind}} \oplus 1$, where $f(x)$ is the ith f-query, then this would trigger $\mathsf{E}_i^{\text{dir-}X}$, hence we can assume $[x]_c \notin X_i^{\text{ind}} \cup X_i^{\text{ind}} \oplus 1$. This means that the query $f(x)$ is truly new, and its value is independently and uniformly distributed. The same reasoning applies to f^{-1}-queries. Therefore, we only need to consider queries to the big oracle $F \in \{\mathcal{E}_K, \$\}$. Let (A, M) be a query made by the distinguisher. Denote by w the number of blocks of M. Denote the corresponding ciphertext and tag by (C, T).

First consider the case $(A, M[1] \| \cdots \| M[j]) \in \mathcal{Q}_i$ for some $j \in \{0, \ldots, w\}$ and assume j is maximal (we will come back to the case of $(A, *) \notin \mathcal{Q}_i$ later in the proof). Let (A', M') be the corresponding earlier query, so $M[1] \| \cdots \| M[j] = M'[1] \| \cdots \| M'[j]$, and denote its ciphertext and tag by (C', T') and block length by w'. Clearly, in the ideal world $(\$, f, f^{-1})$, we have $C[i] = C'[i]$ for $i = 1, \ldots, j$, but $C[i]$ for $i = j+1, \ldots, w$ and T are uniformly randomly drawn. We will consider how these values are distributed in the real world $(\mathcal{E}_K, f, f^{-1})$. We first consider the general case $j < w$, the case $j = w$ is discussed afterwards.

1. $C[1], \ldots, C[j]$. Also in the real world, these values equal $C'[1], \ldots, C'[j]$, which follows clearly from the specification of $\mathcal{E}_K$. Note that in particular, the state value V equals V' after the jth round.

2. $C[j+1]$. We make a distinction between $j > 0$ and $j = 0$, and start with the former case. Write the indirect query corresponding to the jth round as $f(x)$. The input of the $(j+1)$th query will be $f(x) \oplus (M[j+1], 0)$. Note that $(A, M' \| M[j+1]) \notin \mathcal{Q}$, as this would contradict the fact that j is maximal. Now, assume this $(j+1)$th query has already been made before, i.e. $[f(x)]_c \in X^{\text{dir}} \cup X^{\text{ind}}$. This may be the case (it may even date from before the evaluation of (A, M')), but at this particular time the capacity part $[f(x)]_c$ did not hit any element from $X^{\text{dir}} \cup X^{\text{ind}}$ (otherwise it would have triggered $\mathsf{E}^{\text{ind-}X}$). After this query has been made, there has not been any newer indirect query or any newer direct query whose capacity part hit $[f(x)]_c$ (both cases would have triggered $\mathsf{E}^{\text{dir-}X} \cup \mathsf{E}^{\text{ind-}X}$). Thus, the query corresponding to the $(j+1)$th round is generated independently and uniformly at random.

   Now, in the case $j = 0$, V denotes the state right after the hashing of A ($V = (0, K)$ if $A = \varnothing$). The same story as before applies with the difference that now the input to the $(j+1)$th query is $V \oplus (M[j+1], 1)$. Here we use that by $\overline{\mathsf{E}}$, no other query hit $X_j^{\text{ind}} \oplus 1$ (for direct queries) or $X_j \oplus 1$ (for indirect queries) in the meanwhile, and that X^{ind} is initialized with $\{K\}$.
3. $C[j+2], \ldots, C[w]$. By the above argument, the indirect query made in the $(j+1)$th round of (A, M), say $f(x)$ for the sake of presentation, is responded with a uniformly random answer. This query would have triggered $\mathsf{E}^{\text{ind-}X}$ if $[f(x)]_c \in X_i$. Therefore, we know that also the $(j+2)$th query is truly random and so is $C[j+2]$. The same reasoning applies up to $C[w]$.
4. T. The same reasoning applies: the previous query is responded with a truly random answer $f(x)$. Consequently $T = [f(x)]_c \oplus K$ is random too.

A special treatment is needed for $j = w$. In this case, $C[1], \ldots, C[w]$ equals $C'[1], \ldots, C'[w]$ by construction, but the query producing T is not new. Yet, the distinguisher never made that query itself by virtue of $\overline{\mathsf{E}^{\text{dir-}X}}$, so it never learnt $T \oplus K$. Besides, due to the absence of indirect capacity collisions, $\overline{\mathsf{E}^{\text{ind-}X}}$, every f-query will produce a tag at most once. This means that T will look uniformly random to the distinguisher, as it would look if it were produced by \$.

Finally, we consider the case $(A, *) \notin \mathcal{Q}_i$, hence this is the first time a query for this particular associated data A is made. Then, the above reasoning carries over for $j = 0$ with the simplification that if $A \neq \varnothing$, the value V_c right after the hashing of A can be considered new. □

Lemma 2. $\Pr[D^{\mathcal{E}_K, f, f^{-1}} \textit{ sets } \mathsf{E}] \leq \dfrac{m(m+1)}{2^c}$.

Proof. Inspired by (2), we start bounding $\Pr[\mathsf{E}_i \cap \bigcap_{j=1}^{i-1} \overline{\mathsf{E}_j}]$ for $i \in \{1, \ldots, m\}$. Clearly,

$$\Pr[\mathsf{E}_i \cap \textstyle\bigcap_{j=1}^{i-1} \overline{\mathsf{E}_j}] \leq \Pr[\mathsf{E}_i \mid \textstyle\bigcap_{j=1}^{i-1} \overline{\mathsf{E}_j}].$$

Therefore, we assume $\bigcap_{j=1}^{i-1} \overline{\mathsf{E}_j}$ and consider the probability the ith query triggers E_i.

If the ith query is a direct (forward or inverse) query, it triggers $\mathsf{E}_i^{\text{dir-}X}$ if the distinguisher guesses (in case of forward) or hits (in case of inverse) a capacity part in $X_i^{\text{ind}} \cup X_i^{\text{ind}} \oplus 1$, which happens with probability at most $2|X_i^{\text{ind}}|/2^c$. On the other hand, if the ith query is a new indirect query (i.e. for which $(A, M) \notin \mathcal{Q}_i$) it triggers $\mathsf{E}_i^{\text{ind-}X}$ if $[f(x)]_c \in X_i \cup X_i \oplus 1$. This occurs with probability at most $2|X_i|/2^c$.

As the query is either direct or indirect, we could take the maximum of both values. Given that $|X_i^{\text{ind}}| \leq |X_i| \leq i$, we find:

$$\Pr[\mathsf{E}_i \mid \bigcap_{j=1}^{i-1} \overline{\mathsf{E}_j}] \leq \frac{2i}{2^c}.$$

The result is now obtained by summing over $i = 1, \ldots, m$ (as in (2)). □

4.3 Integrity

In this section, we present an integrity security proof for APE.

Theorem 2. *Let $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be the APE construction. Then,*

$$\mathbf{Adv}_{\Pi}^{\text{int}}(q, m) \leq \frac{m^2}{2^{r+c}} + \frac{2m(m+1)}{2^c}.$$

Proof. The basic idea of the proof is the same as for Theorem 1. Again, let D be any information-theoretic distinguisher. By the PRP-PRF switch, we find

$$\Delta_D(\mathcal{E}_K, \mathcal{D}_K, p, p^{-1}; \mathcal{E}_K, \perp, p, p^{-1}) \leq \frac{m^2}{2^{r+c}} + \Delta_D(\mathcal{E}_K, \mathcal{D}_K, f, f^{-1}; \mathcal{E}_K, \perp, f, f^{-1}). \tag{3}$$

We consider D to have oracle access to one of the two worlds: $(\mathcal{E}_K, F, f, f^{-1})$, where $F \in \{\mathcal{D}_K, \perp\}$.

We use the same notation as in Theorem 1, but slightly more involved definitions are required and we re-introduce them. If f is called by D then we call this a direct f-query, and similar for direct f^{-1}-queries. A call of f by $\mathcal{E}_K$ or $\mathcal{D}_K$ (as a result of D calling them) is called an indirect f-query, and similar for indirect f^{-1}-queries (via $\mathcal{D}_K$). Every indirect f-query has a sequence of associated data blocks and/or message blocks leading up to it (from the $\mathcal{E}_K$- or $\mathcal{D}_K$-query calling it); we call this sequence the message chain associated to the indirect f-query. Every indirect f^{-1}-query has a tag and a sequence of ciphertext blocks leading up to it, and we call this sequence the associated ciphertext chain.

Let $\mathcal{Q}_i$ denote the set of all prefixes of all queries made by D to its $\mathcal{E}_K$-oracle before the ith (f, f^{-1})-query, where an $\mathcal{E}_K$-query (A, M) results in prefixes $\{A[1], A[1]\|A[2], \ldots, A\|M\}$. In this set, we also include $\{A[1], \ldots, A\}$ for an F-query (A, C, T). Let $\mathcal{Q}_i^{-1}$ denote the set of all suffixes of all queries made by D to its F-oracle before the ith query, where an F-query (A, C, T) results in suffixes $\{C[w]\|T, C[w-1]\|C[w]\|T, \ldots, C\|T\}$. (The tag value T is included here for technical reasons.) Regarding all *direct* queries before the ith query, we denote

by X_i^{dir} the set of all capacity values input to f-queries or output of f^{-1}-queries, and by Y_i^{dir} the set of all capacity values input to f^{-1}-queries or output of f-queries. For example, a direct forward query $y \leftarrow f(x)$ adds $[x]_c$ to X_i^{dir} and $[y]_c$ to Y_i^{dir}. The sets X_i^{ind} and Y_i^{ind} are defined similarly. We write $X_i = X_i^{\text{dir}} \cup X_i^{\text{ind}}$ and $Y_i = Y_i^{\text{dir}} \cup Y_i^{\text{ind}}$, and initialize $X_0^{\text{ind}} = Y_0^{\text{ind}} = \{K\}$.

We define event $\mathsf{E}_i = \mathsf{E}_i^{\text{dir-}X} \cup \mathsf{E}_i^{\text{ind-}X} \cup \mathsf{E}_i^{\text{dir-}Y} \cup \mathsf{E}_i^{\text{ind-}Y}$, where $\mathsf{E}_i^{\text{dir-}X}$ and $\mathsf{E}_i^{\text{ind-}X}$ are as in the proof of Theorem 1 with the renewed definitions of the sets, and where

$$\begin{aligned}
\mathsf{E}_i^{\text{dir-}Y} &: \text{ direct query } y \leftarrow f(x) \text{ or } x \leftarrow f^{-1}(y) \text{ satisfies } [y]_c \in Y_i^{\text{ind}} \cup Y_i^{\text{ind}} \oplus 1, \\
\mathsf{E}_i^{\text{ind-}Y} &: \text{ indirect query } f^{-1}(y) \text{ with ciphertext chain } (C,T) \notin \mathcal{Q}_i^{-1} \text{ satisfies} \\
&\qquad [f^{-1}(y)]_c \in Y_i \cup Y_i \oplus 1 \text{ or } [y]_c \in Y_i^{\text{dir}} \oplus K.
\end{aligned}$$

Definitions $\hat{\mathsf{E}}_i$ and E are as before. The latter condition of $\mathsf{E}_i^{\text{ind-}Y}$, $[y]_c \in Y_i^{\text{dir}} \oplus K$, covers the case the distinguisher obtains the key by making a direct inverse query and a $\mathcal{D}_K$-query.

Now, the remainder of the proof is divided as follows. In Lemma 3 we will prove that $(\mathcal{E}_K, \mathcal{D}_K, f, f^{-1})$ and $(\mathcal{E}_K, \perp, f, f^{-1})$ are indistinguishable as long as E does not occur. From (3) and the fundamental lemma of game playing [5] we find

$$\mathbf{Adv}_{\Pi}^{\text{cpa}}(q,m) \leq \frac{m^2}{2^{r+c}} + \Pr[D^{\mathcal{E}_K, \mathcal{D}_K, f, f^{-1}} \text{ sets } \mathsf{E}].$$

Then, in Lemma 4, we will prove that $\Pr[D^{\mathcal{E}_K, \mathcal{D}_K, f, f^{-1}} \text{ sets } \mathsf{E}] \leq \dfrac{2m(m+1)}{2^c}$, which completes the proof. □

Lemma 3. *Given that* E *does not occur,* $(\mathcal{E}_K, \mathcal{D}_K, f, f^{-1})$ *and* $(\mathcal{E}_K, \perp, f, f^{-1})$ *are indistinguishable.*

Proof. The proof is given in the full version [1]. It is similar to the proof of Lemma 1. □

Lemma 4. $\Pr[D^{\mathcal{E}_K, \mathcal{D}_K, f, f^{-1}} \text{ sets } \mathsf{E}] \leq \dfrac{2m(m+1)}{2^c}$.

Proof. The proof is given in the full version [1]. It is similar to the proof of Lemma 2. □

5 Standard Model Security of APE

As is conventionally done for existing permutation-based designs, our proof for APE assumes that the underlying permutation is ideal. By considering a generalized version of APE, we now provide a standard model security argument for our scheme. Inspired by [16], we note that APE can also be described as a block cipher based design: we drop the key additions at the beginning and end, and replace the permutations with a keyed block cipher E_K defined by $E_K := KpK := \oplus_{0\|K} \circ p \circ \oplus_{0\|K}$. (One can view E_K as the Even-Mansour [17]

block cipher with partial key addition.) We remark that this is, indeed, an equivalent description of APE if the block cipher is replaced by KpK. In our notation we denote APE as described and based on some block cipher E by $\Pi' = (\mathcal{K}, \mathcal{E}^E, \mathcal{D}^E)$. We first give the privacy and integrity definitions in the standard model and then show that our results of Theorems 1 and 2 easily translate to a standard model security of Π'.

Definition 4. *Let E be a block cipher, and let $\Pi' = (\mathcal{K}, \mathcal{E}^E, \mathcal{D}^E)$ denote an AE scheme. The CPA advantage of a distinguisher D is defined as*

$$\mathbf{Adv}^{\overline{\text{cpa}}}_{\Pi'}(D) = \left|\Pr\left[K \xleftarrow{\$} \mathcal{K} \ : \ D^{\mathcal{E}^E_K} = 1\right] - \Pr\left[D^{\$} = 1\right]\right| .$$

By $\mathbf{Adv}^{\overline{\text{cpa}}}_{\Pi'}(t, q, m)$ we denote the supremum taken over all distinguishers running in time t and making q queries of total length m blocks. Alternatively, we write $\mathbf{Adv}^{\overline{\text{cpa}}}_{\Pi'}(t, q, m) = \Delta^t_{q,m}(\mathcal{E}^E_K; \$)$ as in Definition 2 with the inclusion of t.

Definition 5. *Let E be a block cipher, and let $\Pi' = (\mathcal{K}, \mathcal{E}^E, \mathcal{D}^E)$ denote an AE scheme. The integrity advantage of a distinguisher D is defined as*

$$\mathbf{Adv}^{\overline{\text{int}}}_{\Pi'}(D) = \left|\Pr\left[K \xleftarrow{\$} \mathcal{K} \ : \ D^{\mathcal{E}^E_K, \mathcal{D}^E_K} = 1\right] - \Pr\left[K \xleftarrow{\$} \mathcal{K} \ : \ D^{\mathcal{E}^E_K, \perp} = 1\right]\right| .$$

By $\mathbf{Adv}^{\overline{\text{int}}}_{\Pi'}(t, q, m)$ we denote the supremum taken over all distinguishers running in time t and making q queries of total length m blocks. Alternatively, we write $\mathbf{Adv}^{\overline{\text{int}}}_{\Pi'}(t, q, m) = \Delta^t_{q,m}(\mathcal{E}^E_K, \mathcal{D}^E_K; \mathcal{E}^E_K, \perp)$ as in Definition 3 with the inclusion of t.

In both definitions we refer to the rate of Π', the number of block cipher calls per message block, as ρ. Furthermore, we need the notion of strong pseudorandom permutation, or prp$\pm$1, security of E.

Definition 6. *Let E be a block cipher. The* prp$\pm$1 *advantage of a distinguisher $\mathcal{D}$ is defined as*

$$\mathbf{Adv}^{\text{prp}\pm1}_{E}(D) = \left| \begin{array}{l} \Pr\left[K \xleftarrow{\$} \mathcal{K} \ : \ D^{E_K, E_K^{-1}} = 1\right] - \\ \qquad \Pr\left[\pi \xleftarrow{\$} \text{Perm}(r + c) \ : \ D^{\pi, \pi^{-1}} = 1\right] \end{array} \right| .$$

By $\mathbf{Adv}^{\text{prp}\pm1}_{E}(t, q)$ we denote the maximum advantage taken over all distinguishers that run in time t and make q queries.

We demonstrate that the standard model security of APE with block ciphers is implied by the results of Sect. 4. To this end we introduce two propositions, one with respect to the integrity and one with respect to the privacy of Π'.

Proposition 1. *Let E be a block cipher.*

$$\mathbf{Adv}^{\overline{\text{int}}}_{\Pi'}(t, q, m) \leq \frac{m^2}{2^{r+c}} + \frac{2m(m+1)}{2^c} + 2\mathbf{Adv}^{\text{prp}\pm1}_{E}(t', \rho m).$$

Proof. Let $K \xleftarrow{\$} \mathcal{K}$. Let E be a publicly available block cipher and $\pi, p \xleftarrow{\$} \mathrm{Perm}(r+c)$ be random permutations. We first switch from E to random π:

$$\begin{aligned}\Delta^t_{q,m}(\mathcal{E}^E_K, \mathcal{D}^E_K; \mathcal{E}^E_K, \perp) &\leq \Delta^t_{q,m}(\mathcal{E}^E_K, \mathcal{D}^E_K; \mathcal{E}^\pi_K, \mathcal{D}^\pi_K) + \Delta^t_{q,m}(\mathcal{E}^\pi_K, \mathcal{D}^\pi_K; \mathcal{E}^\pi_K, \perp) \\ &\quad + \Delta^t_{q,m}(\mathcal{E}^\pi_K, \perp; \mathcal{E}^E_K, \perp) \\ &\leq \Delta^t_{q,m}(\mathcal{E}^\pi_K, \mathcal{D}^\pi_K; \mathcal{E}^\pi_K, \perp) + 2\mathbf{Adv}^{\mathrm{prp}\pm 1}_E(t', \rho m),\end{aligned}$$

where $t' \approx t$. As π is a random permutation, we could give the distinguisher unlimited time (effectively considering information-theoretic distinguishers), and the bound simplifies to:

$$\Delta^t_{q,m}(\mathcal{E}^E_K, \mathcal{D}^E_K; \mathcal{E}^E_K, \perp) \leq \Delta_{q,m}(\mathcal{E}^\pi_K, \mathcal{D}^\pi_K; \mathcal{E}^\pi_K, \perp) + 2\mathbf{Adv}^{\mathrm{prp}\pm 1}_E(t', \rho m).$$

For the remaining Δ-term:

$$\begin{aligned}\Delta_{q,m}(\mathcal{E}^\pi_K, \mathcal{D}^\pi_K; \mathcal{E}^\pi_K, \perp) &\leq \Delta_{q,m}(\mathcal{E}^\pi_K, \mathcal{D}^\pi_K; \mathcal{E}^{KpK}_K, \mathcal{D}^{KpK}_K) \\ &\quad + \Delta_{q,m}(\mathcal{E}^{KpK}_K, \mathcal{D}^{KpK}_K; \mathcal{E}^{KpK}_K, \perp) + \Delta_{q,m}(\mathcal{E}^{KpK}_K, \perp; \mathcal{E}^\pi_K, \perp) \\ &\leq 0 + \Delta_{q,m}(\mathcal{E}^{KpK}_K, \mathcal{D}^{KpK}_K; \mathcal{E}^{KpK}_K, \perp) + 0,\end{aligned}$$

where we use that π and KpK are identically distributed as π and p are random permutations and K is random and unknown. The middle term equals $\mathbf{Adv}^{\mathrm{int}}_{\Pi}(q, m)$ with the difference that the distinguisher cannot access p. A distinguisher would only benefit from such additional access, thus:

$$\Delta_{q,m}(\mathcal{E}^{KpK}_K, \mathcal{D}^{KpK}_K; \mathcal{E}^{KpK}_K, \perp) \leq \mathbf{Adv}^{\mathrm{int}}_{\Pi}(q, m),$$

which is bounded in Theorem 2. This completes the proof. □

Proposition 2. *Let E be a block cipher.*

$$\mathbf{Adv}^{\overline{\mathrm{cpa}}}_{\Pi'}(t, q, m) \leq \frac{m^2}{2^{r+c}} + \frac{1m(m+1)}{2^c} + \mathbf{Adv}^{\mathrm{prp}\pm 1}_E(t', \rho m).$$

Proof. The proof is a straightforward simplification of the proof of Proposition 1, and therefore omitted. □

6 Hardware Implementation

We implement APE with the permutations of PHOTON [19], QUARK [3], and SPONGENT [11]. The results are given in Table 1. We use these permutations without any modifications to investigate the hardware performance of APE. As the designs of PHOTON, QUARK, and SPONGENT follow the hermetic sponge strategy [8], the underlying permutations are assumed to be indistinguishable from random permutations. This assumption is necessary in order to achieve the claimed privacy (Theorem 1) and integrity (Theorem 2) security bounds. Since APE is designed for constrained devices, we focus on a security level of 80 and 128 bits, which correspond to a capacity of 160 or 256 bits, respectively. One exception is APE based

on Quark: since Quark is not equipped with a version for 128 bits of security we resort to a permutation that offers 112 bits of security. The versions of Photon and Spongent with 80 bits of security are implemented with a 4-bit serialization, which means that we implement one 4-bit S-box. For the versions with higher security, we use an 8-bit serialization which requires two 4-bit S-boxes for Spongent and one 8-bit S-box for Photon. Unlike Photon and Spongent, the round permutation of Quark is based on Feedback Shift Registers (FSRs). Hence it is possible to update one bit per clock cycle, and in our implementation we choose to do so for area efficiency.

As APE decrypts in reverse order and requires the inverse permutation, for each of the algorithms (Photon, Quark, and Spongent) we have provided both an encryption-only implementation and an implementation with encryption and decryption. In brief, we have implemented APE as follows. The initial state is XORed with the first data inserted nibble by nibble (or byte by byte, or bit by bit). After each permutation evaluation, the resulting ciphertext is output as the new data is inserted in the same clock cycle. At the end of the iteration, the entire state is output and the capacity part is XORed with the key to generate the tag. Similarly, for decryption, the first state corresponds to the ciphertext concatenated with an XOR of the key and the tag, and at the end authenticity is verified.

For the hardware implementation results in Table 1, we used ModelSim to verify the functionality of the designs and Synopsys Design Vision D-2010.03-SP4 for synthesis. We used Faraday Standard Cell Library based on UMC 0.18μm and open-cell 45nm NANGATE [25] library. As main observations, we see that APE with an encryption and decryption mode can be implemented with less than 1.9 kGE and 2.8 kGE for 80-bit and 128-bit security respectively.

When implemented with the same permutation, encryption-only implementations of SpongeWrap and APE will have similar implementation figures. This is because in both constructions, the processing of every message block requires one XOR with the rate part and one permutation function call. We recall that the crucial difference between SpongeWrap and APE is that APE provides nonce misuse resistance. For the decryption operation, the cost of misuse resistance for APE is that the backwards permutation must be implemented as well.

As shown in Table 1, the overhead of implementing both p and p^{-1} is at most 283 GE on our 45 nm implementation. For devices without non-volatile memory, this overhead is very low compared to the cost of providing a hardware source of randomness to generate nonces.

Note that the permutation-based schemes are implemented on 180 nm and 45 nm CMOS, whereas for the block cipher based schemes, lightweight implementations on 65 nm CMOS are provided. Therefore, we cannot compare these implementations directly. Also note that the clock frequencies of the implementations differ, which lead to different throughput figures. However, it seems that APE and ALE have similar performance figures and APE is smaller than ASC-1 A, ASC-1 B and AES-CCM.

Table 1. APE is implemented using the PHOTON, QUARK, and SPONGENT permutations. For each algorithm, we provide an encryption-only implementation, as well as one that does both encryption and decryption (denoted as "e/d"). The area figures depend on the library that we have used: Area A refers to UMC 180 nm, Area B refers to NANGATE 45 nm. Our overview also includes lightweight implementations of the authenticated encryption schemes ALE [12], ASC-1 [20], and AES-CCM [31]. We remark that the clock frequency of the APE implementations is 100 kHz, compared to 20 MHz for the other ciphers.

APE on CMOS process @ 100 kHz, A: UMC 180 nm, B: NANGATE 45 nm						
Design	Security (bits)	Rate (bits)	Latency (cycles)	Throughput (kbps)	Area A (GE)	Area B (GE)
PHOTON-196	80	36	1248	2.9	1398	1309
PHOTON-196 e/d	80	36	1297	2.8	1634	1536
QUARK-176	80	16	880	1.81	1694	1606
QUARK-176 e/d	80	16	880	1.81	1871	1773
SPONGENT-176	80	16	4050	0.4	1423	1598
SPONGENT-176 e/d	80	16	4094	0.4	1868	1838
PHOTON-288	128	32	924	3.45	2154	2104
PHOTON-288 e/d	128	32	960	3.33	2449	2327
QUARK-256	112	32	1270	2.51	2286	2228
QUARK-256 e/d	112	32	1270	2.51	2470	2331
SPONGENT-272	128	16	4480	0.4	2105	2378
SPONGENT-272 e/d	128	16	4652	0.3	2781	2661

Other AE schemes on ST 65 nm CMOS LP-HVT process @ 20 MHz [12]				
Design	Security (bits)	Latency (cycles)	Throughput (kbps)	Area (GE)
ALE	128	105	121.9	2579
ALE e/d	128	105	121.9	2700
ASC-1 A	128	370	34.59	4793
ASC-1 A e/d	128	370	34.59	4964
ASC-1 B	128	235	54.47	5517
ASC-1 B e/d	128	235	54.47	5632
AES-CCM	128	452	28.32	3472
AES-CCM e/d	128	452	28.32	3765

7 Conclusions

In this paper, we introduced APE, the first misuse resistant permutation-based AE scheme. We proved that APE provides security and integrity up to the birthday bound of the capacity, in the ideal permutation model. We show that the security of APE in the ideal permutation model implies the security of APE with block ciphers in the standard model. This not only ensures security of APE when its underlying primitive is considered an ideal permutation, but also allows to employ it with any secure block cipher of specific form. To achieve misuse resistance, the decryption of APE as a permutation-based construction uses the inverse permutation to decrypt in a backwards manner. The advantage of having backwards decryption is that if the tag or last ciphertext block is missing, then decryption is impossible. Our hardware implementations of APE show that it is

well-suited for lightweight applications. In fact, using any of the permutations of Quark, Photon, and Spongent, less than 1.9 kGE (80-bit security) and less than 2.8 kGE (128-bit security) is required for an implementation of APE that supports both encryption and decryption. Due to its resistance against nonce reuse and its low area requirements in hardware, APE is suitable for environments where it is prohibitively expensive to require non-volatile memory or a hardware source of randomness.

Acknowledgments. We would like to thank the various anonymous reviewers for providing useful comments. Furthermore, we would like to thank Reza Reyhanitabar, Ivan Tjuawinata, Anthony Van Herrewege, Ingrid Verbauwhede, and Hongjun Wu for various suggestions to improve the quality of the text. This work was supported in part by the Research Council KU Leuven: GOA TENSE (GOA/11/007). In addition, this work was supported by the Research Fund KU Leuven, OT/13/071. Elena Andreeva and Nicky Mouha are supported by Postdoctoral Fellowships from the Flemish Research Foundation (FWO-Vlaanderen). Atul Luykx and Bart Mennink are supported by Ph.D. Fellowships from the Institute for the Promotion of Innovation through Science and Technology in Flanders (IWT-Vlaanderen). Begül Bilgin is partially supported by the FWO project G0B4213N.

References

1. Andreeva, E., Bilgin, B., Bogdanov, A., Luykx, A., Mennink, B., Mouha, N., Yasuda, K.: APE: Authenticated Permutation-Based Encryption for Lightweight Cryptography. Cryptology ePrint Archive, Report 2013/791 (2013) (full version of this paper)
2. Aumasson, J.-P., Henzen, L., Meier, W., Naya-Plasencia, M.: Quark: A Lightweight Hash. In: Mangard, S., Standaert, F.-X. (eds.) CHES 2010. LNCS, vol. 6225, pp. 1–15. Springer, Heidelberg (2010)
3. Aumasson, J.P., Henzen, L., Meier, W., Naya-Plasencia, M.: Quark: A Lightweight Hash. J. Cryptol. **26**(2), 313–339 (2013)
4. Bellare, M., Boldyreva, A., Knudsen, L.R., Namprempre, C.: On-Line Ciphers and the Hash-CBC Constructions. J. Cryptol. **25**(4), 640–679 (2012)
5. Bellare, M., Rogaway, P.: The Security of Triple Encryption and a Framework for Code-Based Game-Playing Proofs. In: Vaudenay, S. (ed.) EUROCRYPT 2006. LNCS, vol. 4004, pp. 409–426. Springer, Heidelberg (2006)
6. Berger, T.P., D'Hayer, J., Marquet, K., Minier, M., Thomas, G.: The GLUON Family: A Lightweight Hash Function Family Based on FCSRs. In: Mitrokotsa, A., Vaudenay, S. (eds.) AFRICACRYPT 2012. LNCS, vol. 7374, pp. 306–323. Springer, Heidelberg (2012)
7. Bertoni, G., Daemen, J., Peeters, M., Van Assche, G.: Permutation-Based Encryption, Authentication and Authenticated Encryption, Directions in Authenticated Ciphers, pp. 159–170 (July 2012). http://www.hyperelliptic.org/djb/diac/record.pdf
8. Bertoni, G., Daemen, J., Peeters, M., Van Assche, G.: Cryptographic Sponge Functions. http://sponge.noekeon.org/CSF-0.1.pdf
9. Bertoni, G., Daemen, J., Peeters, M., Van Assche, G.: Sponge Functions. In: ECRYPT Hash Function Workshop (May 2007)

10. Bertoni, G., Daemen, J., Peeters, M., Van Assche, G.: Duplexing the Sponge: Single-Pass Authenticated Encryption and Other Applications. In: Miri, A., Vaudenay, S. (eds.) SAC 2011. LNCS, vol. 7118, pp. 320–337. Springer, Heidelberg (2012)
11. Bogdanov, A., Knežević, M., Leander, G., Toz, D., Varıcı, K., Verbauwhede, I.: SPONGENT: A Lightweight Hash Function. In: Preneel, B., Takagi, T. (eds.) CHES 2011. LNCS, vol. 6917, pp. 312–325. Springer, Heidelberg (2011)
12. Bogdanov, A., Mendel, F., Regazzoni, F., Rijmen, V., Tischhauser, E.: ALE: AES-Based Lightweight Authenticated Encryption. In: Moriai, S. (ed.) FSE 2013. LNCS, vol. 8424, pp. 447–466. Springer, Heidelberg (2014)
13. Borisov, N., Goldberg, I., Wagner, D.: Intercepting Mobile Communications: The Insecurity of 802.11. In: Rose, C. (ed.) MOBICOM, pp. 180–189. ACM (2001)
14. CAESAR: Competition for Authenticated Encryption: Security, Applicability, and Robustness (April 2013)
15. Cantero, H.M., Peter, S., Bushing, S.: Console Hacking 2010 - PS3 Epic Fail. In: 27th Chaos Communication Congress, December 2010
16. Chang, D., Dworkin, M., Hong, S., Kelsey, J., Nandi, M.: A Keyed Sponge Construction with Pseudorandomness in the Standard Model. In: The Third SHA-3 Candidate Conference, March 2012
17. Even, S., Mansour, Y.: A Construction of a Cipher from a Single Pseudorandom Permutation. J. Cryptol. **10**(3), 151–162 (1997)
18. Fleischmann, E., Forler, C., Lucks, S.: McOE: A Family of Almost Foolproof On-Line Authenticated Encryption Schemes. In: Canteaut, A. (ed.) FSE 2012. LNCS, vol. 7549, pp. 196–215. Springer, Heidelberg (2012)
19. Guo, J., Peyrin, T., Poschmann, A.: The PHOTON Family of Lightweight Hash Functions. In: Rogaway, P. (ed.) CRYPTO 2011. LNCS, vol. 6841, pp. 222–239. Springer, Heidelberg (2011)
20. Jakimoski, G., Khajuria, S.: ASC-1: An Authenticated Encryption Stream Cipher. In: Miri, A., Vaudenay, S. (eds.) SAC 2011. LNCS, vol. 7118, pp. 356–372. Springer, Heidelberg (2012)
21. Khovratovich, D.: Key Wrapping with a Fixed Permutation. Cryptology ePrint Archive, Report 2013/145 (2013)
22. Kohno, T.: Attacking and Repairing the WinZip Encryption Scheme. In: Atluri, V., Pfitzmann, B., McDaniel, P.D. (eds.) ACM Conference on Computer and Communications Security, pp. 72–81. ACM (2004)
23. Lenstra, A.K., Hughes, J.P., Augier, M., Bos, J.W., Kleinjung, T., Wachter, C.: Public Keys. In: Safavi-Naini, R., Canetti, R. (eds.) CRYPTO 2012. LNCS, vol. 7417, pp. 626–642. Springer, Heidelberg (2012)
24. McGrew, D.A., Viega, J.: The Security and Performance of the Galois/Counter Mode (GCM) of Operation. In: Canteaut, A., Viswanathan, K. (eds.) INDOCRYPT 2004. LNCS, vol. 3348, pp. 343–355. Springer, Heidelberg (2004)
25. NANGATE: The NanGate 45nm Open Cell Library. http://www.nangate.com
26. Rogaway, P.: Authenticated-Encryption with Associated-Data. In: Atluri, V. (ed.) ACM Conference on Computer and Communications Security 2002, pp. 98–107. ACM (2002)
27. Rogaway, P.: Nonce-Based Symmetric Encryption. In: Roy, B., Meier, W. (eds.) FSE 2004. LNCS, vol. 3017, pp. 348–359. Springer, Heidelberg (2004)
28. Rogaway, P., Bellare, M., Black, J.: OCB: A Block-Cipher Mode of Operation for Efficient Authenticated Encryption. ACM Trans. Inf. Syst. Secur. **6**(3), 365–403 (2003)

29. Rogaway, P., Shrimpton, T.: A Provable-Security Treatment of the Key-Wrap Problem. In: Vaudenay, S. (ed.) EUROCRYPT 2006. LNCS, vol. 4004, pp. 373–390. Springer, Heidelberg (2006)
30. Rogaway, P., Zhang, H.: Online Ciphers from Tweakable Blockciphers. In: Kiayias, A. (ed.) CT-RSA 2011. LNCS, vol. 6558, pp. 237–249. Springer, Heidelberg (2011)
31. Whiting, D., Housley, R., Ferguson, N.: Counter with CBC-MAC (CCM). Request For Comments 3610 (2003)
32. Wu, H.: The Misuse of RC4 in Microsoft Word and Excel. Cryptology ePrint Archive, Report 2005/007 (2005)

A APE for Fractional Data

The APE description of Sect. 3 should only be used when the application can guarantee that the length of the plaintext and the associated data is always a multiple of the block size r. In this section, we explain how to adjust APE to handle fractional plaintext and associated data. This is done by applying '10*'-padding to all plaintext and associate data (fractional or not).

The extension of APE to fractional associated data is given in Fig. 3, and to fractional messages in Fig. 4. We elaborate on the extension for fractional messages (the extension for fractional associated data being similar). Split a message M into r-bit blocks, where the last block $M[w]$ is possibly incomplete. We distinguish among three cases:

- $|M[w]| \leq r - 1$ and $w = 1$. The procedure can be seen in the top part of Fig. 4. Note that the corresponding ciphertext will be r bits. This is required for decryption to be possible;
- $|M[w]| \leq r - 1$ and $w \geq 2$. The procedure is depicted in the bottom part of Fig. 4. Note that the ciphertext $C[w-1]$ is of size equal to $M[w]$. The reason we opt for this design property is the following: despite $M[w]$ being smaller than r bits, we require its corresponding ciphertext to be r bits for decryption to be possible. As a toll, the extended APE generates ciphertext $C[w-1]$ to be of size equal to $M[w]$;
- $|M[w]| = r$. In this special case where M consists of integral message blocks, we nevertheless need a padding. However, instead of occupying an extra message block for this, the '10*'-padding spills over into the capacity. This can be seen as an XOR of $10 \cdots 00$ into the capacity part of the state. We recall the reader of the fact that the $\oplus 1$ in the beginning of the function is a shorthand notation for $\oplus 00 \cdots 01$, and hence, these values do not cancel each other out.

The adjustments have no influence on the decryption algorithm $\mathcal{D}$, except if $|M| \leq r$ for which a slightly more elaborate function is needed. Note that the spilling of the padding in case $|M[w]| = r$ causes security to degrade by half a bit: intuitively, APE is left with a capacity of $c' = c - 1$ bits. We have opted for this degrading over an efficiency loss due to an additional round.

The proofs of security of APE with fractional data can be found in the full version of this paper [1].

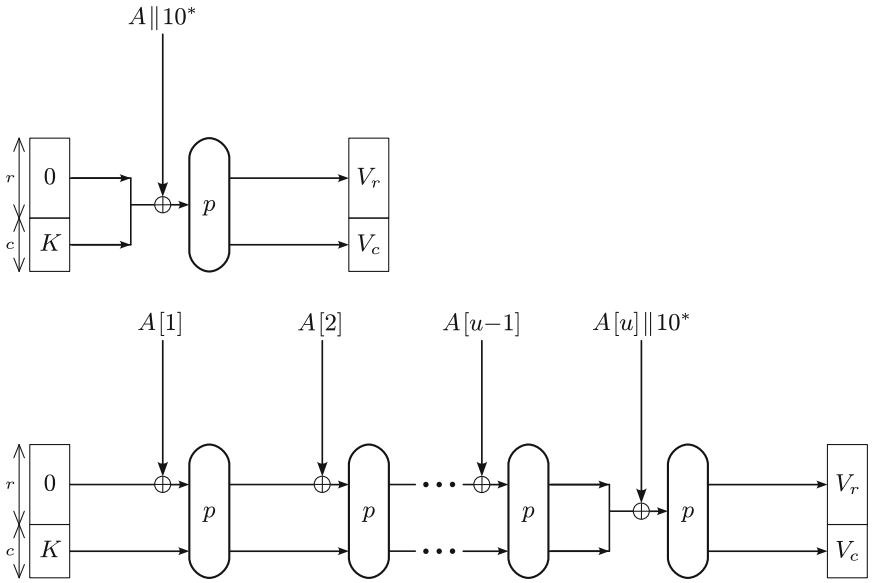


Fig. 3. A generalization of APE that can handle fractional associated data blocks.

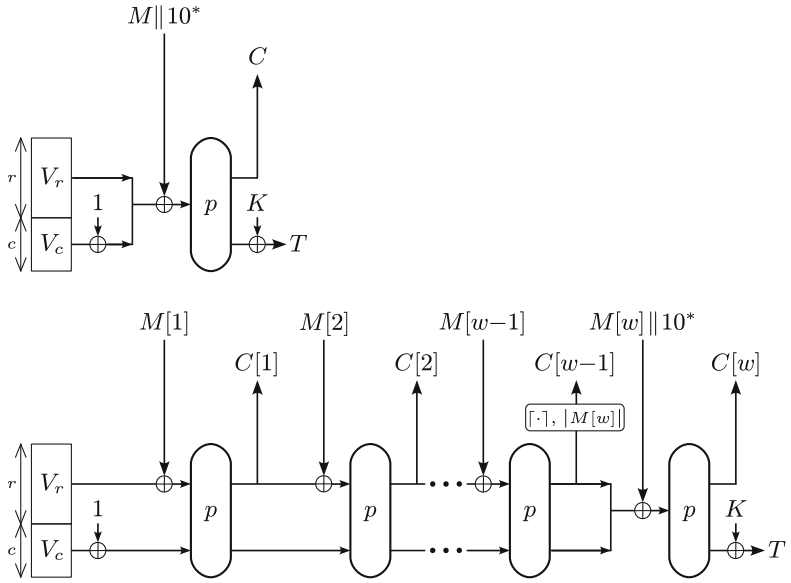


Fig. 4. A generalization of APE that can handle fractional message blocks.

COBRA: A Parallelizable Authenticated Online Cipher Without Block Cipher Inverse

Elena Andreeva[1,2], Atul Luykx[1,2(✉)], Bart Mennink[1,2], and Kan Yasuda[1,3]

[1] Department of Electrical Engineering, ESAT/COSIC,
KU Leuven, Leuven, Belgium
atul.luykx@esat.kuleuven.be
[2] IMinds, Ghent, Belgium
[3] NTT Secure Platform Laboratories, Tokyo, Japan

Abstract. We present a new, misuse-resistant scheme for online authenticated encryption, following the framework set forth by Fleischmann et al. (FSE 2012). Our scheme, COBRA, is roughly as efficient as the GCM mode of operation for nonce-based authenticated encryption, performing one block cipher call plus one finite field multiplication per message block in a parallelizable way. The major difference from GCM is that COBRA preserves privacy up to prefix under nonce repetition. However, COBRA only provides authenticity against nonce-respecting adversaries. As compared to COPA (ASIACRYPT 2013), our new scheme requires no block cipher inverse and hence enjoys provable security under a weaker assumption about the underlying block cipher. In addition, COBRA can possibly perform better than COPA on platforms where finite field multiplication can be implemented faster than the block cipher in use, since COBRA essentially replaces half of the block cipher calls in COPA with finite field multiplications.

Keywords: COPA · OTR · GCM · Feistel network · ManTiCore · Authenticated online cipher · Deterministic · Finite-field multiplication

1 Introduction

Authenticated encryption (AE) schemes target the security goals of privacy and integrity. The field of AE has received more interest in the light of the recently announced CAESAR competition [9]. In the target scope of the competition fall secure and efficient AE algorithms for specific or possibly multiple environments.

While AE can securely be achieved by combining a probabilistic encryption scheme and a message authentication code using Bellare and Namprempre's generic composition [6], this approach comes at the cost of using two keys, one for encryption and one for authentication. This and further efficiency optimization reasons have led to the development of many dedicated nonce-based AE solutions such as CCM [33], CWC [20], EAX [7], GCM [23], IACBC [18], IAPM [18], OCB1-3 [21,27,29], and OTR [24].

C. Cid and C. Rechberger (Eds.): FSE 2014, LNCS 8540, pp. 187–204, 2015.
DOI: 10.1007/978-3-662-46706-0_10

Of these schemes, today GCM is the most widely deployed. GCM has been standardized by many organizations including ANSI, IEEE, ISO/IEC, and NIST. GCM has also been adopted by major cryptographic protocols such as IPsec, SSH, and TLS/SSL.

One advantage of GCM is that it performs well on Intel CPUs. According to Gladman [13], GCM outperforms CCM, CWC, and EAX on Intel P3/P4 and AMD 64(32/64) processors, if a 64 K table is used with GCM. This is mostly due to the fact that finite-field multiplication over GF(2^{128}) can be implemented efficiently on these platforms so that it runs faster than serial AES or hashing modulo $2^{127} - 1$.

There are several ways of parallelizing the polynomial hashing in GCM [14]. For example, instead of performing finite-field multiplications sequentially by Horner's rule as $\big(\big(\big(X[1]L \oplus X[2]\big)L \oplus X[3]\big)L \oplus X[4]\big)L$, one precomputes L^2, L^3 and L^4, stores them in a table, and then computes the hash in a parallelizable way as $X[1]L^4 + X[2]L^3 + X[3]L^2 + X[4]L$. Here L denotes the key of polynomial hashing and $X[i]$ the data blocks.

On more recent Intel CPUs such as Nehalem and Sandy Bridge, finite-field multiplication runs slower than AES [21]. Note that these processors are equipped with dedicated instruction sets, PCLMULQDQ for finite-field multiplication and AES-NI for AES block cipher computation. However, according to the latest report by Gueron [15] PCLMULQDQ is now more efficient on the latest Haswell processor, making finite-field multiplication over GF(2^{128}) faster than AES block cipher computation. This also makes GCM still attractive for use on Intel platforms.

Another advantage of GCM is the fact that it does not require the block cipher inverse. This contrasts sharply with schemes like OCB, where the cipher inverse is necessary for decryption. Besides the extra cost to implement the inverse algorithm, the problem is that the security proof needs to rely on a stronger assumption about the underlying block cipher if its inverse is used by the scheme. This issue has been discussed for OCB [5] and has led to the invention of OTR [24].

Given these features and its wide-spread use, GCM is often considered as a reference AE mode of operation. In fact, the call for submissions of the CAESAR competition [9] requires that authors "must explain, in particular, why users should prefer this cipher over AES-GCM."

All of the above-mentioned dedicated schemes are proven secure in a nonce-respecting model — formalism proposed by Rogaway [28] — where an adversary is limited to making encryption queries only with non-repeating nonce values. For the cases when nonce values do repeat, none of these AE schemes provides any formal security guarantees. Indeed, all of these schemes, including the latest OTR, can be "attacked" under nonce repetition, as described by Fleischmann et al. [12].

Nonce repetition can, however, occur in practice due to the fact that the nonce is chosen by the application programmer rather than the scheme itself as discussed by Fleishmann et al. [12]. Examples of nonce repetition are flawed

implementations [8,10,19,22,34], bad management of nonces by the user, and backup resets or virtual machine clones when the nonce is stored as a counter.

One way to address these situations is to design AE schemes which provide misuse resistance in a model where the adversary can perform queries with repeating nonces. Such schemes include the deterministic AE solutions SIV [30], BTM [16], and HBS [17], and also the authenticated online ciphers McOE-G [12], APE [2], and COPA [3]. The latter schemes are more efficient (need to process the message just once), even though the security under nonce repetition is limited to indistinguishability up to a common prefix.

We note that we are missing a "GCM-like" authenticated online cipher. McOE-G makes one block cipher call plus one finite-field multiplication per message block, but it is inherently sequential and not parallelizable like GCM. APE is permutation-based and sequential. COPA is parallelizable, but it makes two block cipher calls per message block. Moreover, all of these schemes require the inverse primitive calls for decryption. In this paper, therefore, we set out to propose a new authenticated online cipher whose efficiency is comparable to that of GCM.

Our Results. We present a secure and efficient solution for AE, which we name COBRA. A formal description of COBRA for integral message blocks is given in Sect. 3, and it is depicted in Figs. 2 and 3. (A description of COBRA for arbitrary-length messages is given in Appendix A).

Design. At first glance our design may seem to combine characteristics of the COPA [3] and OTR [24] designs. Indeed, to ensure misuse-resistance we include features from COPA and then substitute the parallelization procedures with the two-round balanced Feistel structure as proposed by Minematsu [24] in OTR. The latter design decision enables the use of just a single type of primitive, namely a block cipher in the forward encryption direction, without losing parallelizability, for efficiently authenticating and encrypting at the same time using polynomial hashing. It also allows for a scheme that does not need the inverse of the block cipher in decryption.

However, the construction of COBRA is *not* motivated by the mere combination of the two designs. Indeed, the employment of the Feistel network seems *necessary* for efficiently authenticating and encrypting at the same time using polynomial hashing. It also allows for a scheme that does not need the inverse of the block cipher in decryption. In order to achieve integrity of COBRA, we utilize the checksum of intermediate state values of the Feistel structure, which is similar to a technique proposed by Anderson et al. in their ManTiCore design [1].

Security. In Sect. 4, we prove that COBRA is secure against chosen-plaintext attacks (CPA) and against forgery up to approximately $2^{n/2}$ queries, where n is the block length of the underlying cipher. Our result for privacy covers nonce-repeating attackers. This contrasts sharply with GCM whose security collapses once the nonce is repeated. Note that authenticity of COBRA requires nonce-respecting attackers.

Our new scheme requires no block cipher inverse and hence enjoys provable security under the pseudo-random permutation (PRP) assumption about the underlying block cipher. This is not the case for COPA, whose security proof relies on a stronger assumption about the block cipher.

Our proof itself is simplified by decomposing COBRA into smaller parts which are dealt with individually. The main idea here is to turn a call to the block cipher into a call to a tweakable cipher which we instantiate with Rogaway's XE [27] construction. COBRA utilizes universal hashing (finite-field multiplication) and produces the tag using intermediate values of the Feistel networks. These differences make COBRA's proof slightly simpler than that of COPA.

Efficiency. The efficiency of COBRA is comparable to that GCM. That is, they both perform one block cipher call plus one finite field multiplication per message block in a parallelizable way.

As compared to COPA, COBRA saves the cost of implementing the inverse of the underlying block cipher. COBRA performs potentially better than COPA on platforms where the finite-field multiplication runs faster than the underlying block cipher call. Such CPUs include Intel's latest Haswell processor, where a 128-bit multiplication using the PCLMULQDQ instruction set runs faster than one AES call even using the AES-NI instruction set, hence essentially faster than any other block cipher implemented.

Attack on Previous Scheme. In the period between acceptance and publication of this paper, Nandi found an attack on the authenticity of the scheme using a nonce-repeating adversary [25]. As a result we have reduced the security claim of authenticity from being secure against nonce-repeating adversaries to being secure against nonce-respecting adversaries and we have made a small adjustment in the processing of the nonce to accomplish this security level: instead of multiplying the nonce with the message blocks, we use it in the block cipher call to create the secret value L. This does not change the privacy proof and authenticity is achieved for the same reason that authenticity is achieved in OTR.

2 Preliminaries

By $(\{0,1\}^n)^+$ we denote the set of strings whose length is a positive multiple of n bits. Given two strings A and B, we use $A \parallel B$ and AB interchangeably to denote the concatenation of A and B. For $A \in \{0,1\}^*$, by $A10^*$ we denote the string with a 1 appended, and then padded with zeros until its length is a multiple of n. If X is a string with length a multiple of n, by $X[i]$ we denote the ith n-bit block of X. The length of a string X is denoted by $|X|$.

A block cipher $E : \mathcal{K} \times \{0,1\}^n \to \{0,1\}^n$ is a function that takes as input a key $k \in \mathcal{K}$ and a plaintext $M \in \{0,1\}^n$, and produces a ciphertext $C = E(k, M)$. We sometimes write $E_k(\cdot) = E(k, \cdot)$. For a fixed key k, a block cipher is a permutation on n bits.

We can view the set $\{0,1\}^n$ of bit strings as the finite field $\mathrm{GF}(2^n)$ consisting of 2^n elements. To this end, we represent an element of $\mathrm{GF}(2^n)$ as a polynomial over

the field GF(2) of degree less than n, and a string $a_{n-1}a_{n-2}\cdots a_1a_0 \in \{0,1\}^n$ corresponds to the polynomial $a_{n-1}\mathtt{x}^{n-1} + a_{n-2}\mathtt{x}^{n-2} + \cdots + a_1\mathtt{x} + a_0 \in \mathrm{GF}(2^n)$. The addition in the field is simply addition of polynomials over GF(2) (i.e., bitwise XOR, denoted by $\oplus$). To define multiplication in the field, we fix an irreducible polynomial $f(\mathtt{x})$ of degree n over the field GF(2). For $a(\mathtt{x}), b(\mathtt{x}) \in \mathrm{GF}(2^n)$, their product is defined as $a(\mathtt{x})b(\mathtt{x}) \mod f(\mathtt{x})$ — polynomial multiplication over the field GF(2) reduced modulo $f(\mathtt{x})$. We simply write $a(\mathtt{x})b(\mathtt{x})$ and $a(\mathtt{x}) \cdot b(\mathtt{x})$ to mean the product in the field $\mathrm{GF}(2^n)$, and denote the multiplication by $\otimes$.

The set $\{0,1\}^n$ can alternatively be regarded as a set of integers ranging from 0 through $2^n - 1$, where a string $a_{n-1}a_{n-2}\cdots a_1a_0 \in \{0,1\}^n$ corresponds to the integer $a_{n-1}2^{n-1} + a_{n-2}2^{n-2} + \cdots + a_1 2 + a_0 \in [0, 2^n - 1]$. Based on these conversions, we often simply write elements of $\mathrm{GF}(2^n)$ as integers. For example, "2" means $\mathtt{x}$ and "3" means $\mathtt{x} + 1$. When we write multiplications such as $2 \cdot 3$, we mean those in the field $\mathrm{GF}(2^n)$.

3 Specification

In this section we give the specification of our scheme COBRA. Here we define COBRA for messages whose length is a positive multiple of $2n$, where n denotes the block length of the underlying block cipher. The case of fractional messages is given in Appendix A.

Let $E : \mathcal{K} \times \{0,1\}^n \to \{0,1\}^n$ be an n-bit block cipher. COBRA consists of two functionalities, an encryption function $\mathcal{E}$ and a decryption function $\mathcal{D}$:

$$\mathcal{E} : \mathcal{K} \times \{0,1\}^{n-1} \times \{0,1\}^* \times (\{0,1\}^{2n})^+ \to (\{0,1\}^{2n})^+ \times \{0,1\}^n,$$
$$\mathcal{D} : \mathcal{K} \times \{0,1\}^{n-1} \times \{0,1\}^* \times (\{0,1\}^{2n})^+ \times \{0,1\}^n \to (\{0,1\}^{2n})^+ \cup \{\perp\}.$$

The function $\mathcal{E}$ takes as input a key K, a nonce N, associated data A, and a message M, and returns a ciphertext C and tag T: $(C,T) \leftarrow \mathcal{E}(K,N,A,M)$. The decryption function $\mathcal{D}$ also gets a ciphertext C and tag T in addition to a key, nonce and associated data; it outputs M if the tag is correct and $\perp$ otherwise, which we denote as $M/\perp \leftarrow \mathcal{D}(K,N,A,C,T)$.

On input of a key K, nonce N, associated data A, and a message M padded into n-bit blocks $M[1]M[2]\cdots M[2d]$ (resp., a ciphertext $C = C[1]C[2]\cdots C[2d]$ and tag T), the function $\mathcal{E}$ (resp., $\mathcal{D}$) is defined in Fig. 1. Note that the functions are sound: for any K, N, A, M we have $M \leftarrow \mathcal{D}(K,N,A,\mathcal{E}(K,N,A,M))$. For the case the associated data A is of length at most $4n$ and the message is of length at most $6n$, the function $\mathcal{E}$ is depicted in Figs. 2 and 3.

4 Security

We briefly settle some notation for the security analysis in Sect. 4.1. In Sect. 4.2, we introduce some preliminary results related to COBRA. Confidentiality of COBRA is then proven in Sect. 4.3, and integrity in Sect. 4.4.

COBRA-Encrypt $\mathcal{E}[E](N, A, M)$:

$L \leftarrow E_k(N\|1), \Sigma \leftarrow 0$
$\tau \leftarrow 4L, V \leftarrow L$
for $i = 1, \ldots, d$ **do**
  $V \leftarrow V \oplus M[2i-1]$
  $C[2i-1] \leftarrow V$
  $V \leftarrow (V \otimes L) \oplus M[2i]$
  $C[2i] \leftarrow V$
  $\rho \leftarrow E_k(\tau \oplus C[2i])$
  $\Sigma \leftarrow \Sigma \oplus \rho$
  $C[2i-1] \leftarrow \rho \oplus C[2i-1]$
  $\sigma \leftarrow E_k(\tau \oplus L \oplus C[2i-1])$
  $\Sigma \leftarrow \Sigma \oplus \sigma$
  $C[2i] \leftarrow \sigma \oplus C[2i]$
  if $i < d$ **then**
    $V \leftarrow V \otimes L$
    $\tau \leftarrow 2\tau$
  end if
end for

$U \leftarrow$ProcessAD$[E](A)$
$T \leftarrow$ComputeTag$[E](L, \tau, \Sigma, N, U)$
return (C, T)

COBRA-Decrypt $\mathcal{D}[E](N, A, C, T)$:

$L \leftarrow E_k(N\|1), \Sigma \leftarrow 0$
$\tau \leftarrow 4L, V \leftarrow L$
for $i = 1, \ldots, d$ **do**
  $\sigma \leftarrow E_k(\tau \oplus L \oplus C[2i-1])$
  $\Sigma \leftarrow \Sigma \oplus \sigma$
  $M[2i] \leftarrow \sigma \oplus C[2i]$
  $\rho \leftarrow E_k(\tau \oplus M[2i])$
  $\Sigma \leftarrow \Sigma \oplus \rho$
  $V' \leftarrow \rho \oplus C[2i-1]$
  $M[2i-1] \leftarrow V' \oplus V$
  $V' \leftarrow V' \otimes L$
  $V \leftarrow M[2i]$
  $M[2i] \leftarrow V' \oplus M[2i]$
  if $i < d$ **then**
    $V \leftarrow V \otimes L$
    $\tau \leftarrow 2\tau$
  end if
end for

$U \leftarrow$ProcessAD$[E](A)$
$T' \leftarrow$ComputeTag$[E](L, \tau, \Sigma, N, U)$
return $T = T' \ ? \ M : \perp$

ProcessAD$[E](A)$:

$X \leftarrow A \parallel 10^*$
$J \leftarrow E_k(0)$
$U \leftarrow J$
for $i = 1, \ldots, |X|/n - 1$ **do**
  $U \leftarrow (U \oplus X[i]) \otimes J$
end for
$U \leftarrow E_k(2J \oplus U \oplus X[c])$
return U

ComputeTag$[E](L, \tau, \Sigma, N, U)$:

$\tau \leftarrow 3(\tau \oplus L)$
$T \leftarrow E_k(\tau \oplus \Sigma)$
$\tau \leftarrow 3\tau$
$T \leftarrow E_k(\tau \oplus T \oplus N \oplus U)$
return T

Fig. 1. COBRA.

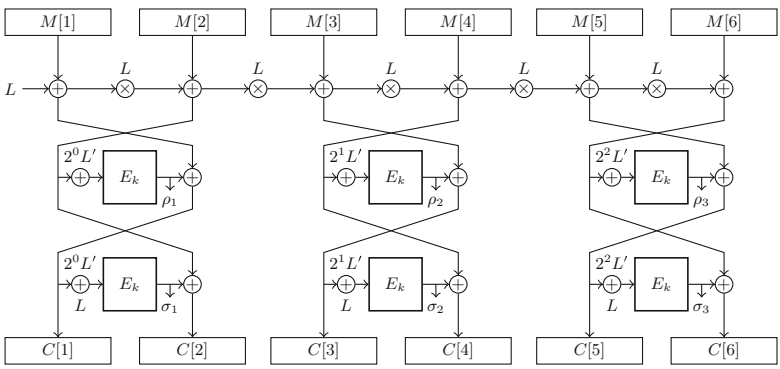


Fig. 2. Processing plaintext. Note that L' is defined in Fig. 3 below.

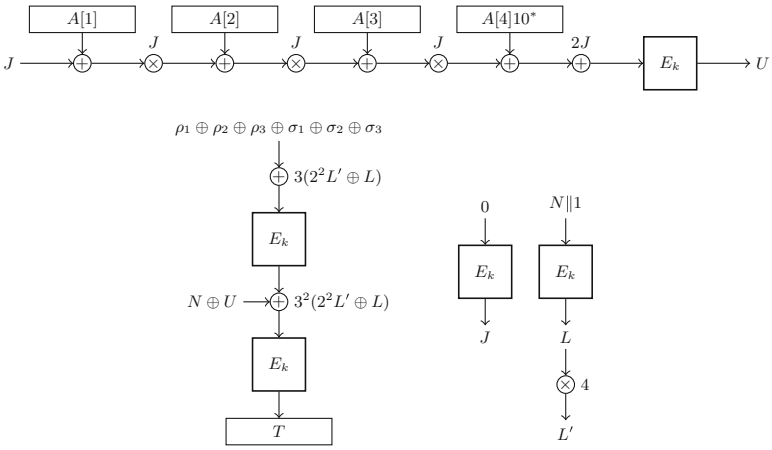


Fig. 3. Processing associated data (top), computing the tag (bottom left), and the secret values (bottom right).

4.1 Notation

When writing $x \xleftarrow{\$} X$ for some finite set X we mean that x is sampled uniformly from X. We write $\Pr[\mathbf{A} \mid \mathbf{B}]$ to denote the probability of event $\mathbf{A}$ given $\mathbf{B}$.

Say that $M \in \{0,1\}^{2n\ell}$. We write $M[1]M[2]\cdots M[2\ell] \xleftarrow{n} M$ to denote the *blocks* that make up M, and $\hat{M}[1]\hat{M}[2]\cdots\hat{M}[\ell] \xleftarrow{2n} M$ to denote the *fragments* that make up M. Note that a fragment is made of two blocks: $\hat{M}[i] = M[2i-1] \,\|\, M[2i]$.

For convenience, we use the notation

$$\underset{\mathbf{D}}{\Delta}(f\,;\,g) := \left|\Pr[\mathbf{D}^f = 1] - \Pr[\mathbf{D}^g = 1]\right| \tag{1}$$

to denote the distinguishing advantages of adversary $\mathbf{D}$ in distinguishing oracles f and g, where the notation $\mathbf{D}^{\mathcal{O}}$ indicates the value output by $\mathbf{D}$ after interacting with oracle $\mathcal{O}$. The probabilities are defined over the random coins used in the oracles and the random coins of the adversary, if any. If a class of distinguishers is described by some parameters, e.g. the number of queries q, then by $\Delta_q(f\,;\,g)$ we denote the supremum of $\Delta_{\mathbf{D}}(f\,;\,g)$ over all distinguishers $\mathbf{D}$ in this class of adversaries. Multiple oracles are separated by a comma or given by a set, e.g. $\Delta(f_1, f_2\,;\,g_1, g_2)$ or $\Delta(\{f_1, f_2\}\,;\,\{g_1, g_2\})$ denotes distinguishing the combination of f_1 and f_2 from the combination of g_1 and g_2.

A uniform random function (URF) from m bits to n bits is a uniformly distributed random variable over the set of all functions from $\{0,1\}^m$ to $\{0,1\}^n$. A uniform random permutation (URP) on n bits is a uniformly distributed random variable over the set of all permutations on n bits.

Definition 1. *Let E be a block cipher. Let π be a URP on n bits. The* prp *advantage of a distinguisher* $\mathbf{D}$ *is defined as*

$$\mathbf{Adv}_E^{\mathrm{prp}}(\mathbf{D}) = \underset{\mathbf{D}}{\Delta}(E_k\,;\,\pi).$$

Here, $\mathbf{D}$ *is a distinguisher with oracle access to either E_k or π. The probabilities are taken over $k \xleftarrow{\$} \mathcal{K}$, the randomness of π, and random coins of* $\mathbf{D}$*, if any. By $\mathbf{Adv}_E^{\mathrm{prp}}(t,q)$ we denote the maximum advantage taken over all distinguishers that run in time t and make q queries.*

4.2 Preliminary Results

The input to each block cipher call in COBRA is first XORed with one of the following masks:

$$\{2J, 2^iL, 2^iL \oplus L, 3(2^iL \oplus L), 3^2(2^iL \oplus L)\}, \tag{2}$$

where $i \geq 2$, $J := E_k(0)$ and $L := E_k(1)$. As a result, each block cipher call can be viewed as a call to an XE construction [27]. Note that by using the doubling method from [27], we can produce many different values of the mask from the secret values J and L. Specifically, we adopt the tweaks used in [24], allowing us to replace each of the XEs with independent URFs.

Lemma 1 ([24,27])**.** *Let $\mathcal{T}$ denote some set of indices such that $\tau \to \mu_\tau$ maps all indices to all tweaks injectively. The permutations $\{E_k(\mu_\tau \oplus \cdot)\}_{\tau\in\mathcal{T}}$ are indistinguishable from independent URFs $\{\varphi_\tau\}_{\tau\in\mathcal{T}}$. Specifically, let* $\mathbf{D}$ *be a distinguisher running in time t and making at most q queries, then*

$$\underset{\mathbf{D}}{\Delta}(\{E_k(\mu_\tau \oplus \cdot)\}_{\tau\in\mathcal{T}}\,;\,\{\varphi_\tau\}_{\tau\in\mathcal{T}}) \leq \frac{5q^2}{2^n} + \mathbf{Adv}_E^{\mathrm{prp}}(\mathbf{D}'),$$

where $\mathbf{D}'$ *is a distinguisher with running time similar to* $\mathbf{D}$*, making $2q$ queries.*

In Fig. 4 one can see a description of COBRA where the XE constructions are replaced with URFs, where the URFs are labeled α (replacing the XE construction in ProcessAD), β_i^N, γ_i^N for $i \geq 1$ and all N (replacing them in COBRA-Encrypt and COBRA-Decrypt), and δ_1^N, δ_2^N for all N (replacing them in ComputeTag). Throughout, we will denote this scheme by $\mathcal{E}'$.

We can describe $\mathcal{E}'$ as a sequence of functions each computing one ciphertext fragment, a function computing the tag, and a function processing the associated data. More formally:

Definition 2. *Say that $\mathcal{E}'$ maps (N, A, M) to (C, T). Define $f_i : \{0,1\}^n \times \{0,1\}^{2ni} \to \{0,1\}^{2n}$ to be the function mapping $(N, \hat{M}[1]\cdots\hat{M}[i])$ to $\hat{C}[i]$, $h : \{0,1\}^* \to \{0,1\}^n$ the function mapping A to U (where U is as shown in Fig. 4), and $f' : \{0,1\}^n \times \{0,1\}^* \times (\{0,1\}^{2n})^+ \to \{0,1\}^n$ the function mapping (N, A, M) to T.*

As a second step, we replace the associated data computation h in $\mathcal{E}'$ with a URF Ω:

Lemma 2. *Let $\Omega : \{0,1\}^* \to \{0,1\}^n$ be a URF and let $\mathbf{D}$ be a distinguisher making at most q queries each of length less than nl, then*

$$\underset{\mathbf{D}}{\Delta}(h\,;\,\Omega) \leq \frac{lq^2}{2^n}.$$

Proof. The URF α generates independent, uniformly distributed values as long as its inputs are unique. The only issue is when two different A's map to the same input to α, which itself reduces to finding zeros of a polynomial in J of degree at most l. Since J is an independent, uniformly distributed value generated using a URP, and polynomials of degree l have at most l distinct zeroes, the probability that a pair of plaintexts collides is $l/2^n$. By allowing the adversary to make q queries we get our desired bound. □

We define $\mathbb{E}$ to be $\mathcal{E}'$ with h replaced by Ω. In other words, $\mathbb{E}$ corresponds to COBRA where (i) the XE constructions have been replaced with independent URFs, and (ii) h has been replaced with Ω. Formally, we obtain the following result for $\mathbb{E}$.

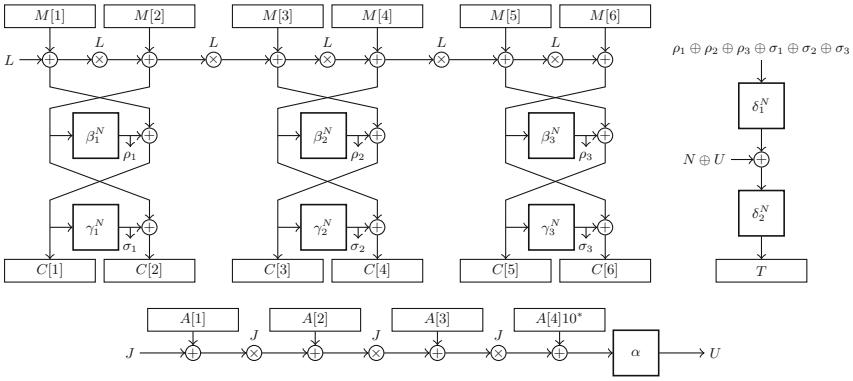


Fig. 4. Construction with independent URFs.

Proposition 1. *Let $(\mathcal{E}, \mathcal{D})$ denote COBRA and let $\mathbf{D}$ be a distinguisher making at most q queries each of length less than $2n\ell$, then*

$$\underset{\mathbf{D}}{\Delta}(\mathcal{E}_k\,;\,\mathbb{E}) \leq \frac{5(2\ell q + 2q)^2}{2^n} + \frac{2\ell q^2}{2^n} + \mathbf{Adv}_E^{\mathrm{prp}}(\mathbf{D}'),$$

where the probability is taken over $k \xleftarrow{\$} \mathcal{K}$ and the URFs in $\mathbb{E}$, and $\mathbf{D}'$ is a distinguisher with running time similar to $\mathbf{D}$, making $4\ell q + 4q$ queries.

Proof. We first apply Lemma 1, where we note that one query by $\mathbf{D}$ leads to at most $2\ell + 2$ block cipher calls, and then Lemma 2 to get the desired result. □

Using Proposition 1, we will prove confidentiality of COBRA in Sect. 4.3 and integrity in Sect. 4.4. For the proof of confidentiality, we present an additional elementary lemma:

Lemma 3. *Say f_1, g_1 are random functions independent of each other, and that f_2, g_2 are independent random functions as well. Let $\mathbf{D}$ be a distinguisher for $\{f_1, g_1\}$ and $\{f_2, g_2\}$ making q_f queries to the f_i oracles and q_g queries to the g_i oracle. Then there exist distinguishers $\mathbf{D}_f$ and $\mathbf{D}_g$ such that*

$$\underset{\mathbf{D}}{\Delta}(f_1, g_1 \,;\, f_2, g_2) \leq \underset{\mathbf{D}_f}{\Delta}(f_1 \,;\, f_2) + \underset{\mathbf{D}_g}{\Delta}(g_1 \,;\, g_2),$$

where $\mathbf{D}_f$ makes q_f queries and $\mathbf{D}_g$ makes q_g queries.

4.3 Confidentiality

We adopt the definitions of security given in [3], yet rather than comparing our scheme to a random variable over the set of all online permutations, we explicitly describe an ideal online function in terms of URFs.

Definition 3 (Ideal Online Function). *Let $g_i : \{0,1\}^n \times \{0,1\}^{2ni} \to \{0,1\}^{2n}$ be URFs and let $g' : \{0,1\}^n \times \{0,1\}^* \times (\{0,1\}^{2n})^+ \to \{0,1\}^n$ be a URF. We define $\$: \{0,1\}^n \times \{0,1\}^* \times (\{0,1\}^{2n})^+ \to (\{0,1\}^{2n})^+ \times \{0,1\}^n$ as*

$$\$(N, A, M) = g_1(N, \hat{M}[1]) \parallel g_2(N, \hat{M}[1]\hat{M}[2]) \parallel \cdots \parallel g_\ell(N, M) \parallel g'(N, A, M)$$

where $\hat{M}[1]\hat{M}[2]\cdots\hat{M}[\ell] \xleftarrow{2n} M$.

Definition 4 (IND-CPA). *Let $\mathcal{E}$ be an encryption scheme. The IND-CPA advantage of a distinguisher $\mathbf{D}$ relative to $\mathcal{E}$ is given by*

$$\mathbf{Adv}^{\mathrm{cpa}}_{\mathcal{E}}(\mathbf{D}) := \underset{\mathbf{D}}{\Delta}(\mathcal{E}_k \,;\, \$),$$

where $k \xleftarrow{\$} \mathcal{K}$ and $\$$ is as defined in Definition 3.

Theorem 1. *Let $\mathcal{E}$ denote COBRA and let $\mathbf{D}$ be a distinguisher running in time t and making at most q queries to $\mathcal{E}$, each of length less than $2n\ell$, then*

$$\mathbf{Adv}^{\mathrm{cpa}}_{\mathcal{E}}(\mathbf{D}) \leq \frac{22(\ell+1)^2 q^2}{2^n} + \mathbf{Adv}^{\mathrm{prp}}_{E}(\mathbf{D}'),$$

where $\mathbf{D}'$ is a distinguisher with running time similar to $\mathbf{D}$, making $4\ell q + 4q$ queries.

Proof. Let $\mathbf{D}$ be a distinguisher running in time t and making at most q queries each of length less than $2n\ell$. As a first step, we move from $\mathcal{E}$ to $\mathbb{E}$, where the underlying XE constructions are replaced by independent URFs, and h by Ω. By Proposition 1:

$$\underset{\mathbf{D}}{\Delta}(\mathcal{E}_k \,;\, \mathbb{E}) \leq \frac{5(2\ell q + 2q)^2}{2^n} + \frac{2q^2\ell}{2^n} + \mathbf{Adv}^{\mathrm{prp}}_{E}(\mathbf{D}'), \tag{3}$$

where $\mathbf{D}'$ has running time similar to $\mathbf{D}$ and makes at most $4\ell q + 4q$ queries. Next, note that the f_i's and f' (cf. Definition 2) are independent functions, as their underlying URFs $\beta_i^N, \gamma_i^N, \delta_i^N$ are independent functions. By Lemma 3:

$$\underset{\mathbf{D}}{\Delta}(\mathbb{E}\,;\,\$) \leq \underset{\mathbf{D}_t}{\Delta}(f'\,;\,g') + \sum_{i=1}^{\ell-1} \underset{\mathbf{D}_i}{\Delta}(f_i\,;\,g_i). \tag{4}$$

for some $\mathbf{D}_t$ that makes at most q queries of total length less than $2n\ell$ and $\mathbf{D}_i$ (for $i \in \{1, \ldots, \ell-1\}$) that makes at most q queries (of fixed length). A bound on $\Delta_{\mathbf{D}_i}(f_i\,;\,g_i)$ for arbitrary i is derived in Lemma 4. In Lemma 5 we compute a bound on $\Delta_{\mathbf{D}_t}(f'\,;\,g')$. We find:

$$\underset{\mathbf{D}}{\Delta}(\mathbb{E}\,;\,\$) \leq \frac{q^2(2\ell+3)}{2^n} + \sum_{i=1}^{\ell-1} \frac{q^2 2(2i+1)}{2^n} = \frac{q^2(2\ell+3)}{2^n} + \frac{q^2 2(\ell^2-1)}{2^n}. \tag{5}$$

The proof is completed by simplifying the obtained bound. □

Lemma 4. *Let $\mathbf{D}_i$ be a distinguisher making at most q queries, then*

$$\underset{\mathbf{D}_i}{\Delta}(f_i\,;\,g_i) \leq \frac{2(2i+1)q^2}{2^n}.$$

Proof. The proof is similar to that of Lemma 2. We use that the inputs to the URFs are polynomials of degree at most $2i+1$, and that f_i consists of two URFs β_i^N and γ_i^N. □

Lemma 5. *Let $\mathbf{D}_t$ be a distinguisher making at most q queries each of length less than $2n\ell$, then*

$$\underset{\mathbf{D}_t}{\Delta}(f'\,;\,g') \leq \frac{(2\ell+3)q^2}{2^n}.$$

Proof. Without loss of generality, we may assume that the distinguisher does not make repeat queries. Say that $\{(N_1, A_1, M_1), \ldots, (N_i, A_i, M_i)\}$ is the query history. We consider what happens on query (N^*, A^*, M^*).

Let $U_i := \Omega(A_i)$ and let Σ_i denote input to $\delta_1^{N_i}$ (see Fig. 4). Let U^* and Σ^* be the corresponding values for (N^*, A^*, M^*). We compute the probability that

$$U^* \oplus \delta_1^{N^*}(\Sigma^*) \oplus N^* = U_j \oplus \delta_1^{N_j}(\Sigma_j) \oplus N_j \tag{6}$$

for some j for which $1 \leq j \leq i$.

1. If $A^* \neq A_j$ then U^* is independent of $U_j \oplus \delta_1^{N_j}(\Sigma_j) \oplus \delta_1^{N^*}(\Sigma^*) \oplus N^* \oplus N_j$, hence the probability that Eq. (6) is satisfied is not more than $1/2^n$.
2. If $A^* = A_j$, then $U^* = U_j$ and we focus on the probability that

$$\delta_1^{N^*}(\Sigma^*) \oplus N^* = \delta_1^{N_j}(\Sigma_j) \oplus N_j. \tag{7}$$

(a) If $M^* = M_j$, then $\Sigma^* = \Sigma_j$ and Eq. (7) reduces to $\delta_1^{N^*}(\Sigma^*) \oplus \delta_1^{N_j}(\Sigma^*) = N^* \oplus N_j$. Since we do not allow repeat queries $N^* \neq N_j$, and so this occurs with probability $1/2^n$.

(b) Say that $M^* \neq M_j$, and let ρ^* and ρ_j denote the output of the last call to β made when processing M^* and M_j, respectively. If ρ^* and ρ_j are independent, then $\Sigma^* = \Sigma_j$ with probability $1/2^n$. If $\Sigma^* \neq \Sigma_j$, then $\delta_1^{N^*}(\Sigma^*)$ and $\delta_1^{N_j}(\Sigma_j)$ are independent (and also independent of N^* and N_j), hence the probability of Eq. (7) being true is upper bounded by $2/2^n$. The probability that ρ^* and ρ_j are not independent is upper bounded by the probability of having a collision in the inputs to the last β call (and only if M^* and M_j are the same length), which is $(2\ell + 1)/2^n$.

Putting our results together we get that

$$\Pr(\text{Eq. (6) holds}) \leq \max\left\{\frac{1}{2^n}, \frac{2\ell+1}{2^n} + \frac{2}{2^n}\right\} = \frac{2\ell+3}{2^n}. \tag{8}$$

This means that the probability that $U^* \oplus \delta_1^{N^*}(\Sigma^*) \oplus N^*$ collides with any of the previous $U_j \oplus \delta_1^{N_j}(\Sigma_j) \oplus N_j$ is upper bounded by $\frac{q(2\ell+3)}{2^n}$. As long as the input to δ_2 is unique, the tag produced is uniform and independent of all previous values, hence f' remains indistinguishable from g'. Summing over all queries, we get the desired bound. □

4.4 Integrity

Definition 5. *Let $\mathcal{E}$ be an AE scheme. The integrity advantage of a distinguisher $\mathbf{D}$ relative to $\mathcal{E}$ is given by*

$$\mathbf{Adv}^{\text{int}}_{\mathcal{E}}(\mathbf{D}) := \underset{\mathbf{D}}{\Delta}(\mathcal{E}_k, \mathcal{D}_k\,;\, \mathcal{E}_k, \perp),$$

where $k \xleftarrow{\$} \mathcal{K}$ and $\perp$ is a function that responds with $\perp$ on every query. We assume that the distinguisher does not make queries of the form $\mathcal{D}_k(N, A, C, T)$, where $(C, T) = \mathcal{E}_k(N, A, M)$ for some previously queried (N, A, M) and that it does not query $\mathcal{E}_k$ twice under the same nonce.

Theorem 2. *Let $\mathcal{E}$ denote COBRA and let $\mathbf{D}$ be a distinguisher running in time t and making at most q queries to $\mathcal{E}$ and q_f forgery attempts, each of length less than $2n\ell$, then*

$$\mathbf{Adv}^{\text{int}}_{\mathcal{E}}(\mathbf{D}) \leq \frac{(3q+1)q_f}{2^n} + \frac{22(\ell+1)^2q^2}{2^n} + \mathbf{Adv}^{\text{prp}}_{E}(\mathbf{D}'),$$

where $\mathbf{D}'$ is a distinguisher with running time similar to $\mathbf{D}$, making $4\ell q + 4q$ queries.

Proof. As with the proof of confidentiality, we use Proposition 1 to switch to $\mathbb{E}$. We first focus on adversaries with one forgery attempt, with (N^*, A^*, C^*, T^*) being the attempt. Let $\{(N_1, A_1, M_1), \ldots, (N_q, A_q, M_q)\}$ denote the history of

queries made by the adversary to the encryption oracle, where (C_i, T_i) is the output corresponding to (N_i, A_i, M_i) and each N_i is distinct. Let $U_i := \Omega(A_i)$, $U^* := \Omega(A^*)$, and let Σ_i denote the input to $\delta_1^{N_i}$ during the computation of (N_i, A_i, M_i); define Σ^* similarly. Note that

$$\delta_2^{N_i}\left(U_i \oplus \delta_1^{N_i}(\Sigma_i) \oplus N_i\right) = T_i, \tag{9}$$

and similarly for T^*.

If

$$U^* \oplus \delta_1^{N^*}(\Sigma^*) \oplus N^* \neq U_i \oplus \delta_1^{N_i}(\Sigma_i) \oplus N_i \tag{10}$$

for all i, then the input to δ_2 is distinct from all previous inputs to δ_2, hence the output of δ_2 from the forgery query is uniformly distributed and independent of T^*, which means that the forgery will be successful with probability at most $1/2^n$. Hence we focus on computing the probability that there is an i resulting in a collision in the δ_2 input.

Fix an i such that $1 \leq i \leq q$. We compute the probability that

$$U^* \oplus \delta_1^{N^*}(\Sigma^*) \oplus N^* = U_i \oplus \delta_1^{N_i}(\Sigma_i) \oplus N_i. \tag{11}$$

1. If $A^* \neq A_i$, then U^* is uniformly distributed and independent of U_i, hence the probability that Eq. (11) is satisfied is bounded above by $1/2^n$.
2. If $A^* = A_i$, then $U^* = U_i$ and we focus on the probability that

$$\delta_1^{N^*}(\Sigma^*) \oplus N^* = \delta_1^{N_i}(\Sigma_i) \oplus N_i. \tag{12}$$

   (a) If $C^* = C_j$, Eq. (12) reduces to

$$\delta_1^{N^*}(\Sigma^*) \oplus N^* = \delta_1^{N_i}(\Sigma^*) \oplus N_i. \tag{13}$$

   Since $A^* = A_j$, then either $N^* \neq N_i$, in which case we only get a successful forgery with probability $1/2^n$, or $N^* = N_j$ and $T^* \neq T_j$, in which case we get a failed forgery attempt as well.

   (b) Say that $C^* \neq C_j$, and that they differ at the mth fragment, i.e. $\hat{C}^*[m] \neq \hat{C}_j[m]$. We also assume that $N^* = N_j$, because if $N^* \neq N_j$ then the tags are independent of each other since they are produced by independent URFs $\delta_2^{N^*}$ and $\delta_2^{N_j}$.
   Since $N^* = N_j$, and N_j does not equal any of the other N_i, Σ^* is independent of all Σ_i for $i \neq j$. If $C^*[2m-1] = C_j[2m-1]$, then $C^*[2m] \neq C_j[2m]$, hence the inputs to $\beta_m^{N^*}$ for C^* and C_j are different. This means that $\Sigma^* = \Sigma_j$ with probability at most $1/2^n$, hence $\delta_1(\Sigma^*) = \delta_1(\Sigma_j)$ with probability at most $2/2^n$. If $\delta_1^{N^*}(\Sigma^*) \neq \delta_1^{N_j}(\Sigma_j)$, then Eq. (12) is satisfied with probability at most $1/2^n$ since N^* and N_i are independent of the outputs of δ_1. If $C^*[2m-1] \neq C_j[2m-1]$, we can apply the same reasoning.

Putting the above results together, we get that the probability of Eq. (11) being satisfied is bounded above by $3/2^n$. Hence, the probability that there exists an i satisfying (11) is bounded above by $3q/2^n$. The probability that the forgery is successful is thus bounded above by $3q/2^n + 1/2^n$.

Generalizing to adversaries which can make up to q_f forgery queries as explained in Andreeva et al. [4], we have our desired bound. □

5 Future Work

We shall implement COBRA and compare its software performance with GCM and COPA. It is interesting to see how much of an overhead COBRA actually has over GCM on a specific platform, possibly due to the Feistel network, larger state, extra mask generation, and the reverse order of multplication and block cipher call.

Acknowledgments. The authors would like to thank FSE 2014 reviewers for their valuable comments. This work has been funded in part by the IAP Program P6/26 BCRYPT of the Belgian State (Belgian Science Policy), in part by the European Commission through the ICT program under contract ICT-2007-216676 ECRYPT II, in part by the Research Council KU Leuven: GOA TENSE, and in part by the Research Fund KU Leuven, OT/08/027. Elena Andreeva is supported by a Postdoctoral Fellowship from the Flemish Research Foundation (FWO-Vlaanderen). Bart Mennink and Atul Luykx are supported by Ph.D. Fellowships from the Institute for the Promotion of Innovation through Science and Technology in Flanders (IWT-Vlaanderen).

References

1. Anderson, E., Beaver, C.L., Draelos, T., Schroeppel, R., Torgerson, M.: ManTiCore: encryption with joint cipher-state authentication. In: Wang, H., Pieprzyk, J., Varadharajan, V. (eds.) ACISP 2004. LNCS, vol. 3108, pp. 440–453. Springer, Heidelberg (2004)
2. Andreeva, E., Bogdanov, A., Luykx, A., Mennink, B., Mouha, N., Yasuda, K.: APE: authenticated permutation-based encryption for lightweight cryptography. In: Cid, C., Rechberger, C. (eds.) FSE 2014. LNCS, vol. 8540, pp. 168–186. Springer, Heidelberg (2014)
3. Andreeva, E., Bogdanov, A., Luykx, A., Mennink, B., Tischhauser, E., Yasuda, K.: Parallelizable and authenticated online ciphers. In: Sako, K., Sarkar, P. (eds.) ASIACRYPT 2013, Part I. LNCS, vol. 8269, pp. 424–443. Springer, Heidelberg (2013)
4. Andreeva, E., Bogdanov, A., Luykx, A., Mennink, B., Tischhauser, E., Yasuda, K.: Parallelizable and authenticated online ciphers. Cryptology ePrint Archive (2013). (full version of this paper)
5. Aoki, K., Yasuda, K.: The security of the OCB mode of operation without the SPRP assumption. In: Susilo, W., Reyhanitabar, R. (eds.) ProvSec 2013. LNCS, vol. 8209, pp. 202–220. Springer, Heidelberg (2013)

6. Bellare, M., Namprempre, C.: Authenticated encryption: relations among notions and analysis of the generic composition paradigm. In: Okamoto, T. (ed.) ASIACRYPT 2000. LNCS, vol. 1976, pp. 531–545. Springer, Heidelberg (2000)
7. Bellare, M., Rogaway, P., Wagner, D.: The EAX mode of operation. In: Roy and Meier [32], pp. 389–407
8. Borisov, N., Goldberg, I., Wagner, D.: Intercepting mobile communications: the insecurity of 802.11. In: Rose, C. (ed.) MOBICOM, pp. 180–189. ACM (2001)
9. CAESAR: Competition for Authenticated Encryption: Security, Applicability, and Robustness (April 2013). http://competitions.cr.yp.to/caesar.html
10. Cantero, H.M., Peter, S., Bushing, Segher: Console hacking 2010 - PS3 epic fail. In: 27th Chaos Communication Congress, December 2010
11. Chakraborty, D., Sarkar, P.: Hch: a new tweakable enciphering scheme using the hash-counter-hash approach. IEEE Trans. Inf. Theory **54**(4), 1683–1699 (2008)
12. Fleischmann, E., Forler, C., Lucks, S.: McOE: a family of almost foolproof on-line authenticated encryption schemes. In: Canteaut, A. (ed.) FSE 2012. LNCS, vol. 7549, pp. 196–215. Springer, Heidelberg (2012)
13. Gladman, B.: AES and combined encryption/authentication modes (2006). http://www.gladman.me.uk/
14. Gopal, V., Ozturk, E., Feghali, W., Guilford, J., Wolrich, G., Dixon, M.: Optimized Galois-Counter-Mode implementation on Intel architecture processors. Intel Corporation White Paper (2010)
15. Gueron, S.: AES-GCM software performance on the current high end CPUs as a performance baseline for CAESAR competition. In: Directions in Authenticated Ciphers (DIAC) (2013)
16. Iwata, T., Yasuda, K.: BTM: a single-key, inverse-cipher-free mode for deterministic authenticated encryption. In: Jacobson Jr, M.J., Rijmen, V., Safavi-Naini, R. (eds.) SAC 2009. LNCS, vol. 5867, pp. 313–330. Springer, Heidelberg (2009)
17. Iwata, T., Yasuda, K.: HBS: a single-key mode of operation for deterministic authenticated encryption. In: Dunkelman, O. (ed.) FSE 2009. LNCS, vol. 5665, pp. 394–415. Springer, Heidelberg (2009)
18. Jutla, C.S.: Encryption modes with almost free message integrity. J. Cryptology **21**(4), 547–578 (2008)
19. Kohno, T.: Attacking and repairing the WinZip encryption scheme. In: Atluri, V., Pfitzmann, B., McDaniel, P.D. (eds.) ACM Conference on Computer and Communications Security, pp. 72–81. ACM (2004)
20. Kohno, T., Viega, J., Whiting, D.: CWC: a high-performance conventional authenticated encryption mode. In: Roy and Meier [32], pp. 408–426
21. Krovetz, T., Rogaway, P.: The software performance of authenticated-encryption modes. In: Joux, A. (ed.) FSE 2011. LNCS, vol. 6733, pp. 306–327. Springer, Heidelberg (2011)
22. Lenstra, A.K., Hughes, J.P., Augier, M., Bos, J.W., Kleinjung, T., Wachter, C.: Public keys. In: Safavi-Naini, R., Canetti, R. (eds.) CRYPTO 2012. LNCS, vol. 7417, pp. 626–642. Springer, Heidelberg (2012)
23. McGrew, D.A., Viega, J.: The security and performance of the galois/counter mode (GCM) of operation. In: Canteaut, A., Viswanathan, K. (eds.) INDOCRYPT 2004. LNCS, vol. 3348, pp. 343–355. Springer, Heidelberg (2004)
24. Minematsu, K.: Parallelizable rate-1 authenticated encryption from pseudorandom functions. In: Nguyen, P.Q., Oswald, E. (eds.) EUROCRYPT 2014. LNCS, vol. 8441, pp. 275–292. Springer, Heidelberg (2014)

25. Nandi, M.: Forging Attack on COBRA. Cryptographic Competitions Google Group (2014). https://groups.google.com/d/msg/crypto-competitions/nhqcgEThcPc/ryvKY7lfMhMJ
26. Ristenpart, T., Rogaway, P.: How to enrich the message space of a cipher. In: Biryukov, A. (ed.) FSE 2007. LNCS, vol. 4593, pp. 101–118. Springer, Heidelberg (2007)
27. 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)
28. Rogaway, P.: Nonce-based symmetric encryption. In: Roy, B., Meier, W. (eds.) FSE 2004. LNCS, vol. 3017, pp. 348–359. Springer, Heidelberg (2004)
29. Rogaway, P., Bellare, M., Black, J., Krovetz, T.: OCB: a block-cipher mode of operation for efficient authenticated encryption. In: Reiter, M.K., Samarati, P. (eds.) ACM Conference on Computer and Communications Security, pp. 196–205. ACM (2001)
30. Rogaway, P., Shrimpton, T.: A provable-security treatment of the key-wrap problem. In: Vaudenay, S. (ed.) EUROCRYPT 2006. LNCS, vol. 4004, pp. 373–390. Springer, Heidelberg (2006)
31. Rogaway, P., Wooding, M., Zhang, H.: The security of ciphertext stealing. In: Canteaut, A. (ed.) FSE 2012. LNCS, vol. 7549, pp. 180–195. Springer, Heidelberg (2012)
32. Roy, B., Meier, W. (eds.): FSE 2004. LNCS, vol. 3017. Springer, Heidelberg (2004)
33. Whiting, D., Housley, R., Ferguson, N.: AES encryption and authentication using CTR mode and CBC-MAC. IEEE 802.11-02/001r2 (2002)
34. Wu, H.: The Misuse of RC4 in Microsoft Word and Excel. Cryptology ePrint Archive, Report 2005/007 (2005)

A COBRA for Arbitrary-Length Messages

We use ciphertext stealing [31] in order to deal with messages of arbitrary length. Let M be a message where $M[1]M[2]\cdots M[2\ell-1]M[2\ell] = M$ and $|M[i]| = n$ for $1 \leq i < 2\ell - 1$.

A.1 $\ell > 1$, $|M[2\ell-1]| = n$, and $0 < |M[2\ell]| < n$

We start by computing the ciphertext of $M[1]\cdots M[2\ell-2]$ as is usually done in COBRA, resulting in $C[1]\cdots C[2\ell-2]$. Let M^* denote the rightmost $|M[2\ell]|$ bits of $C[2\ell-2]$, and we write $C[2\ell-2] = C'[2\ell-2]M^*$. Then we compute the final ciphertext fragment $C[2\ell-1]C[2\ell]$ using $M[2\ell-1]M[2\ell]M^*$ as our "new" final message fragment, using different tweaks for the final block cipher calls. The resulting ciphertext is

$$C[1]\cdots C[2\ell-3]C'[2\ell-2]C[2\ell-1]C[2\ell]. \tag{A.1}$$

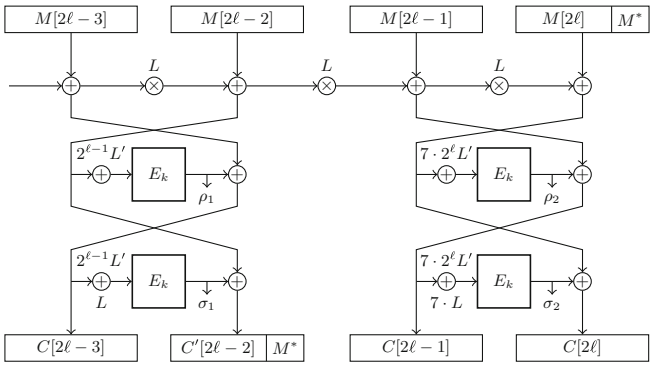


Fig. 5. Messages where the last block is not of full length, i.e. $0 < |M[2\ell]| < n$. Here M^* is "stolen" from ciphertext block $C[2\ell - 2]$ and used in the input to the final fragment.

Fig. 5 shows a diagram of the process. Note that we can recover M^* with just knowledge of $C[2\ell - 1]$ and $C[2\ell]$:

$$M[2\ell]M^* = \Big[C[2\ell] \oplus E_{k,\tau_2}(C[2\ell - 1])\Big] \oplus \Big(\Big[E_{k,\tau_1}\big(C[2\ell] \oplus E_{k,\tau_2}(C[2\ell - 1])\big) \oplus C[2\ell - 1]\Big] \otimes L\Big),$$

where $E_{k,\tau_1}(x) := E_k(x \oplus 7 \cdot 2^\ell L')$ and $E_{k,\tau_2}(x) := E_k(x \oplus 7 \cdot (2^\ell L' \oplus L))$.

A.2 $\ell > 2$ and $0 < |M[2\ell - 1]| \leq n$

When there is no last block $M[2\ell]$, we replace it with the preceding ciphertext block, $C[2\ell - 2]$. Then we steal ciphertext M^* of length $|M[2\ell - 1]|$ from the ciphertext block $C[2\ell - 4]$ such that $C[2\ell - 4] = C'[2\ell - 4]M^*$. The rest of the computation is similar to the previous case (Sect. A.1) and is depicted in Fig. 6.

A.3 $|M| \leq 3n$

The above methods only work for messages of length greater than $3n$ (otherwise there is no ciphertext to steal from). We need to use different techniques in order to deal with shortest messages. For $2n < |M| \leq 3n$ we can use a technique similar as to what is used in COPA [3]. Instead of using XLS [26] which uses the inverse block cipher, we can use HCH [11] in order to compute the output as follows:

$$C[1]C[2]T' \leftarrow \mathcal{E}(M[1]M[2]) \tag{A.2}$$

$$C[3]T \leftarrow \mathrm{HCH}(M[3]T'), \tag{A.3}$$

where $\mathcal{E}$ denotes COBRA and the final output of the scheme is $C[1]C[2]C[3]T$.

For $n < |M| < 2n$ we can use the tag-splitting method: we first compute

$$C[1]C[2]T \leftarrow \mathcal{E}(M[1]M[2]10^*), \tag{A.4}$$

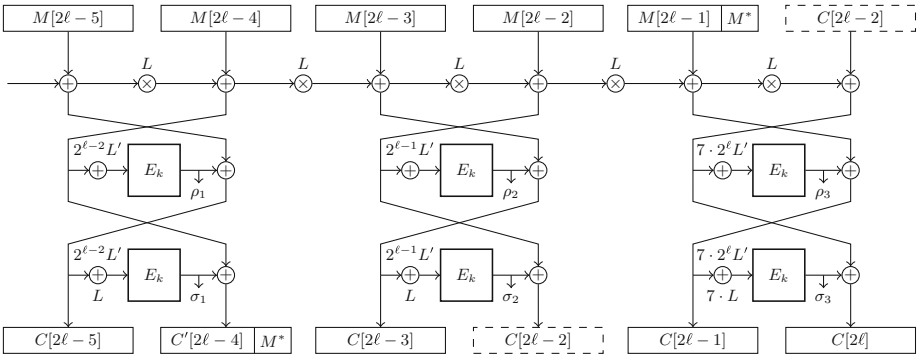


Fig. 6. Messages where the last fragment is of length less than or equal to n, i.e. $0 < |M[2\ell - 1]| \leq n$. Here M^* is stolen from ciphertext block $C[2\ell - 4]$ and used in the input to the final fragment together with ciphertext fragment $C[2\ell - 2]$.

then remove part of the tag so that the length of the output is equal to the length of the input. Here, again, $\mathcal{E}$ is COBRA, except different tweaks must be used from the case in which $|M| = 2n$.

Pipelineable On-line Encryption

Farzaneh Abed[1], Scott Fluhrer[2], Christian Forler[1](✉), Eik List[1], Stefan Lucks[1], David McGrew[2], and Jakob Wenzel[1]

[1] Bauhaus-Universität Weimar, Weimar, Germany
{farzaneh.abed,christian.forler,eik.list,stefan.lucks,
jakob.wenzel}@uni-weimar.de
[2] Cisco Systems, San Jose, USA
{sfluhrer,mcgrewd}@cisco.com

Abstract. Correct authenticated decryption requires the receiver to buffer the decrypted message until the authenticity check has been performed. In high-speed networks, which must handle large message frames at low latency, this behavior becomes practically infeasible. This paper proposes CCA-secure on-line ciphers as a practical alternative to AE schemes since the former provide some defense against malicious message modifications. Unfortunately, all published on-line ciphers so far are either inherently sequential, or lack a CCA-security proof.

This paper introduces POE, a family of on-line ciphers that combines provable security against chosen-ciphertext attacks with pipelineability to support efficient implementations. POE combines a block cipher and an ϵ-AXU family of hash functions. Different instantiations of POE are given, based on different universal hash functions and suitable for different platforms. Moreover, this paper introduces POET, a provably secure on-line AE scheme, which inherits pipelineability and chosen-ciphertext-security from POE and provides additional resistance against nonce-misuse attacks.

Keywords: On-line cipher · Chosen-ciphertext security · Authenticated encryption

1 Introduction

Authenticated Encryption (AE) schemes (such as EAX [8], GCM [31], OCB [28], etc.) perform an authentication check on the entire ciphertext before they output a decrypted message. This practice is inherent in the idea of authenticated

C. Forler—The research leading to these results received funding from the European Research Council under the European Union's Seventh Framework Programme (FP/2007–2013)/ERC Grant Agreement no. 307952.

S. Lucks—A part of this research was done while Stefan Lucks was visiting the National Institute of Standards and Technologies (NIST), during his sabbatical.

C. Cid and C. Rechberger (Eds.): FSE 2014, LNCS 8540, pp. 205–223, 2015.
DOI: 10.1007/978-3-662-46706-0_11

encryption and part of its strength. However, it is incompatible with settings that pose demanding performance requirements (e.g., high speed, low latency, long messages).

One example for such settings are Optical Transport Networks (OTNs) [24], in which the links between multiple network channels must be capable of transmitting, multiplexing, and switching between immense data streams in a fast and secure manner. OTNs are characterized by high throughput rates of up to 100 Gbps, low latencies in the order of a few clock cycles, and large message frames of up to 64 kB. At that size, a mode of operation of a 128-bit block cipher would require over 4,096 clock cycles to complete a decryption—which exceeds the allowed latency in OTN systems by far.

In such uses of AE, implementations have to pass along (part of) a decrypted message before validating its authenticity; if the message later turns out to be invalid, this fact will be discovered and reported, but only after some information has been leaked. The literature calls this practice *decryption misuse* [19], and describes severe vulnerabilities for conventional AE schemes. A chosen-ciphertext adversary can exploit it to determine unknown plaintexts, or to introduce forged message fragments that may get passed to the application and are processed before the authentication check is completed. As a consequence, common existing AE schemes do not suit well in this environment. To overcome this issue, this work considers authenticated encryption schemes that provide robustness against decryption misuse through *on-line chosen-ciphertext security* (OPRP-CCA) [5]. Implementations of AE schemes that allow decryption misuse abound, even when latency is not a consideration. For example, many software libraries provide access to encryption and decryption operations through a stream-oriented interface that consists of functions for initialization, updating, and finalization. In these interfaces the decrypt-update function can be called multiple times.[1] *Every* invocation of this function performs decryption misuse, because it releases the would-be plaintext before completing the authentication check. This type of interface is incompatible with existing authenticated encryption schemes. But its use is widespread, well-established and will not easily go away.

Decryption-Misuse Resistance. An encryption scheme is called *non-malleable* if any change to a ciphertext causes its entire post-decryption plaintext to be pseudorandom [18]. We call such a scheme *decryption-misuse-resistant* since the decryption of manipulated ciphertext results in uncontrollable random noise. Unfortunately, non-malleability and on-line encryption are mutually exclusive: if an adversary manipulates the i-th block of a ciphertext, an on-line encryption scheme leaves the previous $(i-1)$ blocks unchanged. But OPRP-CCA-security is the strongest form of non-malleability and decryption-misuse resistance an on-line cipher can provide: if an adversary manipulates the i-th block, all plaintext blocks starting from the i-th one will become pseudorandom.

The concept of decryption-misuse-resistant AE schemes is controversial. During the *Dagstuhl Seminar* on Symmetric Cryptography in January 2014 some

[1] For example, see the OpenSSL `EVP_DecryptUpdate` function [44].

researchers were worried about the risk of *advertising* decryption-misuse resistance as a feature for AE schemes since it could *invite* application programmers to improperly implement authenticated decryption. Of course, misuse must be avoided where possible, e.g., by user education. Nevertheless, decryption misuse is common in practice,[2] as our example of OTNs illustrates. The choice for the cryptograph is to either deal with decryption misuse, or to abandon AE completely.

Support for Intermediate Tags. Beyond limiting the harm of decryption misuse OPRP-CCA-secure on-line ciphers allow another desirable feature: *Intermediate tags* [9] allow the receiver to early detect if parts of a decrypted message are invalid—which is handy when authenticating large messages. They can be integrated easily into an OPRP-CCA-secure on-line cipher by adding some form of well-formed redundancy (e.g., fixed constants or non- cryptographic checksums) to the plaintexts. For example, the headers of IP, TCP, or UDP [36–38] packets already contain a 16-bit checksum each, which is *verified* by the receiver and/or network routers. In OTNs, a single 64-kB message frame consists of multiple IP packets. Due to the low-latency constraints, receiving routers cannot buffer incoming messages until the authentication check has finished and must forward the first packets to their destination. However, they can test the packets' checksums to detect forgery attempts early. Hence, OPRP-CCA-security ensures that false TCP/IP packets only pass with probability of at most 2^{-16}.

Previous Work and Contributions. An ideal on-line cipher should be both IND-CCA-secure *and* non- sequential, i.e., parallelizable or pipelineable.[3] Already in 1978 Campbell published an early on-line cipher, called *Infinite Garble Extension* (IGE), which is far from complying with current security goals. In 2000 Knudsen [26] proposed his *Accumulated Block Chaining* (ABC) mode. In their landmark paper from 2001 Bellare et al. [5] coined the term of and security notions for on-line ciphers, and presented two instances, HCBC-1 and HCBC-2, based on the combination of a block cipher and a keyed hash function. Both constructions are inheritly sequential—HCBC-2 was slightly slower than HCBC-1, but provided additional IND-CCA-security. In 2002 Rivest, Liskov and Wagner [29,30] presented a non-sequential, tweakable on-line cipher, called TIE. However, TIE could not provide CCA-security due to a counter-based tweak input. In 2003 Halevi and Rogaway [22] proposed the *EME* approach (encryption-mix-encryption), which has inspired several on-line cipher designs since then. EME is a symmetric design concept that consists of five layers: an initial whitening step, an ECB layer, a linear mixing, a second ECB layer, and a final whitening. In 2004 Boldyrea and Taseombut [11] proposed security notions for on-line authenthentication ciphers, and the *HPCBC* mode as an instantiation. In 2007 and 2008 Nandi proposed further on-line ciphers similar to that of Bellare et al. [32,33]. In the same year the IEEE standardized the XTS [23]

[2] As is nonce misuse: considering security under nonce-misuse has been a novelty a few years ago [40], but has become an established design goal nowadays.

[3] We call an operation f *pipelineable* if it can be split into multiple parts $f = f_2 \circ f_1$, s.t. f_1 can process the $(i+1)$-th input block before f_2 has finished processing the i-th block.

Table 1. Classification of on-line encryption schemes.

	Sequential	Non-sequential
CCA-insecure	ABC [26], CBC [34], CFB [34], HCBC-1 [5], IGE [12], OFB [34], TC [30], TC1 [41]	COPE [3], CTR [17], ECB [34], TIE [30], XTS [23]
CCA-secure	APE(X) [4], CMC [21], HCBC-2 [5], MCBC [33], McOE [19], MHCBC [33], TC2/3 [41]	**POE**

mode of operation for disk encryption; however, which also lacked CCA-security. In 2011 Rogaway and Zhang [41] described methods to construct secure on-line ciphers from tweakable block ciphers. However, it is easy to see that all mentioned schemes until here are either inherently sequential or CCA-insecure. Table 1 shows a summarized classification.

Contribution. This paper introduces the *Pipelineable On-line Encryption* (POE, hereafter) family of on-line ciphers, which consists of an ECB layer that is wrapped by two chaining layers of a keyed family of ϵ-AXU hash functions. The resulting construction is provably IND-CCA-secure and pipelineable, i.e., POE allows to process neighboring input blocks efficiently. To address different platforms, this work proposes three instantiations of POE, based on the AES as cipher and different families of universal hash functions. Furthermore, we show that POE can be easily transformed into an OPRP-CCA-secure, robust on-line AE (OAE) scheme, called *Pipelineable On-line Encryption with Tag* (POET hereafter), using well-studied methods from [19].

Recent Related Work. To the best of our knowledge, only four nonce-misuse-resistant OAE schemes were published prior to this work:[4] (1) McOE [19], (2) APE(X) [4], (3) COPA [3], and (4) ELmE [15]. McOE is a TC3-like design that was introduced at FSE 2012, and pioneered nonce-misuse resistance as a considerable feature for OAE schemes; APE(X), COPA, and ELmE are recent designs, where APE(X) bases on the Sponge, and COPA as well as ELmE on the EME design. McOE and APE(X) provide OPRP-CCA-security, but work inherently sequential, COPA and ELmE are parallelizable, and may outperform POET when running on high- end hardware or multi-core systems. However, the EME structure implies that both require two block-cipher calls for each message block, whereas POE and POET employ only a single cipher and two hash-function calls. Hence, we expect POET to perform better than EME-based designs on medium- and low-end systems with few cores and no native AES instructions. Moreover, we illustrate in Appendix A that EME- based designs lose the OPRP-CCA-security in the

[4] Regarding the state *before* the CAESAR submission deadline. The research inspired by the CAESAR competition brought multiple further constructions that can be added to this list, including but not limited to COBRA, ELmD, or AEZ.

decryption-misuse setting, which disqualifies COPA and ELmE for the OTN application scenario. More generally, Datta and Nandi [16] showed recently that EME constructions with linear mixing can not provide IND-CCA-security. Therefore, POET represents the first non-sequential OAE scheme with resistance against *both* nonce *and* decryption misuse.

Outline. The remainder of this work is structured as follows. Section 2 recalls the preliminary information about universal hash functions, on-line ciphers, and AE schemes that is necessary for this work. In Sect. 3, we propose the POE family of on-line ciphers and prove its security against chosen-plaintext and chosen-ciphertext attacks. Thereupon, Sect. 4, introduces POET, and provides a proof for the security against chosen-ciphertext attacks. Section 5 proposes three practical instantiations for POE and POET. Finally, we draw a conclusion in Sect. 6.

2 Preliminaries

This section revisits the well- known definitions of universal hash-function families from Carter and Wegman [13,43], as well as notions for on-line ciphers from Bellare et al. [5,6]. Prior, Table 2 summarizes the general notions.

2.1 Notions for Universal Hash Functions

Definition 1 (ϵ-Almost-(XOR-)Universal Hash Functions). *Let $m, n \geq 1$ be integers. Let $\mathcal{H} = \{H : \{0,1\}^m \rightarrow \{0,1\}^n\}$ be a family of hash functions. We call $\mathcal{H}$* ϵ-almost-universal *(ϵ-AU) if for all $X, X' \in \{0,1\}^m$, $X \neq X'$:*

$$\Pr_{H}[H \xleftarrow{\$} \mathcal{H} : \ H(X) = H(X')] \leq \epsilon.$$

Table 2. Notions used throughout this paper.

N	Nonce (initial value)
M	Plaintext message
C	Ciphertext
K	User-given secret key
$\|X\|$	Length of X in bits
n	Block length in bits
k	Key length in bits
X_i	i-th block of a value X
$X \mid\mid Y$	Concatenation of two values X and Y
$\mathcal{X}$	Set X
$X \xleftarrow{\$} \mathcal{X}$	X is a uniformly at random chosen sample from $\mathcal{X}$.

We call $\mathcal{H}$ ϵ-almost-XOR-universal *(ϵ-AXU) if for all* $X, X' \in \{0,1\}^m$, $Y \in \{0,1\}^n$, $X \neq X'$*:*

$$\Pr_H[H \xleftarrow{\$} \mathcal{H} : \ H(X) \oplus H(X') = Y] \leq \epsilon.$$

Boesgaard et al. [10] showed that an ϵ-AXU family of hash functions can be reduced to a family of ϵ-AU hash functions by the following theorem:

Theorem 1 (Theorem 3 from [10]). *Let* $m, n \geq 1$ *be integers. Let* $\mathcal{H} = \{H : \{0,1\}^m \to \{0,1\}^n\}$ *be a family of* ϵ*-AXU hash functions. Then, the familiy* $\mathcal{H}' = \{H' : \{0,1\}^m \times \{0,1\}^n \to \{0,1\}^n\}$ *with* $H'(X,Y) = H(X) \oplus Y$ *is* ϵ*-AU.*

2.2 Notions for On-line Ciphers

Block Ciphers. A block cipher is a keyed family of n-bit permutations $E{:}\{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n$ which takes a k-bit key K and an n-bit message M and outputs an n-bit ciphertext C. We define $\texttt{Block}(k,n)$ as the set of all (k,n)-bit block ciphers for $n > 0$. For any $E \in \texttt{Block}(k,n)$ and a fixed key $K \in \{0,1\}^k$, the encryption of a message M is defined by $E_K(M)$, and the decryption is defined as the inverse function, i.e., $E_K^{-1}(M)$. For any key $K \in \{0,1\}^k$, it applies that $E_K^{-1}(E_K(M)) = M$.

Definition 2 (On-line Cipher). *Let* $k, n \geq 1$ *be integers and let* $\Gamma : \{0,1\}^k \times (\{0,1\}^n)^* \to (\{0,1\}^n)^*$ *be a keyed family of* n*-bit permutations which takes a* k*-bit key* K *and a message* M *of an arbitrary number of* n*-bit blocks, and outputs a ciphertext* C *consisting of the same number of* n*-bit blocks as* M*. We call* Γ *an* on-line cipher *iff the encryption of message block* M_i*, for all* $i \in [1, |M|/n]$*, depends only on the blocks* $M_1, \ldots, M_i$*.*

A secure cipher should behave like a random permutation. It is easy to see that on-line ciphers cannot guarantee this property since the encryption of message block M_i does not depend on M_{i+1}. The on-line behavior implies that two messages M, M' that share an m-block common prefix are always encrypted to two ciphertexts C, C' that also share an m-block common prefix. Hence, an on-line cipher Γ is called secure iff no ciphertext reveals any further information about a plaintext than its length and the *longest common prefix* with previous messages. For a formal definition of the longest common prefix of two messages, we refer to [19].

Definition 3 (On-line Permutation). *Let* $i, j, \ell, n \geq 1$ *be integers. Let* $F_i : (\{0,1\}^n)^i \to \{0,1\}^n$ *be a family of indexed* n*-bit permutations, i.e., for a fixed index* $j \in (\{0,1\}^n)^{i-1}$ *it applies that* $F_i(j, \cdot)$ *is a permutation. We define an* n*-bit on-line permutation* $P : (\{0,1\}^n)^\ell \to (\{0,1\}^n)^\ell$ *as a composition of* ℓ *permutations* $F_1 \cup F_2 \cup \ldots \cup F_\ell$*. An* ℓ*-block message* $M = (M_1, \ldots, M_\ell)$ *is mapped to an* ℓ*-block output* $C = (C_1, \ldots, C_\ell)$ *by*

$$C_i = F_i(M_1 \,||\, \ldots \,||\, M_{i-1}, M_i), \quad \forall i \in [1, \ell].$$

Remark 1. For any two ℓ-block inputs M, M', with $M \neq M'$, that share an exactly m-block common prefix $M_1 \,||\, \ldots \,||\, M_m$, the corresponding outputs $C = P(M)$, $C' = P(M')$ satisfy $C_i = C'_i$ for all $i \in [1, m]$ and $m \leq \ell$. However, it applies that $C_{m+1} \neq C'_{m+1}$, and all further blocks C_i, C'_i, with $i \in [m+2, \ell]$, are likely to be different.

In the following, we denote by OPerm_n the set of all n-bit on-line permutations. Furthermore, we denote by $P \xleftarrow{\$} \mathsf{OPerm}_n$ that P is chosen as a random on-line permutation. Note that a random on-line permutation can be efficiently implemented by lazy-sampling.

On-line Authenticated Encryption Scheme (With Associated Data). An authenticated encryption scheme is a triple $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$. $\mathcal{K}$ denotes a key-generation procedure that returns a randomly chosen key K; the encryption algorithm $\mathcal{E}_\mathcal{K}(H, M)$ and its inverse decryption algorithm $\mathcal{D}_\mathcal{K}(H, C, T)$ are deterministic algorithms, where H denotes the header, M the message, T the authentication tag, and C the ciphertext, with $H, M, C \in (\{0,1\}^n)^*$ and $T \in \{0,1\}^n$. We define that the final header block is a nonce. $\mathcal{E}$ always outputs a ciphertext C, and $\mathcal{D}$ outputs either the plaintext M that corresponds to C, or $\perp$ if the authentication tag T is invalid. Note that we call an authenticated encryption scheme $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ on-line if $\mathcal{E}$ is an on-line cipher and $\mathcal{D}$ is its inverse operation.

3 The On-line Cipher POE

This section introduces the POE family of on-line ciphers and shows that it is secure against chosen-plaintext and chosen-ciphertext attacks.

3.1 Definition of POE

Definition 4 (POE). *Let $k, n \geq 1$ be integers, $E\colon \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n$ a block cipher, and $F : \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n$ a family of keyed ϵ-AXU hash functions. Furthermore, let $F_i : \{0,1\}^{ni} \to \{0,1\}^n$ be $i\epsilon$-AXU family of hash functions defined as follows:*

$$F_0 = F(1); \quad F_i(M) = F(F_{i-1}(M_1, \ldots, M_{i-1}) \oplus M_i) \qquad i \in \mathbb{N}^+.$$

Let $K, K_1, K_2 \in \{0,1\}^k$ denote three pair-wise independent keys. Then, we define the encryption of POE and its inverse as shown in Algorithm 1.

A schematic illustration of the encryption algorithm is given in Fig. 1.

3.2 Security Notions for On-line Ciphers

The IND-SPRP-security of a block cipher E is defined by the success probability of an adversary $\mathcal{A}$ to distinguish the output of E, E^{-1} from that of an n-bit random permutation π.

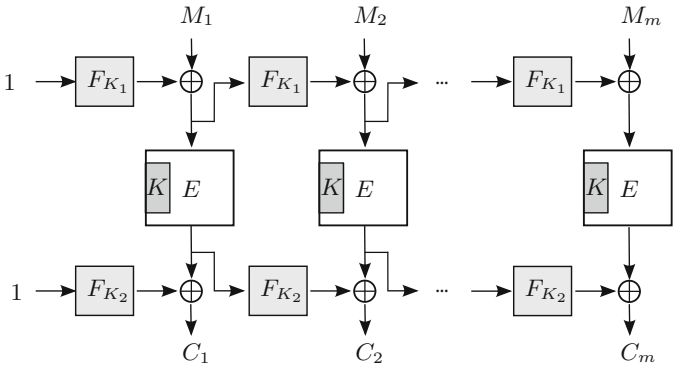


Fig. 1. The encryption process for an m-block message M with POE.

Algorithm 1. Procedures **Encrypt** and **Decrypt** for POE.

Encrypt(M)	**Decrypt**(C)
1: $m \leftarrow \lvert M\rvert/n$, $X_0 \leftarrow 1, Y_0 \leftarrow 1$	11: $m \leftarrow \lvert C\rvert/n$, $X_0 \leftarrow 1, Y_0 \leftarrow 1$
2: **for** $i = 1, \ldots, m$ **do**	12: **for** $i = 1, \ldots, m$ **do**
3: $\quad X_i \leftarrow F_{K_1}(X_{i-1}) \oplus M_i$	13: $\quad Y_i \leftarrow F_{K_2}(Y_{i-1}) \oplus C_i$
4: $\quad Y_i \leftarrow E_K(X_i)$	14: $\quad X_i \leftarrow E_K^{-1}(Y_i)$
5: $\quad C_i \leftarrow F_{K_2}(Y_{i-1}) \oplus Y_i$	15: $\quad M_i \leftarrow F_{K_1}(X_{i-1}) \oplus X_i$
6: **end for**	16: **end for**
7: **return** $(C_1 \parallel \ldots \parallel C_m)$	17: **return** $(M_1 \parallel \ldots \parallel M_m)$

Definition 5 (IND-SPRP-Security). *Let $E \in \mathtt{Block}(k, n)$ denote a block cipher and E^{-1} its inverse. Let Perm_n be the set of all n-bit permutations. The IND-SPRP advantage of $\mathcal{A}$ against E is then defined by*

$$\mathbf{Adv}^{\mathsf{IND\text{-}SPRP}}_{E,E^{-1}}(\mathcal{A}) \leq \left|\Pr\left[\mathcal{A}^{E(\cdot),E^{-1}(\cdot)} \Rightarrow 1\right] - \Pr\left[\mathcal{A}^{\pi(\cdot),\pi^{-1}(\cdot)} \Rightarrow 1\right]\right|,$$

where the probabilities are taken over $K \xleftarrow{\$} \{0,1\}^k$ and $\pi \xleftarrow{\$} \mathsf{Perm}_n$. We define $\mathbf{Adv}^{\mathsf{IND\text{-}SPRP}}_{E,E^{-1}}(q,t)$ as the maximum advantage over all IND-SPRP-adversaries $\mathcal{A}$ on E that run in time at most t and make at most q queries to the available oracles.

We borrow the OPRP-CCA notion from Bellare et al. [5,6]. The OPRP-CCA-security specifies the maximal advantage of an adversary $\mathcal{A}$ with access to an encryption and decryption oracle to distinguish the outputs of a on-line cipher Γ under a randomly chosen key K from that of a random permutation.

Definition 6 (OPRP-CCA-Security). *Let K a k-bit key, P a random on-line permutation, and $\Gamma\colon \{0,1\}^k \times (\{0,1\}^n)^* \to (\{0,1\}^n)^*$ be an on-line cipher. Then, we define the OPRP-CCA-advantage of an adversary $\mathcal{A}$ by*

$$\mathbf{Adv}^{\mathsf{OPRP\text{-}CCA}}_{\Gamma}(\mathcal{A}) = \left|\Pr\left[\mathcal{A}^{\Gamma_K(\cdot),\Gamma_K^{-1}(\cdot)} \Rightarrow 1\right] - \Pr\left[\mathcal{A}^{P(\cdot),P^{-1}(\cdot)} \Rightarrow 1\right]\right|, \quad (1)$$

where the probabilities are taken over $K \xleftarrow{\$} \mathcal{K}$ *and* $P \xleftarrow{\$} \mathit{OPerm}_n$. *Further, we define* $\mathbf{Adv}^{\mathsf{OPRP\text{-}CCA}}_{\Gamma}(q, \ell, t)$ *as the maximum advantage over all adversaries* $\mathcal{A}$ *that run in time at most* t, *and make at most* q *queries of total length of at most* ℓ *blocks to the available oracles.*

Bellare and Namprempre showed in [7] that IND-CCA-security implies *non-malleable chosen-ciphertext*-security (NM-CCA). Hence, OPRP-CCA implies *weak* non-malleability, i.e., an adversary that manipulates the i-th ciphertext block cannot distinguish the $(i+1)$-th, $(i+2)$-th,... ciphertext blocks of Γ from random.

3.3 OPRP-CCA-Security of POE

Theorem 2. *Let* $E\colon \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n$ *be a block cipher and* E^{-1} *its inverse operation. Let* $\pi \xleftarrow{\$} \mathit{Perm}_n$ *denote an* n*-bit random permutation that was chosen uniformly from random, and let* π^{-1} *denote its inverse. Then, it holds that*

$$\mathbf{Adv}^{\mathsf{OPRP\text{-}CCA}}_{\mathsf{POE}_{E,E^{-1}}}(q,\ell,t) \leq \ell^2\epsilon + \frac{\ell^2}{2^n - \ell} + \mathbf{Adv}^{\mathsf{IND\text{-}SPRP}}_{E,E^{-1}}(\ell, O(t)). \tag{2}$$

Proof. Let $\mathcal{A}$ be an OPRP-CCA-adversary with access to an oracle $\mathcal{O}$, which responds either with real encryption/decryptions using $\mathsf{POE}_{E_K,E_K^{-1}}$ or a random on-line permutation P, as given in Definition 6. We say that $\mathcal{A}$ collects its queries and the corresponding oracle response as tuples (M, C) in a query history $\mathcal{Q}$. Wlog., we assume that $\mathcal{A}$ will not make queries to which it already knows the answer.

It is easy to see that we can rewrite Eq. (1) as (cf. [19], Sect. 4):

$$\mathbf{Adv}^{\mathsf{OPRP\text{-}CCA}}_{\mathsf{POE}_{E,E^{-1}}}(\mathcal{A}) \leq \left|\Pr\left[\mathcal{A}^{\mathsf{POE}_E,\mathsf{POE}^{-1}_{E^{-1}}} \Rightarrow 1\right] - \Pr\left[\mathcal{A}^{\mathsf{POE}_\pi,\mathsf{POE}^{-1}_{\pi^{-1}}} \Rightarrow 1\right]\right| \tag{3}$$

$$+ \left|\Pr\left[\mathcal{A}^{\mathsf{POE}_\pi,\mathsf{POE}^{-1}_{\pi^{-1}}} \Rightarrow 1\right] - \Pr\left[\mathcal{A}^{P(\cdot),P^{-1}(\cdot)} \Rightarrow 1\right]\right|. \tag{4}$$

It is easy to see that Eq. (3) can be upper bounded by

$$\mathbf{Adv}^{\mathsf{IND\text{-}SPRP}}_{E,E^{-1}}(\ell, \mathrm{O}(t)).$$

It remains to study the difference in (4), which refers to the advantage of $\mathcal{A}$ to distinguish POE instantiated with an n-bit random permutation π from P. We can identify two cases from the structure of POE: (1) collisions between internal values of POE occur (COLL), or (2) no collisions occur (NOCOLL). From the law of total probability follows that we can rewrite (4) as

$$\begin{aligned}&\left|\Pr\left[\mathcal{A}^{\mathsf{POE}_\pi,\mathsf{POE}^{-1}_{\pi^{-1}}} \Rightarrow 1\right] - \Pr\left[\mathcal{A}^{P(\cdot),P^{-1}(\cdot)} \Rightarrow 1\right]\right| \\ &\leq \Pr\left[\mathrm{COLL}\right] \cdot \Pr\left[\mathrm{COLLWIN}\right] + \Pr\left[\neg\,\mathrm{COLL}\right] \cdot \Pr\left[\mathrm{NOCOLLWIN}\right],\end{aligned}$$

with

$$\Pr\left[\mathrm{COLLWIN}\right] = \left|\Pr\left[\mathcal{A}^{\mathsf{POE}_\pi,\mathsf{POE}^{-1}_{\pi^{-1}}} \Rightarrow 1 \,|\, \mathrm{COLL}\right] - \Pr\left[\mathcal{A}^{P(\cdot),P^{-1}(\cdot)} \Rightarrow 1\right]\right|,$$

$$\Pr\left[\mathrm{NOCOLLWIN}\right] = \left|\Pr\left[\mathcal{A}^{\mathsf{POE}_\pi,\mathsf{POE}^{-1}_{\pi^{-1}}} \Rightarrow 1 \,|\, \neg\,\mathrm{COLL}\right] - \Pr\left[\mathcal{A}^{P(\cdot),P^{-1}(\cdot)} \Rightarrow 1\right]\right|.$$

For the sake of simplicity, we upper bound $\Pr[\text{COLLWIN}]$ and $\Pr[\neg\,\text{COLL}]$ by 1. Thus, we only have to look at $\Pr[\text{COLL}]$ and $\Pr[\text{NOCOLLWIN}]$.

***Case 1:* COLL.** In this case, $\mathcal{A}$ tries to distinguish POE from random by exploiting some collision between internal values. Since π is a random permutation, any *fresh* (i.e., not previously queried) input to $\pi(\cdot)$ or $\pi^{-1}(\cdot)$ produces a random output and therefore:

1. For any fresh X_i, the result of $\pi(X_i) \oplus F_{K_2}(Y_{i-1})$ will be random.
2. For any fresh Y_i, the result of $\pi^{-1}(Y_i) \oplus F_{K_1}(X_{i-1})$ will be random.

We obtain two possible subcases: a collision between internal values in the top row occurred (COLL_{top}), or a collision between in internal values in the bottom row occurred (COLL_{bot}). COLL then represents the event that either (or both) subcases occurred.

$$\text{COLL} \;=\; \text{COLL}_{\text{top}} \vee \text{COLL}_{\text{bot}}.$$

***Subcase 1.1:* $\mathbf{COLL_{top}}$.** By an internal collision in the top row, we refer to the event that $X_i = X'_j$ for two distinct tuples (X_{i-1}, M_i) and (X'_{j-1}, M'_j), with $i, j \geq 1$:

$$X_i = F_{K_1}(X_{i-1}) \oplus M_i, \quad \text{and} \quad X'_j = F_{K_1}(X'_{j-1}) \oplus M'_j.$$

Since F is an ϵ-AXU family of hash functions, the family F' of hash functions

$$F'_{K_1}(X_{i-1}, M_i) \;:=\; F_{K_1}(X_{i-1}) \oplus M_i$$

is ϵ-AU (cf. Theorem 1). Thus, the probability of a top-row collision for at most ℓ queried message blocks can be upper bounded by

$$\Pr[\text{COLL}_{\text{top}}] \;=\; \frac{\ell(\ell-1)}{2} \cdot \epsilon \;\leq\; \frac{\ell^2}{2}\epsilon.$$

***Subcase 1.2:* $\mathbf{COLL_{bot}}$.** We define a bottom-row collision as the event that two distinct tuples (Y_{i-1}, C_i) and (Y'_{j-1}, C'_j) produce the same values $Y_i = Y'_j$, with

$$Y_i = F_{K_2}(Y_{i-1}) \oplus E_K(X_i), \quad \text{and} \quad Y'_j = F_{K_2}(Y'_{j-1}) \oplus E_K(X'_j).$$

Due to the symmetric structure of POE, the analysis for bottom-row collisions is similar to that of top-row collisions. Thus, the probability for this event can also be upper bounded by

$$\Pr[\text{COLL}_{\text{bot}}] = \frac{\ell(\ell-1)}{2} \cdot \epsilon \;\leq\; \frac{\ell^2}{2}\epsilon.$$

Hence, we can upper bound $\Pr[\text{COLL}] \;\leq\; \Pr[\text{COLL}_{\text{top}}] + \Pr[\text{COLL}_{\text{top}}] \leq \ell^2\epsilon$.

***Case 2:* NOCOLLWIN.** Next, we regard the case that $\mathcal{A}$ shall distinguish $(\mathsf{POE}_\pi, \mathsf{POE}^{-1}_{\pi^{-1}})$ from $(P(\cdot), P^{-1}(\cdot))$ when no internal collisions occur. We can generalize that each pair of tuples $(M, C), (M', C') \in \mathcal{Q}$ shares a common prefix of 0 to $\min(|M|, |M'|)/n$ blocks. Wlog., say that the pair $M, M' \in \mathcal{Q}$ shares an i-block common prefix, i.e., $M_j = M'_j$, $\forall j \in [1, i]$, and $M_{i+1} \neq M'_{i+1}$. In the following,

we study the difference in the behavior of POE and P for three subcases: (2.1) for the message blocks in the common prefix, $M_1, \ldots, M_i$, (2.2) for the $(i+1)$-th block, or (2.3) for the message blocks after the $(i+1)$-th one.

Subcase 2.1: Common Prefix. Since an OPERM is deterministic, input and output behaviors of $(\mathsf{POE}_\pi, \mathsf{POE}_{\pi^{-1}}^{-1})$ and $(P\cdot, P^{-1}(\cdot))$ are identical for the common prefix. Hence, the advantage for $\mathcal{A}$ in this subcase is 0.

Subcase 2.2: Directly After the Common Prefix. Since $M_j = M'_j, \forall j \in [1, i]$, it must hold in the real case that $Y_i = Y'_i$ and $X_i = X'_i$. From $M_{i+1} \neq M'_{i+1}$ follows

$$C_{i+1} = \pi(F_{K_1}(X_i) \oplus M_{i+1}) \oplus F_{K_2}(Y_i) \neq \pi(F_{K_1}(X'_i) \oplus M'_{i+1}) \oplus F_{K_2}(Y'_i) = C'_{i+1}.$$

Since π is a random permutation, C_{i+1}, C'_{i+1} are chosen uniformly at random in the real case. In the random case P is used with two different prefixes $M_1 \,||\, \ldots \,||\, M_{i+1}$ and $M'_1 \,||\, \ldots \,||\, M'_{i+1}$. Since P is an OPERM, $C_{i+1} \neq C'_{i+1}$ also must hold in this case. Hence, the advantage for $\mathcal{A}$ in this subcase is also 0.

Subcase 2.3: After the $(i+1)$***-th Message Block.*** In the random case, each query output is chosen uniformly at random from $\{0,1\}^n$. However, in the real world each output of either an encryption or a decryption query is chosen uniformly at random from the set $\{0,1\}^n \setminus \mathcal{Q}$. This means that in the real case POE loses randomness with every query. We can upper bound the success probability of an adversar to distinguish POE from a random OPERM by

$$\frac{\ell^2}{2^n - \ell}.$$

Our claim in Eq. (2) follows from summing up the individual terms. □

4 The On-line AE Scheme POET

For McOE, Fleischmann et al. [19] showed that an OPRP-CCA-secure on-line cipher can be easily transformed into an on-line AEAD scheme that is resistant against nonce and decryption misuse. This section shows how to apply their approach to transform POE into a nonce- misuse-resistant AE scheme for messages whose lengths are a multiple of the block length.

4.1 Definition of POET

Definition 7 (POET). *Let $k, n \geq 1$ be integers. Let $\mathsf{POET} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be an AE scheme, E: $\{0,1\}^k \times \{0,1\}^* \to \{0,1\}^*$ a block cipher, and F: $\{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$ a family of keyed ϵ-AXU hash functions. Furthermore, let F_i: $\{0,1\}^{ni} \to \{0,1\}^n$ be $i\epsilon$-AXU family of hash functions defined as follows:*

$$F_0 = F(1); \quad F_i(M) = F(F_{i-1}(M_1, \ldots, M_{i-1}) \oplus M_i) \qquad i \in \mathbb{N}^+.$$

Let H be the header (including the nonce as its final block), M the message, T the authentication tag, and C the ciphertext, with $H, M, C \in (\{0,1\}^n)^*$ and $T \in \{0,1\}^n$. Then, we define encryption and decryption algorithms of POET as shown in Algorithm 2.

Algorithm 2. Procedures **Encrypt** and **Decrypt** for POET.

Encrypt(H, M)
101: $X_0 \leftarrow Y_0 \leftarrow 1$, $m \leftarrow \frac{|M|}{n}$ $h \leftarrow \frac{|H|}{n}$
102: **for** $i \leftarrow 1, \ldots, h$ **do**
103: $\quad X_i \leftarrow F_{K_1}(X_{i-1}) \oplus H_i$
104: $\quad Y_i \leftarrow E_K(X_i)$
105: **end for**
106: $\tau \leftarrow F_{K_2}(Y_{h-1}) \oplus Y_h$
107: $M_m \leftarrow M_m \oplus E_K(|M|)$
108: **for** $i \leftarrow 1, \ldots, m$ **do**
109: $\quad j \leftarrow i + h$
110: $\quad X_j \leftarrow F_{K_1}(X_{j-1}) \oplus M_i$
111: $\quad Y_j \leftarrow E_K(X_j)$
112: $\quad C_i \leftarrow F_{K_2}(Y_{j-1}) \oplus Y_j$
113: **end for**
114: $j \leftarrow m + h$
115: $X_{j+1} \leftarrow F_{K_1}(X_j) \oplus \tau$
116: $T \leftarrow F_{K_2}(Y_j) \oplus E_K(X_{j+1})$
117: **return** $(C_1 \,||\, \ldots \,||\, C_m, T)$

Decrypt(H, C, T)
201: $X_0 \leftarrow Y_0 \leftarrow 1$, $m \leftarrow \frac{|C|}{n}$ $h \leftarrow \frac{|H|}{n}$
202: **for** $i \leftarrow 1, \ldots, h$ **do**
203: $\quad X_i \leftarrow F_{K_1}(X_{i-1}) \oplus H_i$
204: $\quad Y_i \leftarrow E_K(X_i)$
205: **end for**
206: $\tau \leftarrow F_{K_2}(Y_{h-1}) \oplus Y_h$
207: **for** $i \leftarrow 1, \ldots, m$ **do**
208: $\quad j \leftarrow i + h$
209: $\quad Y_j \leftarrow F_{K_2}(Y_{j-1}) \oplus C_i$
210: $\quad X_j \leftarrow E_K^{-1}(Y_j)$
211: $\quad M_i \leftarrow F_{K_1}(X_{j-1}) \oplus X_j$
212: **end for**
213: $M_m \leftarrow M_m \oplus E_K(|C|)$
214: $j \leftarrow m + h$
215: $X_{j+1} \leftarrow F_{K_1}(X_j) \oplus \tau$
216: $T' \leftarrow F_{K_2}(Y_j) \oplus E_K(X_{j+1})$
217: **if** $T = T'$ **then**
218: $\quad$ **return** $(M_1 \,||\, \ldots \,||\, M_m)$
219: **end if**
220: **return** $\perp$

A schematic illustration of the encryption algorithm is given in Fig. 2.

Remark 2. POET uses the common 10*-padding for headers $|H|$ whose length is not a multiple of n. As a result, H consists of at least a single block, and the entire header can be seen as a nonce. For messages whose length is not a multiple of the block size, POET *borrows* the provably secure tag-splitting approach from McOE [19]. Therefore, it is sufficient to prove the OCCA3-security *only* for messages whose length is a multiple of the block size.

4.2 Security Notions for On-line AE Schemes

We define an on-line authenticated encryption scheme Π to be OCCA3-secure iff it provides both OPRP-CPA *and* INT-CTXT security. Note that we explicitly regard nonce-ignoring adversaries which are allowed to use a nonce multiple times, similar to the security notions of integrity for authenticated encryption schemes in [19]. In the next part, we briefly revisit the formal definitions of INT-CTXT and OCCA3.

The INT-CTXT-advantage of an adversary $\mathcal{A}$ is given by the success probability of winning the game $G_{\mathsf{INT\text{-}CTXT}}$ that is defined in Fig. 3. Thus, we obtain

$$\mathbf{Adv}_{\Pi}^{\mathsf{INT\text{-}CTXT}}(\mathcal{A}) \leq \Pr\left[\mathcal{A}^{G_{\mathsf{INT\text{-}CTXT}}} \Rightarrow 1\right], \tag{5}$$

where $\mathbf{Adv}_{\Pi}^{\mathsf{INT\text{-}CTXT}}(q, \ell, t)$ is the maximum advantage over all INT-CTXT adversaries $\mathcal{A}$ that run in time at most t, and make at most q queries with a total length of at most ℓ blocks to the available oracles.

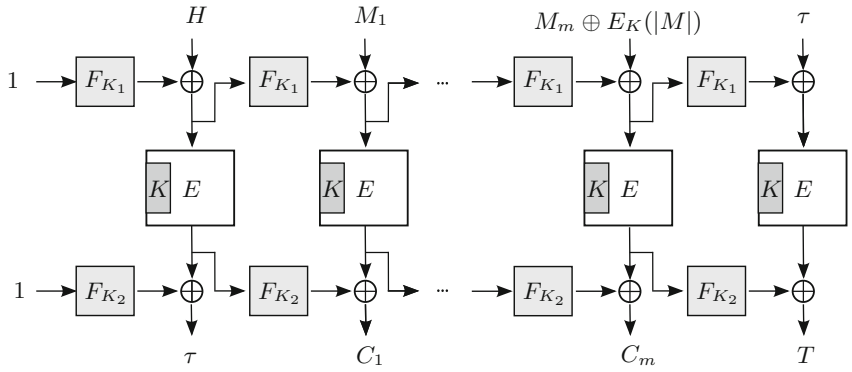


Fig. 2. The encryption process for an m-block message M of POET.

Definition 8 (OCCA3-Security). *Let $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be an on-line authenticated encryption scheme. Then, the OCCA3-advantage of an adversary $\mathcal{A}$ is upper bounded by*

$$\mathbf{Adv}^{\mathsf{OCCA3}}_{\Pi}(\mathcal{A}) \leq \mathbf{Adv}^{\mathsf{OPRP\text{-}CPA}}_{\Pi}(q, \ell, t) + \mathbf{Adv}^{\mathsf{INT\text{-}CTXT}}_{\Pi}(q, \ell, t). \tag{6}$$

The OCCA3-advantage of Π, $\mathbf{Adv}^{\mathsf{OCCA3}}_{\Pi}(q, \ell, t)$, is then defined by the maximum advantage of all adversaries $\mathcal{A}$ that run in time at most t, and make at most q queries of a total length of at most ℓ blocks to the available oracles.

Note that an OPRP-CPA-adversary $\mathcal{A}$ on some encryption scheme Γ can always be used by an OPRP-CCA-adversary $\mathcal{A}$' on Γ that inherits the advantage of $\mathcal{A}$. In reverse direction, an upper bound for the OPRP-CCA-advantage of Γ is always an upper bound for the OPRP-CPA-advantage of Γ.

4.3 OCCA3-Security of POET

Theorem 3. *Let $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be a POET scheme as defined in Definition 7. Then, it applies that*

$$\mathbf{Adv}^{\mathsf{OCCA3}}_{\Pi}(q, \ell, t) \leq 2\,(\ell + 2q)^2\,\epsilon + \frac{(\ell + 2q)^2 + q}{2^n - (\ell + 2q)} + 2\mathbf{Adv}^{\mathsf{IND\text{-}SPRP}}_{E, E^{-1}}(\ell + 2q, O(t)).$$

```
1  Initialize()
2  K ← K(); Q ← ∅;

3  Finalize()
4  return win;
```

```
10 Encrypt(H, M)
11 (C, T) ← E_K(H, M);
12 Q ← Q ∪ {(H, C)};
13 return (C, T);
```

```
20 Verify(H, C, T)
21 M ← D_K(H, C, T);
22 if ((H, C) ∉ Q and M ≠ ⊥) then
23    win ← true;
24 return (M ≠ ⊥);
```

Fig. 3. The $G_{\mathsf{INT\text{-}CTXT}}$ game for an authenticated encryption scheme $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$.

Proof. The proof follows from Theorem 2 and the bound for the INT-CTXT-security of POET. Due to the lack of space we omit the proof for the latter in this version and refer the reader to Lemma 1 in the full version of this paper [2]. Since Theorem 2 yields an upper bound for the OPRP-CCA-advantage on POE, it also provides an upper bound for the OPRP-CPA-advantage on POET. Though, ℓ (the number of encrypted message and header blocks from Theorem 2) must be replaced by $(\ell+2q)$ since the tag-generation process of POET includes two additional block-cipher calls per query. □

5 Key Derivation and Instantiations

5.1 Key Derivation

POE and POET require three internal keys: one key K for the block cipher, and two keys K_1 and K_2 for the two instances of F. Since our goal was to put no further restrictions on the used hash function families, we borrowed the idea from [25] to obtain pair-wise independent keys. At setup, the user supplies a k-bit secret key L. The further keys are then derived from L by encrypting three distinct constants $const_0$, $const_1$, $const_2$ with E:

$$K \leftarrow E_L(const_0), \qquad K_1 \leftarrow E_L(const_1), \qquad K_2 \leftarrow E_L(const_2).$$

For simplicity, we recommend $const_0 = 1, const_1 = 2, const_2 = 3$. Therefore, under the assumption that E is a PRP-secure block cipher, we can ensure to obtain independent keys for the block-cipher and hash-function calls.

5.2 ϵ-AXU Hash Functions

We recommend to instantiate POE/POET with AES-128 as block cipher. For the ϵ-AXU families of hash functions F, we propose three suitable instantiations in the following.

POE/POET ***with Four-Round AES.*** When trying to minimize the implementation footprint, it may be desirable to have an encryption scheme based on only a single primitive. Furthermore, maximizing the throughput is often critical. Therefore, POE/POET with the first four rounds of the AES as a family of keyed hash functions may be an excellent choice for restricted devices and/or devices with support for AES native instructions. The drawback of this solution would be a slightly lower number than the common 2^{64} message blocks that can be processed under the same key. As shown by Daemen et al. in [14], four-round AES is a family of ϵ-AXU hash functions—under the reasonable assumption that all used round keys are independent—with

$$\epsilon \leq 1.88 \cdot 2^{-114} \approx 2^{-113}.$$

This implies that at most $\ll 2^{56}$ message blocks can be encrypted or decrypted under the same key.

POE/POET ***with Full-Round AES.*** As a more conservative variant we propose the full AES-128 for the family of hash functions. Under the common PRF assumption—where we assume that AES is indistinguishable from a random 128-bit permutation, this constructions yields $\epsilon \approx 2^{-128}$.

POE/POET ***with Galois-Field Multiplications.*** In addition, one can use a multiplication in $GF(2^{128})$, similar to that in AES-GCM [31], as a universal hash function. This approach yields an $\epsilon \approx 2^{-128}$. Moreover, POE and POET can be fully parallelized with Galois-Field multiplications. For instance, consider a message of at least four blocks, $M_1 \,||\, \ldots \,||\, M_4$. Using Galois- Field multiplications, the input for the second block-cipher call is $K^2 + KM_1 + M_2$. Instead of sequentially multiplying with K, adding M_3, multiplying with K and adding M_4, one can compute in parallel:

- For the third block-cipher call: $K \cdot (K^2 + KM_1 + M_2) + M_3$.
- For the fourth block-cipher call: $K^2 \cdot (K^2 + KM_1 + M_2) + KM_3 + M_4$.

This approach increases the total number of multiplications, but decreases the latency. Given c cores, and c subsequent message blocks to process, this approach reduces the latency from c hash-function calls to $O(\log c)$. This approach is used, e.g., in carry-lookahead adders, GCM [31], or CWC [27].

When using multiplications in $GF(2^{128})$, one has to consider the risk of weak keys and forgery polynomials. At FSE'12 Saarinen [42] pointed out that, since $2^{128} - 1$ is not prime and produces 2^9 smooth- order multiplicative groups, one can obtain a weak key with probability 2^{-96} that allows to efficiently construct a forgery. Saarinen's observation was generalized by Procter and Cid at FSE'13 [39] who showed that an adversary can choose an arbitrary message as a polynomial $q(x)$ with a preferably high degree and no repeated roots. Then, it can create two messages M, M' that collide with $p = \frac{\#\text{roots of } q(x)}{2^{128}}$. As a result of their work, any key can be considered potentially weak. After the FSE'14, Abdelraheem et al. [1] applied the observations of Procter and Cid to the version of POET that was submitted to the CAESAR competition, and showed that one could build forgeries for POET with Galois-Field multiplication with success probability between 2^{-96} and 2^{-66}. Therefore, we recommend to use (round-reduced) AES for hashing in POET in favor to a Galois-Field multiplication.

6 Conclusion

This paper presented POE, the first family of on-line ciphers which is both non-sequential *and* provably OPRP-CCA-secure. Its design combines two layers of ϵ-AXU hashing and a wrapped layer of ECB encryption.

Most on-line AE schemes have a significant latency since they must buffer a would-be plaintext until the tag has been been verified. The latency can be significantly decreased when the would-be plaintext is passed beforehand – however, this approach raises security issues when applied to AE schemes that lack OPRP-CCA-security, i.e., an adversary could obtain partial control about the would-be plaintext, even when these include additional checksums. On the other hand, previous

OPRP-CCA-secure encryption schemes were inherently sequential. POE is well-suited for high-speed networks that require performant, low-latency encryption of large message frames, especially when classical authenticated decryption would increase latency significantly. Our application scenario targets optical transport networks (OTNs), but the latency imposed by authenticated decryption is an issue for other applications as well. In general, POE is an option for such applications.

We proposed three instantiations, where we recommended the AES as block cipher and either four-round AES, full AES, or a multiplication in $GF(2^{128})$ as ϵ-AXU families of hash functions. Additionally, we presented POET, a state-of-the-art on-line authenticated encryption scheme, which inherits the chosen-ciphertext-security and pipelineability from POE. Concluding, POET combines pipelineability with misuse-resistance in a novel way, at the cost of only a single block-cipher and two additional hash-function calls per message block.

Acknowledgments. We thank all reviewers of the FSE 2014 for their helpful comments and Daniel J. Bernstein and Tetsu Iwata for fruitful discussions. Finally, we thank Jian Guo, Jérémy Jean, Thomas Peyrin, and Lei Wang who pointed out a mismatch between the specified and the analyzed version of POET in the pre-proceedings version [20].

References

1. Abdelraheem, M.A., Bogdanov, A., Tischhauser, E.: Weak-key analysis of POET. Cryptology ePrint Archive, Report 2014/226 (2014). http://eprint.iacr.org/
2. Abed, F., Fluhrer, S., Forler, C., List, E., Lucks, S., McGrew, D., Wenzel, J.: Pipelineable on-line encryption. Cryptology ePrint Archive, Report 2014/297 (2014). http://eprint.iacr.org/
3. Andreeva, E., Bogdanov, A., Luykx, A., Mennink, B., Tischhauser, E., Yasuda, K.: Parallelizable and authenticated online ciphers. In: Sako, K., Sarkar, P. (eds.) ASIACRYPT 2013, Part I. LNCS, vol. 8269, pp. 424–443. Springer, Heidelberg (2013)
4. Andreeva, E., Bogdanov, A., Luykx, A., Mennink, B., Mouha, N., Yasuda, K.: APE(X): authenticated permutation-based encryption with extended security features. In: Directions in Authenticated Ciphers (2013)
5. Bellare, M., Boldyreva, A., Knudsen, L.R., Namprempre, C.: Online ciphers and the hash-CBC construction. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 292–309. Springer, Heidelberg (2001)
6. Bellare, M., Boldyreva, A., Knudsen, L.R., Namprempre, C.: On-line ciphers and the hash-CBC constructions. J. Cryptol. **25**(4), 640–679 (2012)
7. Bellare, M., Namprempre, C.: Authenticated encryption: relations among notions and analysis of the generic composition paradigm. In: Okamoto, T. (ed.) ASIACRYPT 2000. LNCS, vol. 1976, pp. 531–545. Springer, Heidelberg (2000)
8. Bellare, M., Rogaway, P., Wagner, D.: The EAX mode of operation. In: Roy, B., Meier, W. (eds.) FSE 2004. LNCS, vol. 3017, pp. 389–407. Springer, Heidelberg (2004)
9. Bertoni, G., Daemen, J., Peeters, M., Van Assche, G.: Duplexing the sponge: single-pass authenticated encryption and other applications. In: Miri, A., Vaudenay, S. (eds.) SAC 2011. LNCS, vol. 7118, pp. 320–337. Springer, Heidelberg (2012)

10. Boesgaard, M., Christensen, T., Zenner, E.: Badger—a fast and provably secure MAC. In: Ioannidis, J., Keromytis, A.D., Yung, M. (eds.) ACNS 2005. LNCS, vol. 3531, pp. 176–191. Springer, Heidelberg (2005)
11. Boldyreva, A.,Taesombut, N.: Online encryption schemes: new security notions and constructions. In: Okamoto [35], pp. 1–14
12. Campbell, C.: Design and specification of cryptographic capabilities. In: Proceedings of the Conference on Computer Security and the Data Encryption Standard Held at the National Bureau of Standards in Gaithersburg, NBS Special Publication, Gaithersburg, Md., February 1978. U.S. National Bureau of Standards
13. Carter, L., Wegman, M.N.: Universal classes of hash functions. J. Comput. Syst. Sci. **18**(2), 143–154 (1979)
14. Daemen, J., Lamberger, M., Pramstaller, N., Rijmen, V., Vercauteren, F.: Computational aspects of the expected differential probability of 4-round AES and AES-like ciphers. Computing **85**(1–2), 85–104 (2009)
15. Datta, N., Nandi, M.: Misuse resistant parallel authenticated encryptions. Cryptology ePrint Archive, Report 2013/767 (2013). http://eprint.iacr.org/
16. Datta, N., Nandi, M.: Characterization of EME with linear mixing. Cryptology ePrint Archive, Report 2014/009 (2014). http://eprint.iacr.org/
17. Diffie, W., Hellman, M.E.: Privacy and authentication: an introduction to cryptography. Proc. IEEE **67**, 397–427 (1979). (Invited Paper)
18. Dolev, D., Dwork, C., Naor, M.: Nonmalleable cryptography. SIAM J. Comput. **30**(2), 391–437 (2000)
19. Fleischmann, E., Forler, C., Lucks, S.: McOE: a family of almost foolproof on-line authenticated encryption schemes. In: Canteaut, A. (ed.) FSE 2012. LNCS, vol. 7549, pp. 196–215. Springer, Heidelberg (2012)
20. Guo, J., Jean, J., Peyrin, T., Lei, W.: Breaking POET authentication with a single query. Cryptology ePrint Archive, Report 2014/197 (2014). http://eprint.iacr.org/
21. Halevi, S., Rogaway, P.: A tweakable enciphering mode. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 482–499. Springer, Heidelberg (2003)
22. Halevi, S., Rogaway, P.: A parallelizable enciphering mode. In: Okamoto [35], pp. 292–304
23. IEEE. IEEE Standard for Cryptographic Protection of Data on Block-Oriented Storage Devices. IEEE Std. 1619–2007, pp. c1–32 (2008)
24. ITU-T. Interfaces for the Optical Transport Network (OTN). Recommendation G.709/Y.1331, International Telecommunication Union, Geneva, December 2009
25. Iwata, T., Kurosawa, K.: OMAC: one-key CBC MAC. In: Johansson, T. (ed.) FSE 2003. LNCS, vol. 2887, pp. 129–153. Springer, Heidelberg (2003)
26. Knudsen, L.R.: Block chaining modes of operation. In: Symmetric-Key Block-Cipher Modes of Operation Workshop, October 2000
27. Kohno, T., Viega, J., Whiting, D.: CWC: a high-performance conventional authenticated encryption mode. In: Roy, B., Meier, W. (eds.) FSE 2004. LNCS, vol. 3017, pp. 408–426. Springer, Heidelberg (2004)
28. Krovetz, T., Rogaway, P.: The software performance of authenticated-encryption modes. In: Joux, A. (ed.) FSE 2011. LNCS, vol. 6733, pp. 306–327. Springer, Heidelberg (2011)
29. Liskov, M., Rivest, R.L., Wagner, D.: Tweakable block ciphers. In: Yung, M. (ed.) CRYPTO 2002. LNCS, vol. 2442, pp. 31–46. Springer, Heidelberg (2002)
30. Liskov, M., Rivest, R.L., Wagner, D.: Tweakable block ciphers. J. Cryptol. **24**(3), 588–613 (2011)

31. McGrew, D., Viega, J.: The Galois/Counter Mode of Operation (GCM). Submission to NIST (2004). http://csrc.nist.gov/CryptoToolkit/modes/proposedmodes/gcm/gcm-spec.pdf
32. Nandi, M.: A Simple Security Analysis of Hash-CBC and a New Efficient One-Key Online Cipher. Cryptology ePrint Archive, Report 2007/158 (2007). http://eprint.iacr.org/
33. Nandi, M.: Two New efficient CCA-secure online ciphers: MHCBC and MCBC. In: Chowdhury, D.R., Rijmen, V., Das, A. (eds.) INDOCRYPT 2008. LNCS, vol. 5365, pp. 350–362. Springer, Heidelberg (2008)
34. US Department of Commerce. DES Modes of Operation. Technical report FIPS PUB 81, US Department of Commerce/National Bureau of Standards, December 1998
35. Okamoto, T. (ed.): CT-RSA 2004. LNCS, vol. 2964. Springer, Heidelberg (2004)
36. Postel, J.: User Datagram Protocol. RFC 768 (INTERNET STANDARD), August 1980
37. Postel, J.: Internet Protocol. RFC 791 (INTERNET STANDARD), September 1981. (Updated by RFCs 1349, 2474, 6864)
38. Postel, J.: Transmission Control Protocol. RFC 793 (INTERNET STANDARD), September 1981. (Updated by RFCs 1122, 3168, 6093, 6528)
39. Procter, G., Cid, C.: On weak keys and forgery attacks against polynomial-based MAC schemes. In: Moriai, S. (ed.) FSE 2013. LNCS, vol. 8424, pp. 287–304. Springer, Heidelberg (2014)
40. Rogaway, P., Shrimpton, T.: A provable-security treatment of the key-wrap problem. In: Vaudenay, S. (ed.) EUROCRYPT 2006. LNCS, vol. 4004, pp. 373–390. Springer, Heidelberg (2006)
41. Rogaway, P., Zhang, H.: Online ciphers from tweakable blockciphers. In: Kiayias, A. (ed.) CT-RSA 2011. LNCS, vol. 6558, pp. 237–249. Springer, Heidelberg (2011)
42. Saarinen, M.-J.O.: Cycling attacks on GCM, GHASH and other polynomial MACs and hashes. In: Canteaut, A. (ed.) FSE 2012. LNCS, vol. 7549, pp. 216–225. Springer, Heidelberg (2012)
43. Wegman, M.N., Carter, J.L.: New hash functions and their use in authentication and set equality. J. Comput. Syst. Sci. **22**(3), 265–279 (1981)
44. Young, E.A., Hudson, T.J.: OpenSSL: The Open Source toolkit for SSL/TLS, September 2011. http://www.openssl.org/

A Observations on COPE

COPE is a parallelizable on-line cipher designed by Andreeva et al. [3] and is the underlying construction of the AE scheme COPA. A formal description is given as follows. Let $E \in \texttt{Block}$ and a fixed key $K \in \{0,1\}^k$. Then, COPE and its inverse are defined as shown in Algorithm 3.

Algorithm 3. Definition of COPE following [3].

Encrypt(M)	**Decrypt**(C)
$L \leftarrow E_K(0), \Delta_0 \leftarrow 3L, \Delta_1 \leftarrow 2L$	$L \leftarrow E_K(0), \Delta_0 \leftarrow 3L, \Delta_1 \leftarrow 2L$
$Y_0 \leftarrow L$	$X_0 \leftarrow L$
for $i = 1, \ldots, \ell$ **do**	**for** $i = 1, \ldots, \ell$ **do**
$\quad X_i \leftarrow E_K(M_i \oplus \Delta_0)$	$\quad Y_i \leftarrow E_K^{-1}(C_i \oplus \Delta_1)$
$\quad Y_i \leftarrow X_i \oplus Y_{i-1}$	$\quad X_i \leftarrow Y_i \oplus X_{i-1}$
$\quad C_i \leftarrow E_K(Y_i) \oplus \Delta_1$	$\quad M_i \leftarrow E_K^{-1}(X_i) \oplus \Delta_0$
$\quad \Delta_0 \leftarrow 2\Delta_0, \Delta_1 \leftarrow 2\Delta_1$	$\quad \Delta_0 \leftarrow 2\Delta_0, \Delta_1 \leftarrow 2\Delta_1$
end for	**end for**
return $(C_1 \,\|\| \ldots \|\| C_\ell)$	**return** $(M_1 \,\|\| \ldots \|\| M_\ell)$

In the following we show that COPE is not OPRP-CCA-secure.

OPRP-CCA-*Attack.* Let $\mathcal{A}$ be an OPRP-CCA adversary that communicates with two oracles $\mathcal{E}_K$ and $\mathcal{D}_K$. Let $M_a \neq M_b$ two distinct message blocks. Then, we denote $Y_a = E_K(M_a \oplus \Delta_0) \oplus L$ and $Y_b = E_K(M_b \oplus \Delta_0) \oplus L$.

1. First, $\mathcal{A}$ sends the encryption query (M_a, M_c) to $\mathcal{E}$, which responds with $(C_a, C_{(a,c)})$. In the "real" setting, it holds that
$$X_c = E_K(M_c \oplus 2\Delta_0), \qquad Y_{(a,c)} = Y_a \oplus X_c, \qquad C_{(a,c)} = E_K(Y_{(a,c)} \oplus 2\Delta_1).$$
2. Next, $\mathcal{A}$ requests the encryption of (M_b, M_c) and obtains $(C_b, C_{(b,c)})$. It holds that
$$X_c = E_K(M_c \oplus 2\Delta_0), \qquad Y_{(b,c)} = Y_b \oplus X_c, \qquad C_{(b,c)} = E_K(Y_{(b,c)} \oplus 2\Delta_1).$$
3. Then, $\mathcal{A}$ requests the decryption of the tuple $(C_a, C_{(b,c)})$, and $\mathcal{D}$ responds with $(M_a, M_{(a,bc)})$.
$$Y_{(b,c)} = E_K^{-1}(C_{(b,c)} \oplus 2\Delta_1), \quad X_{(a,bc)} = Y_{(b,c)} \oplus Y_a = Y_b \oplus X_c \oplus Y_a.$$
4. Finally, $\mathcal{A}$ sends the decryption query $(C_b, C_{(a,c)})$ and obtains $(M_b, M_{(b,ac)})$. It applies that
$$Y_{(a,c)} = E_K^{-1}(C_{(a,c)} \oplus 2\Delta_1), \quad X_{(b,ac)} = Y_{(a,c)} \oplus Y_b = Y_a \oplus X_c \oplus Y_b = X_{(a,bc)}.$$

From $X_{(a,bc)} = X_{(b,ac)}$ follows that $M_{(a,bc)} = M_{(b,ac)}$ in the real case. Hence, $\mathcal{A}$ returns `true` if $M_{(a,bc)} = M_{(b,ac)}$ and `false` otherwise, and can distinguish COPE from a random OPERM with probability $1 - 2^{-n}$.

Discussion. Our observation is also applicable to the on-line cipher of ELmE – or more generally to any on-line EME cipher with linear mixing layer. We want to stress that the shown attack does not invalidate any of the stated security claims of COPE or ELmE. However, our observation points out the importance of keeping the would-be plaintexts secret—otherwise, linear EME schemes can neither protect the data privacy of messages anymore. Therefore, one can not consider such designs secure in the decryption-misuse setting, which makes them a suboptimal choice for high-speed networks with low-latency requirements. Nevertheless, it remains an open research question if this property is undesired for further practical use cases.

Cryptanalysis of FIDES

Itai Dinur[1] and Jérémy Jean[2(✉)]

[1] École Normale Supérieure, Paris, France
Itai.Dinur@ens.fr
[2] Nanyang Technological University, Singapore, Singapore
Jeremy.Jean@ens.fr

Abstract. FIDES is a lightweight authenticated cipher, presented at CHES 2013. The cipher has two version, providing either 80-bit or 96-bit security. In this paper, we describe internal state-recovery attacks on both versions of FIDES, and show that once we recover the internal state, we can use it to immediately forge *any* message. Our attacks are based on a guess-and-determine algorithm, exploiting the slow diffusion of the internal linear transformation of FIDES. The attacks have time complexities of 2^{75} and 2^{90} for FIDES-80 and FIDES-96, respectively, use a very small amount of memory, and their most distinctive feature is their very low data complexity: the attacks require at most 24 bytes of an arbitrary plaintext and its corresponding ciphertext, in order to break the cipher with probability 1.

Keywords: Authenticated encryption · FIDES · Cryptanalysis · Guess-and-determine

1 Introduction

The design and analysis of authenticated encryption primitives have recently become major research areas in cryptography, mostly driven by the NIST-funded CAESAR competition for authenticated encryption [6]. At CHES 2013, the new lightweight authenticated cipher FIDES was proposed by Bilgin et al. [2], providing an online single-pass nonce-based authenticated encryption algorithm. The cipher claims to simultaneously maintain a highly competitive footprint and a time-efficient implementation.

The cipher has two versions, FIDES-80 and FIDES-96, which have similar designs, but differ according to their key sizes, and thus according to the security level they provide. For each version, the same security level (80 bits for FIDES-80 and 96 bits FIDES-96) is claimed against all key recovery, internal state recovery, and forgery attacks, under the assumption that the attacker cannot obtain the encryptions of two different messages with the same key/nonce pair.

J. Jean—Supported by the Singapore National Research Foundation Fellowship 2012 (NRF-NRFF2012-06).

C. Cid and C. Rechberger (Eds.): FSE 2014, LNCS 8540, pp. 224–240, 2015.
DOI: 10.1007/978-3-662-46706-0_12

The structure of FIDES is similar to the duplex sponge construction [1], having a secret internal state, where the encryption/authentication process alternates between input of message blocks and applications of unkeyed permutations to the state. The computation of the ciphertext is based on the notion of *leak-extraction*, formalized in the design document of the stream cipher LEX [3]. Namely, between the applications of the permutations, parts of the secret internal state are extracted (leaked), and used as a key-stream which is XORed with the plaintext to produce the ciphertext. The notion of leak-extraction was also borrowed by ALE, which (similarly to FIDES) is another recently proposed authenticated encryption primitive, presented at FSE 2013 [4]. However, despite the novelty of the leak-extraction idea, it is quite risky. Indeed, both the LEX stream cipher and ALE were broken using differential cryptanalysis techniques that exploit the leakage data available to the attacker [5,9–11].

The main idea in differential attacks on standard iterated block ciphers is to find a differential characteristic which covers most rounds of the cipher, and gives the ability to distinguish them from a random function. Once the data has been collected, the key of the cipher can be recovered using a guess-and-determine algorithm: we guess a partial round subkey and exploit the limited diffusion of its last few rounds in order to partially decrypt the ciphertexts and verify the guess using the distinguisher.

In order to avoid differential distinguishers on FIDES, its internal non-linear components (i.e., its S-Boxes) were carefully chosen to offer optimal resistance against differential attacks. Indeed, as the authors of FIDES dedicate a large portion of the design document to analyze its resistance against differential cryptanalysis, it is clear that this consideration played a crucial role in the design of FIDES. On the other hand, no analysis is given as to the strength of FIDES against "pure" non-statistical guess-and-determine attacks. Seemingly, this is not required, as modern block ciphers are typically designed using many iterative rounds, providing sufficient diffusion. This ensures that guess-and-determine attacks can only penetrate a small fraction of the rounds, and thus such attacks on block ciphers are quite rare.

Although most block ciphers do not require special countermeasures against guess-and-determine attacks, FIDES is an authenticated cipher based on leak-extraction and is therefore far from a typical block cipher. Indeed, since the attacker obtains partial information on the internal state during the encryption process, such schemes need to be designed very carefully in order to avoid state-recovery attacks. In the case of FIDES, the designers chose a linear transformation with non-optimal diffusion due to efficiency considerations, exposing its internal state even further to guess-and-determine attacks.

In this paper, we show how to exploit the weakness in the linear transformation of FIDES in order to mount guess-and-determine state-recovery attacks on both of its versions: we start by guessing a relatively small number of internal state variables, and the slow diffusion of the linear transformation enables us to propagate this limited knowledge in order to calculate other variables, eventually recovering the full state.

Our state-recovery attacks clearly contradict the security claims of the designers regarding the resistance of the cipher against such attacks. However, a state-recovery attack on an authenticated cipher does not directly compromise the security of its users. Nevertheless, as we show in this paper, once we obtain its internal state, two additional design properties of FIDES immediately allow us to mount a forgery attack and forge *any* message. Thus, in the case of FIDES, the resistance against state-recovery attacks is crucial to the security of its users.

The most simple state-recovery attacks we present in this paper are "pure" guess-and-determine attacks which do not involve any statistical analysis that requires a large amount of data in order to distinguish the correct guess from the incorrect ones. As a result, the attacks are very close to the *unicity bound*, i.e., they require no more than 24 bytes of an arbitrary plaintext and its corresponding ciphertext in order to fully recover the state (and thus forge any message) faster than exhaustive search, using a very small amount of memory. More specifically, for FIDES-80, our basic attack has a time complexity of 2^{75} computations (compared to 2^{80} for exhaustive search) and a memory complexity of 2^{15}, and for FIDES-96, our basic attack has a time complexity of 2^{90} computations (compared to 2^{96} for exhaustive search) and a memory complexity of 2^{18}.

In addition to the basic attacks described in this paper, we also provide optimized attacks in the extended version [8] of this paper, which allow to mount faster state-recovery attacks, by exploiting t-way collisions on the output that exist when we can collect more data. In particular, we show how to recover the internal state and thus forge messages in reduced time complexities of 2^{73} and 2^{88} computations for FIDES-80 and FIDES-96, respectively. A summary of these attacks is given in Table 1.

While the idea of the basic guess-and-determine attack is very simple, finding such attacks is a highly non-trivial task. Indeed, our attack includes several phases in which we guess the value of a subset of variables and propagate the information to another set of variables. In some of these phases, the information cannot be propagated using simple relations (directly derived from the FIDES internal round function), but is rather propagated in a complex way using meet-in-the-middle algorithms that exploit pre-computed look-up tables. It is therefore clear that the search space for such multi-phase attacks is huge. Luckily, we could use the publicly-available automated tool of [5], which was especially designed in order to aid searching for efficient attacks of this type. However, as it is mostly the case with such generic tools, we had to fine-tune it using many trials in which we artificially added external constraints in order reduce the search space and eventually find an efficient attack.

The rest of the paper is organized as follows. In Sect. 2, we give a brief description of FIDES, and in Sect. 3, we describe its design properties that we exploit in our attacks. In particular, we show in this section how to utilize any state-recovery attack in order to forge arbitrary messages. In Sect. 4, we give an overview of our basic state-recovery attack, and describe its details in Sect. 5. Finally, we conclude in Sect. 6.

Table 1. Summary of our state-recovery/forgery attacks

Cipher	Time	Data	Memory	Reference
FIDES-80	2^{75}	1 KP	2^{15}	Section 4
FIDES-96	2^{90}	1 KP	2^{18}	Section 4
FIDES-80	2^{73}	2^{64} KP	2^{64}	Extended version [8]
FIDES-96	2^{88}	2^{77} KP	2^{77}	Extended version [8]

KP: Known plaintext.

2 Description of FIDES

The lightweight authenticated cipher FIDES [2] was published at CHES 2013 by Bilgin et al. It uses a secret key K and a public nonce N in order to encrypt and authenticate a message M into the ciphertext C, and optionally authenticate at the same time some padded associated data A. At the end of the encryption process, an authentication tag T is generated and transmitted in the clear, along with C and N, for decryption by the other party.

FIDES comes in two versions: FIDES-80 and FIDES-96, having similar designs, but providing different levels of security. These versions are characterized by an internal nibble (word) size of c bits, where $c = 5$ in FIDES-80 and $c = 6$ in FIDES-96. The key K of FIDES is of size $16c$ bits (80 bits for FIDES-80 and 96 bits for FIDES-96), and similarly, the nonce N and the tag T are also $16c$-bit strings.

Internal State. The design of FIDES is influenced by the AES [7]. Its internal state X is represented as a matrix of 4×8 nibbles of c bits, where $X[i, j]$ denotes the nibble located at row $i \in \{0, 1, 2, 3\}$ and column $j \in \{0, 1, 2, 3, 4, 5, 6, 7\}$.

The encryption process of FIDES has three phases, as described below.

Initialization. The state is first initialized with the $16c$-bit secret key K, concatenated with a $16c$-bit nonce N. Then, the FIDES round function is applied 16 times to the state. Finally, K is XORed again into the left half of the state (columns 0, 1, 2 and 3).

Encryption/Authentication Process. After the initialization phase, the associated data is processed[1] and the message is encrypted in blocks of $2c$ bits. In order to encrypt a plaintext block, the two nibbles $X[3, 0]$ and $X[3, 2]$ (see Fig. 1) are extracted and XORed with the current plaintext block to produce the corresponding ciphertext block. Then, the c-bit halves of the (original) plaintext block are XORed into $X[3, 0]$ and $X[3, 2]$, respectively. Finally, the round function is applied.

[1] Since our attacks do not use any associated data, we do not elaborate on its processing.

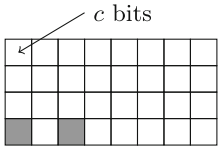


Fig. 1. The $32c$-bit internal state of FIDES, where $X[3,0]$ and $X[3,2]$ act as input/output during the encryption process.

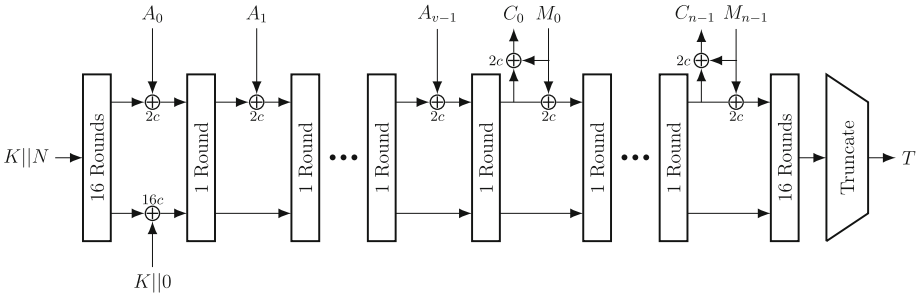


Fig. 2. The encryption/authentication process of FIDES.

Finalization. After all the message blocks have been processed, the round function is applied 16 more times to the internal state, and finally its left half (columns 0, 1, 2 and 3) is outputted as the tag T.

The full encryption/authentication process in visualized in Fig. 2. We note that in order to decrypt, a similar process is performed, where the ciphertext is XORed with the leaked nibbles in order to decrypt and obtain the message block, which is then XORed into the state. Finally, the tag is calculated and validated against the one received.

Description of the Round Function. The round function of FIDES uses AES-like transformations (see Fig. 3).

At the beginning of round i, the two nibbles of the message block M_i are processed and injected to produce the state X_i. The SubBytes (SB) transformation applies a non-linear S-Box S to each nibble of the state X_i independently, and the ShiftRows (SR) transformation rotates the r'th row by $\tau[r]$ positions to the left, where $\tau = [0, 1, 2, 7]$. This produces a state that we denote by Y_i. The state is then updated to the state W_i by applying the linear transformation MixColumns (MC), which left-multiplies each column of Y_i independently by the binary matrix $\mathbf{M}$:

$$\mathbf{M} = \begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix}.$$

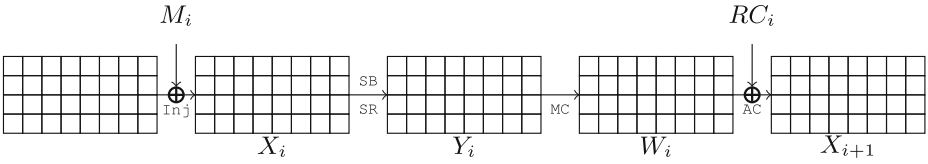


Fig. 3. The round function of FIDES.

Finally, the AddConstant (AC) transformation XORs a 32-nibble round-dependent constant RC_i (where $RC_i[\ell, j]$ denotes the nibble at position (ℓ, j)) into the state W_i to produce the initial state of the next round, X_{i+1}. Since we assume that both the round constants and the message blocks are known to us, we can obtain an equivalent scheme by removing the message injections and "embedding" them to the round constants, XORed into the state at the end of the previous round. Thus, for the sake of simplicity, we ignore the message injections in the rest of this paper.

We note that since our attack is *structural*, it is independent of the particular choices of S-boxes and round-constants of FIDES. Thus, we omit their description, which can be found in [2].[2]

3 Design Properties of FIDES Exploited in Our Attacks

In this section, we emphasize the properties of FIDES that we exploit in our attacks. First, we describe two basic linear properties of the round function that are extensively used in our state-recovery attack. Then, we describe two design properties of FIDES, and use them to show that any state-recovery attack immediately enables the attacker to forge any message.

3.1 Properties of the MixColumns Transformation

The binary matrix $\mathbf{M}$ (that defines the MixColumns transformation) has a *branch number* of 4. This implies that there are linear dependencies between 4 nibbles of $\mathbf{x}$ and $\mathbf{y} = \mathbf{Mx}$ (where $\mathbf{x} = [x_0, x_1, x_2, x_3]$ and $\mathbf{y} = [y_0, y_1, y_2, y_3]$):

Property 1. For all $i, j \in \{0, 1, 2, 3\}$ such that $i \neq j$: $x_i \oplus x_j = y_i \oplus y_j$.

Property 2. For all $i \in \{0, 1, 2, 3\}$: $x_{i+3} = y_i \oplus x_{i+1} \oplus x_{i+2}$ (where addition is performed modulo 4), and (analogously): $y_{i+3} = x_i \oplus y_{i+1} \oplus y_{i+2}$.

Such equalities are extremely useful in guess-and-determine attacks on AES-based schemes, where the attacker guesses a few internal nibbles of various states and tries to determine the values of as many nibbles as possible in order to verify his guesses. Indeed, as the branch number of $\mathbf{M}$ is 4, it is possible to determine the value of an unknown nibble of $\mathbf{x}$ or $\mathbf{y}$, given the values of only 3 out of the 4 nibbles in an equation above.

[2] In fact, the round-constants are also not defined in the specifications.

We note that the maximal possible branch number for a 4×4 matrix is 5 (the AES MixColumns transformation was especially designed to have this property). Interestingly, the matrix $\mathbf{M}$ of FIDES was not be designed to have the maximal branch number due to implementation efficiency considerations. As we demonstrate in our attack, this is a significant design-flaw. Indeed, we use the two properties above more than 150 times in order to mount a state-recovery attack which is faster than exhaustive search.

3.2 Properties Exploited in Forgery Attacks

In this section, we show that a state-recovery attack enables the attacker to forge any message. This is a result of two design properties (refer to Sect. 2 for details):

Property 3. The initial internal state of FIDES is computed using a secret key K and a public nonce N, and does not depend on the encrypted message.

Property 4. Once the internal state has been recovered, the rest of the computation (including the tag generation process) does not depend on K, and can be fully simulated for any message.

As a result of Property 3, once we recover the internal state generated by one (K, N) pair in the encryption process of an arbitrary plaintext M, we can immediately deduce it for the encryption of any other plaintext M', encrypted using the same (K, N) pair. Combined with Property 4, a state-recovery attack therefore enables to immediately forge *any* message by simulating the encryption process and computing the produced tag.

We note that the design of FIDES places a restriction on the encryption device, such that it cannot send two different messages with the same (K, N) pair. However, the ciphertexts decrypted by the decryption device are not restricted in such a way, namely, the device is allowed to decrypt two ciphertexts with the same (K, N) pair (assuming that their tag is valid). Thus, our attacks are applicable in the weak known plaintext model, and do not require advanced capabilities (such as intercepting messages, required in man-in-the-middle attacks).

4 Overview of the State-Recovery Attack

In this section, we give an overview of our state-recovery attack on both versions of FIDES, distinguished by the nibble size of c bits. As any meaningful attack must be more efficient than exhaustive search, we first formalize the state-recovery problem for FIDES and analyze the simple exhaustive search algorithm.

The State-Recovery Problem and Exhaustive Search. The input to the state-recovery problem is a message M, its corresponding ciphertext C, encrypted using a key/nonce pair (K, N), and the actual value of the nonce N.[3]

[3] One may also include the tag of the message in the inputs to the state-recovery problem, however, we do not require it.

The goal of this problem is to recover X_0, which denotes the 32 nibbles of the initial state obtained after the initialization of FIDES with the (K, N) pair. In order to recover the initial state, it is possible to exhaustively enumerate the 2^{32c} possibilities for X_0 and check if each one of them encrypts[4] M to C. However, a much more efficient exhaustive search procedure is to enumerate all the 2^{16c} possibilities for the key. Since the nonce is known, one executes the initialization procedure of FIDES for each value of the key, obtains a suggestion for X_0, and then uses it to verify that M is indeed encrypted to C.

Complexity Evaluation of Our Attack. As shown above, the time complexity of (efficient) exhaustive search for X_0 is about 2^{16c} iterations (or time-units), where in each iteration, FIDES is initialized using 16 round function evaluations (and additional few rounds in order to verify that M is indeed encrypted to C). As described in the detailed attack (Sect. 5), compared to exhaustive search, the time complexity of our state-recovery attack is only 2^{15c} time-units. Moreover, in each such time-unit, we perform computations on c-bit nibbles that are equivalent to only about nine FIDES round function evaluations (in addition to a few memory look-ups). However, as this smaller time-unit does not give our attack an additional significant advantage over exhaustive search, we ignore it in the remainder of this paper, and assume for the sake of simplicity that our attack uses the exhaustive search time-unit.

In terms of memory, the basic unit that we use contains 32 nibbles, which is the size of the FIDES internal state.

4.1 The Main Procedure of Our Attack

The attack uses the knowledge of a single 9-block known-plaintext message $M_0||\cdots||M_8$, and we denote by $C_0||\cdots||C_8$ the associated ciphertext. Thus, according to the design of FIDES (see (Sect. 2), we have the knowledge of two nibbles of c bits in 9 consecutive internal states $X_0, \ldots, X_8$, linked by 8 rounds. The attack enumerates in 2^{15c} computations the expected number of $2^{(32-2\times 9)\times c} = 2^{14c}$ valid states (i.e., solutions) which can possibly produce $C_0||\cdots||C_8$. By using additional output (given by additional ciphertext blocks, or by the tag corresponding to the message), we can post-filter these states and recover the correct internal state X_0 (which allows us to determine all X_i for $i \geq 0$) with a time complexity of 2^{15c} computations. This is less than the time complexity of exhaustive search by a factor of 2^c.

The main procedure of the attack is given in Algorithm 1, where the nibble sets $\mathcal{N}_1$, $\mathcal{N}_1'$, $\mathcal{N}_2$ and $\mathcal{N}_2'$ are defined in Sect. 5.

The first step (Step 1 – lines 2,3) consists of an initial guess-and-determine phase. The following two steps (Step 2a – line 4 and Step 2b – line 5) construct the look-up tables T_1 and T_2, respectively. These two steps are independent of each other, however, both of them depend on Step 1. In the final steps of the

[4] Recall that after the initialization, the encryption process does not depend on K, and thus X_0 fully determines the result of the encryption.

Algorithm 1. Main Procedure of the State-Recovery Attack.

1:	**function** StateRecovery	
2:	**Guess** nibbles of $\mathcal{N}_1$	# *Step 1* – $\lvert\mathcal{N}_1\rvert$ =*12 nibbles*
3:	Determine values for nibbles of $\mathcal{N}_1'$	# *Step 1*
4:	Construct table T_1	# *Step 2a* – 2^{3c} *operations*
5:	Construct table T_2	# *Step 2b* – 2^{3c} *operations*
6:	**Guess** nibbles of $\mathcal{N}_2$	# *Step 3a* – $\lvert\mathcal{N}_2\rvert$ =*3 nibbles*
7:	Determine values for nibbles of $\mathcal{N}_2'$	# *Step 3a*
8:	Use table T_1 to determine additional nibbles	# *Step 3b*
9:	Use table T_2 to determine internal state	# *Step 3c*
10:	**if** all output nibbles are consistent **then**	# $p = 2^{-c}$
11:	**return** State	# $2^{12c+3c-c} = 2^{14c}$ *states*

attack, we perform an additional guess-and-determine phase (Step 3a – lines 6,7), use the look-up table T_1 in order to determine the values of additional nibbles (Step 3b – line 8), and use the look-up table T_2 in order to determine the full state (Step 3c – line 9). Finally, we post-filter the remaining states (line 10) to return the 2^{14c} valid states. We note that these states are returned and post-filtered "on-the-fly", and thus the memory complexity of the attack is only 2^{3c} (which is the size of the look-up tables T_1 and T_2).

4.2 The Structure of the Steps in Our Attack

In general, all the steps of the attack are comprised of guessing/enumerating the values of several nibbles of the internal states $X_0, \ldots, X_8$, and then propagating the knowledge forwards and backwards through the states. The knowledge propagation uses a small number of simple equalities $\mathbb{E}$ (formally defined in Sect. 4) that are derived from the internal mappings of `FIDES`.

As $X_0, \ldots, X_8$ contain hundreds of nibbles, our attack uses hundreds of computations on the nibbles in order to propagate the knowledge through the states. As a result, manual verification of the attack is rather tedious. On the other hand, it is important to stress that *automatic verification* of the attack is rather simple, as one needs to program the main procedure of Algorithm 1 with the nibbles that are guessed/enumerated in each step. The program greedily propagates the knowledge through the states using $\mathbb{E}$, until X_0 is recovered.

Despite the simplicity of the automatic verification, we still aim to give the reader a good intuition of how the knowledge is propagated throughout the attack without listing all of its calculations in the text (which would make the paper very difficult to read). Thus, we provide in the next section figures that visualize the determined nibbles of the state after each step. In the extended version [8], we additionally provide in tables that describe some of the low-level calculations.

The Look-up Tables T_1 and T_2. We conclude this section with a remark regarding the look-up tables T_1 and T_2: as each one of these look-up tables is constructed using simple equalities, it may raise the concern that they do not contribute to the attack. Namely, it may seem possible to simply guess the 15 nibbles of $\mathcal{N}_1$ and $\mathcal{N}_2$ in the outer loop of the attack and recover the internal state by propagating the knowledge using simple equations. However, the data that the look-up tables store is inverted and indexed in a way which cannot be described using simple equations. In fact, the look-up tables are used in meet-in-the-middle algorithms (Steps 3b and 3c) in order to propagate the information in a more complex way.

5 Details of the State-Recovery Attack

In this section, we describe in detail all the steps of the attack. For each step, we use a figure that visualizes the nibbles of the state that we guess or enumerate, and the nibbles that we determine using $\mathbb{E}$. For the sake of completeness, we additionally provide in the extended version [8] of this paper, tables that describe how we use the equalities of $\mathbb{E}$ in each step.

We partition $\mathbb{E}$ into two groups, $\mathbb{E}_1$ and $\mathbb{E}_2$, where $\mathbb{E} = \mathbb{E}_1 \bigcup \mathbb{E}_2$. The first group $\mathbb{E}_1$ contains equalities that are directly derived from the FIDES internal mappings AddConstant, SubBytes, ShiftRows (applied independently to each nibble) and MixColumns (applied independently to each column), in addition to their inverses. The equalities of $\mathbb{E}_1$ can only be directly applied to a single nibble or a column of the state. The second group $\mathbb{E}_2$ contains the linear equalities of Sect. 3.1. These equalities are somewhat less "trivial" as they can be used in several ways in order to factor out an unknown variable (or a linear combination of variables), and express it as a linear combination of variables from one column of a state or two columns of two states, linked by MixColumns.

5.1 Step 1: Initial Guess-and-Determine

We start by guessing the values of the following 12 nibbles that define the set $\mathcal{N}_1$ (see hatched nibbles in Fig. 4):

$$\mathcal{N}_1 \stackrel{\text{def}}{=} \left\{ \begin{array}{l} X_3[0,0], X_3[0,1], X_3[0,2], X_3[3,1], \\ X_4[1,0], X_4[1,1], X_4[1,2], \\ X_5[0,0], X_5[0,1], X_5[0,2], \\ X_6[0,0], X_6[3,1] \end{array} \right\}.$$

Then, we propagate their values throughout the state. All the nibbles determined at the end of this step define the nibble-set $\mathcal{N}_1'$, and are given in Fig. 4.

We note that Step 1 depends on the known leaked nibbles $X_i[3,0]$ and $X_i[3,2]$ for $1 \leq i \leq 7$. However, this step is independent of the values of $X_0[3,0]$, $X_0[3,2]$, $X_8[3,0]$ and $X_8[3,2]$.

In total, given the 18 leaked nibbles, we expect about $2^{(32-18)c} = 2^{14c}$ conforming internal states, and thus after we guess the values of 12 nibbles, we expect to reduce the number of solutions to $2^{(14-12)c} = 2^{2c}$. In the sequel, we describe how to enumerate these 2^{2c} solutions in 2^{3c} computations and 2^{3c} memory.

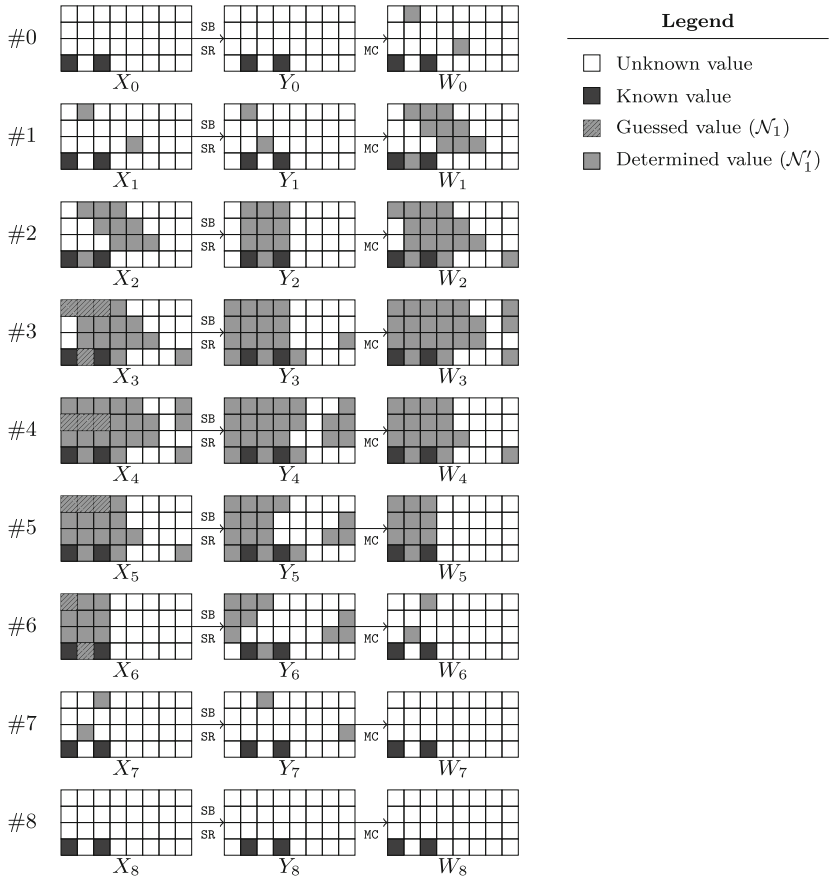


Fig. 4. Step 1: initial guess-and-determine.

5.2 Step 2a: Construction of T_1

In this step, we use the values of the nibbles of $\mathcal{N}_1 \cup \mathcal{N}_1'$ to construct the look-up table T_1, which contains 2^{3c} entries. During its construction, we enumerate all the possible values of the 3 nibbles $X_1[2,1]$, $X_2[2,0]$ and $X_2[1,7]$, and for each such value, we calculate and store the values described in Fig. 5. As an index to the table, we choose the following triplet of independent linear relations of the computed nibbles

$$\Big(W_1[1,7] \oplus Y_1[2,7],\ \ Y_1[1,0] \oplus W_1[2,0],\ \ Y_2[1,6] \oplus Y_2[2,6]\Big).$$

We note that unlike Step 1, this step depends on the value of the known nibble $X_0[3,0]$.

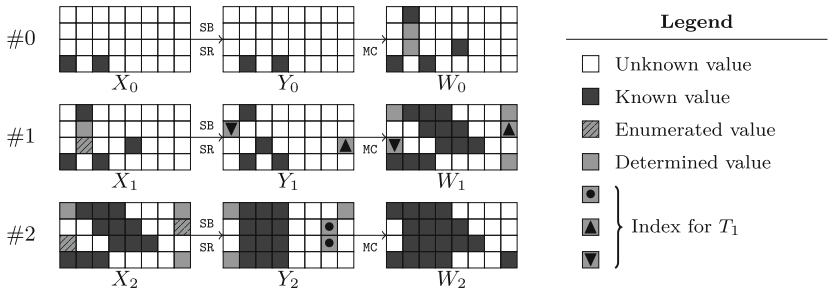


Fig. 5. Step 2a: construction of T_1.

5.3 Step 2b: Construction of T_2

In this step, we use the values of the nibbles of $\mathcal{N}_1 \cup \mathcal{N}_1'$ to construct the second look-up table T_2, which (similarly to T_1) contains 2^{3c} entries. During its construction, we enumerate all the possible values of the 3 nibbles $X_4[1,6]$, $X_5[1,4]$ and $X_6[2,4]$, and for each such value, we calculate and store the values described in Fig. 6. As an index to the table, we choose the following triplet of nibbles/linear relations of the computed nibbles

$$\Big(W_3[1,6] \oplus W_3[2,6],\ \ Y_4[2,5],\ \ Y_7[1,0] \oplus Y_7[2,0]\Big).$$

We note that this step depends on the value of $X_8[3,0]$ (unlike Step 1 and Step 2a).

5.4 Step 3a: Final Guess-and-Determine

In this step, we guess 3 additional nibbles to the 12 initial ones (see hatched nibbles on Fig. 7):

$$\mathcal{N}_2 \stackrel{\text{def}}{=} \Big\{X_1[0,3],\ X_1[1,3],\ X_3[2,7]\Big\}.$$

Their values allow to determine all the values marked in gray on Fig. 7, which define the set $\mathcal{N}_2'$. We note that unlike the previous steps, this step depends on the value of the leaked nibble $X_0[3,2]$.

5.5 Step 3b: Table T_1 Look-Up

In this step, we perform a look-up in table T_1 in order to determine the values of additional nibbles, and then propagate the knowledge further through the internal state. We access T_1 using the determined values of $W_1[0,7] \oplus W_1[3,7]$, $W_1[1,0] \oplus Y_1[2,0]$ and $W_2[1,6] \oplus W_2[2,6]$ (see nibbles ▲, ▼ and • in Fig. 8). Indeed, using the properties of the matrix $\mathbf{M}$:

$$\begin{aligned} W_1[0,7] \oplus W_1[3,7] &= Y_1[2,7] \oplus W_1[1,7] \\ W_1[1,0] \oplus Y_1[2,0] &= Y_1[1,0] \oplus W_1[2,0] \\ W_2[1,6] \oplus W_2[2,6] &= Y_2[1,6] \oplus Y_2[2,6], \end{aligned}$$

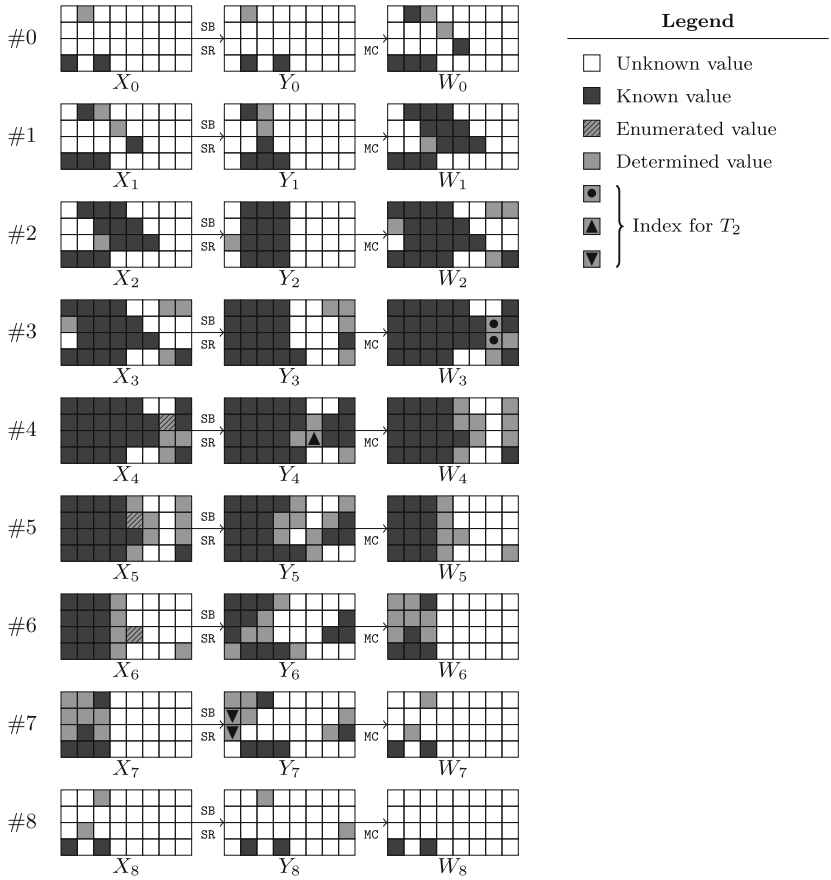


Fig. 6. Step 2b: construction of T_2.

where the right-hand sides define the elements of the index triplet to the table T_1. As T_1 contains 2^{3c} entries, we expect one match on average for each table look-up, which immediately determines all the additional hatched values in Fig. 8.

After the table T_1 look-up, we propagate the additional knowledge through the internal states.

5.6 Step 3c: Table T_2 Look-Up and State Recovery

In this step, we perform a look-up in table T_2 in order to determine the values of additional nibbles, and then propagate the knowledge further through the internal states in order to fully recover them. We access T_2 using the three determined values $Y_3[1,6] \oplus Y_3[2,6]$, $Y_4[2,5]$ and $Y_7[0,0] \oplus W_7[3,0]$. The nibble $Y_4[2,5]$ is an element of the index triplet to the table T_2, and for the two other elements (using the properties of the matrix $\mathbf{M}$), we have:

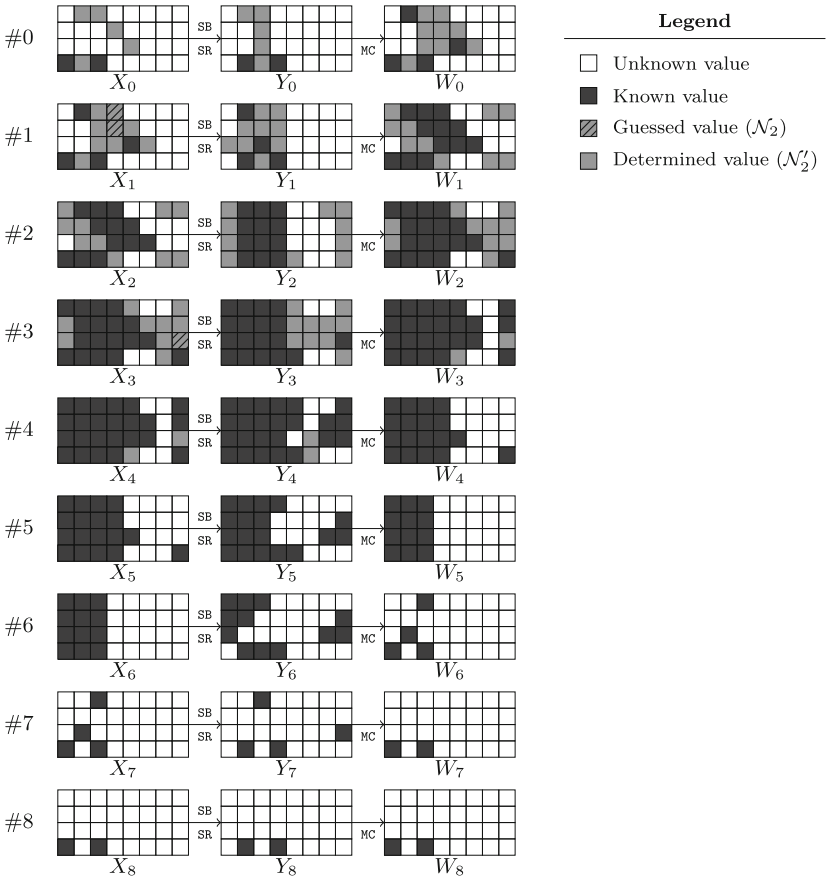


Fig. 7. Step 3a: three more nibbles are guessed to determine more values.

$$Y_3[1,6] \oplus Y_3[2,6] = W_3[1,6] \oplus W_3[2,6]$$
$$Y_7[0,0] \oplus W_7[3,0] = Y_7[1,0] \oplus Y_7[2,0],$$

where the right-hand sides for the two equations define the two remaining elements of the index triplet to the look-up table T_2. As T_2 contains 2^{3c} entries, we expect one match on average for each table look-up, which immediately determines all the additional values marked on Fig. 9.

After the table T_2 look-up, we propagate the additional knowledge through the internal states, which allows us to recover them fully (for the nibbles determine using the properties of the matrix $\mathbf{M}$. Namely, we fully recover X_4 by the following operations:

$$
\begin{aligned}
W_3[0,6] &= Y_3[1,6] \oplus W_3[2,6] \oplus W_3[3,6] \\
X_4[0,6] &= W_3[0,6] \oplus RC_3[0,6] \\
Y_4[0,5] &= W_4[1,5] \oplus Y_4[2,5] \oplus Y_4[3,5] \\
X_4[0,5] &= S^{-1}(Y_4[0,5]) \\
W_3[3,5] &= W_3[0,5] \oplus W_3[1,5] \oplus Y_3[2,5] \\
X_4[3,5] &= W_3[3,5] \oplus RC_3[3,5].
\end{aligned}
$$

Given X_4, we can compute all the states forwards and backwards.

Post-Filtering. Once the internal state is fully determined, we verify that the additional output $X_8[3,2]$ matches its leaked value. Indeed, the 18 leaked nibbles have *all* been used in the attack, with the exception of the very last one, $X_8[3,2]$. As this match occurs with probability 2^{-c}, the algorithm indeed enumerates the

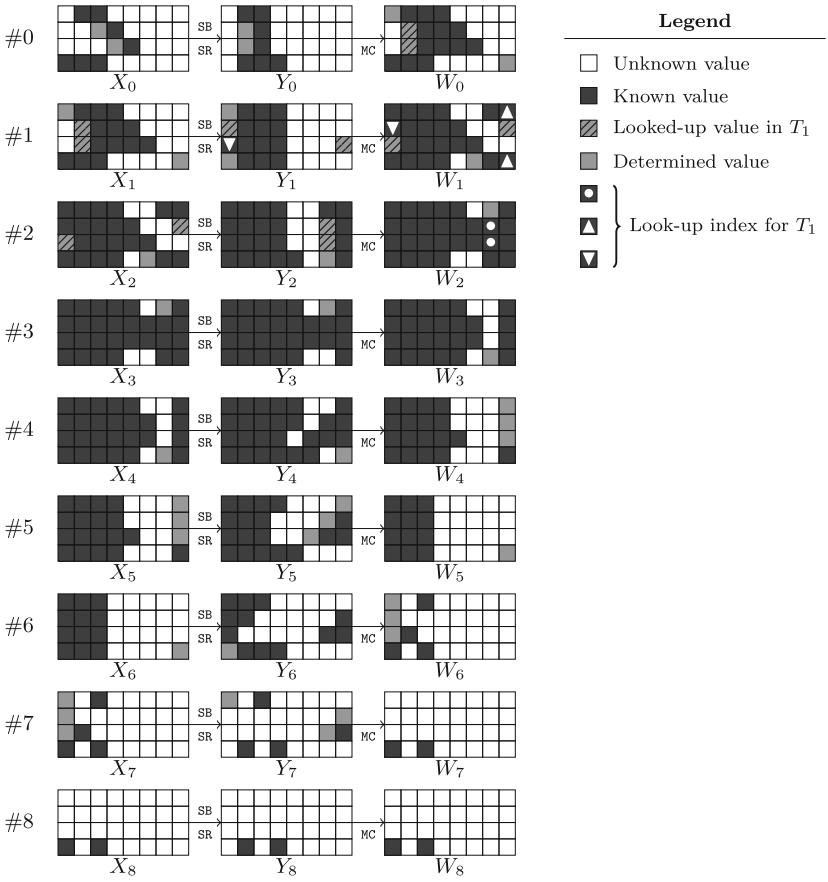


Fig. 8. Step 3b: table T_1 look-up.

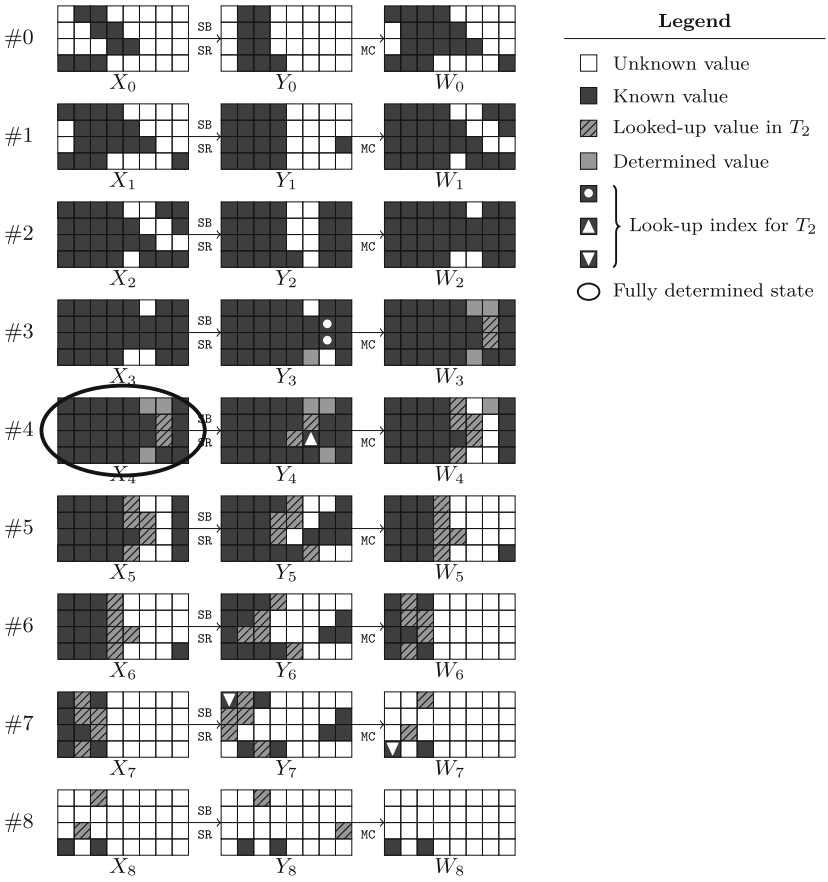


Fig. 9. Step 3c: table T_2 look-up and state recovery

2^{14c} internal states that produce the 18 leaked nibbles in about $2^{(12+3)c} = 2^{15c}$ computations, using a memory of about 2^{3c} elements. Finally, we can post-filter the solutions further using additional output (given by additional ciphertext blocks, or by the tag corresponding to the message).

6 Conclusions and Open Problems

In this paper, we presented state-recovery attacks on both versions of FIDES, and showed how to use them in order to forge messages. Our attacks use a guess-and-determine algorithm in order to break the security of the primitive given very little data and a small amount of memory.

A simple way to repair FIDES such that it would resist our attacks, is to use a linear transformation with a branch number of 5. However, this would have a negative impact on the efficiency of the implementation, and moreover, it is unclear whether such a change would guarantee resistance against different

(perhaps more complex) guess-and-determine attacks. In general, although the leak-extraction notion allows building cryptosystems with very efficient implementations, designing such systems which also offer a large security margin remains a challenging task for the future. In particular, it would be very interesting to design such cryptosystems which provably resist guess-and-determine attacks, such as the ones presented in this paper.

References

1. Bertoni, G., Daemen, J., Peeters, M., Van Assche, G.: Duplexing the sponge: single-pass authenticated encryption and other applications. In: Miri, A., Vaudenay, S. (eds.) SAC 2011. LNCS, vol. 7118, pp. 320–337. Springer, Heidelberg (2012)
2. Bilgin, B., Bogdanov, A., Knežević, M., Mendel, F., Wang, Q.: FIDES: lightweight authenticated cipher with side-channel resistance for constrained hardware. In: Bertoni, G., Coron, J.-S. (eds.) CHES 2013. LNCS, vol. 8086, pp. 142–158. Springer, Heidelberg (2013)
3. Biryukov, A.: The design of a stream cipher LEX. In: Biham, E., Youssef, A.M. (eds.) SAC 2006. LNCS, vol. 4356, pp. 67–75. Springer, Heidelberg (2007)
4. Bogdanov, A., Mendel, F., Regazzoni, F., Rijmen, V., Tischhauser, E.: ALE: AES-based lightweight authenticated encryption. In: Moriai, S. (ed.) FSE 2013. LNCS, vol. 8424, pp. 447–466. Springer, Heidelberg (2013)
5. Bouillaguet, C., Derbez, P., Fouque, P.-A.: Automatic search of attacks on round-reduced AES and applications. In: Rogaway, P. (ed.) CRYPTO 2011. LNCS, vol. 6841, pp. 169–187. Springer, Heidelberg (2011)
6. CAESAR: Competition for Authenticated Encryption: Security, Applicability, and Robustness. http://competitions.cr.yp.to/caesar.html
7. Daemen, J., Rijmen, V.: The Design of Rijndael: AES - The Advanced Encryption Standard. Springer, Heidelberg (2002)
8. Dinur, I., Jean, J.: Cryptanalysis of FIDES. Cryptology ePrint Archive, Report 2014/058 (2014)
9. Dunkelman, O., Keller, N.: A new attack on the LEX stream cipher. In: Pieprzyk, J. (ed.) ASIACRYPT 2008. LNCS, vol. 5350, pp. 539–556. Springer, Heidelberg (2008)
10. Khovratovich, D., Rechberger, C.: The LOCAL attack: cryptanalysis of the authenticated encryption scheme ALE. In: Lange, T., Lauter, K., Lisoněk, P. (eds.) SAC 2013. LNCS, vol. 8282, pp. 174–184. Springer, Heidelberg (2013)
11. Wu, S., Wu, H., Huang, T., Wang, M., Wu, W.: Leaked-state-forgery attack against the authenticated encryption algorithm ALE. In: Sako, K., Sarkar, P. (eds.) ASIACRYPT 2013, Part I. LNCS, vol. 8269, pp. 377–404. Springer, Heidelberg (2013)

Foundations and Theory

Security Analysis of Key-Alternating Feistel Ciphers

Rodolphe Lampe[1(✉)] and Yannick Seurin[2]

[1] University of Versailles, Versailles, France
rodolphe.lampe@gmail.com
[2] ANSSI, Paris, France
yannick.seurin@m4x.org

Abstract. We study the security of *key-alternating Feistel* ciphers, a class of key-alternating ciphers with a Feistel structure. Alternatively, this may be viewed as the study of Feistel ciphers where the pseudo-random round functions are of the form $F_i(x \oplus k_i)$, where k_i is the (secret) round key and F_i is a *public* random function that the adversary is allowed to query in a black-box way. Interestingly, our results can be seen as a generalization of traditional results *à la* Luby-Rackoff in the sense that we can derive results for this model by simply letting the number of queries of the adversary to the public random functions F_i be zero in our general bounds. We make an extensive use of the coupling technique. In particular (and as a result of independent interest), we improve the analysis of the coupling probability for balanced Feistel schemes previously carried out by Hoang and Rogaway (CRYPTO 2010).

Keywords: Block cipher · Key-alternating cipher · Feistel cipher · Coupling · Provable security

1 Introduction

Block Ciphers. Block cipher designs roughly fall in two main classes, namely Feistel networks and substitution-permutation networks (SPNs). The primary security notion when studying a block cipher is pseudorandomness: it should be impossible except with negligible probability for any adversary with reasonable resources which has black-box access to a permutation oracle (and potentially its inverse) to distinguish whether it is interacting with the block cipher with a uniformly random key, or with a truly random permutation. Since proving upper bounds on the distinguishing advantage of a general adversary for a concrete block cipher seems out of reach of current techniques, research has focused on proving results by idealizing some components of the block cipher.

R. Lampe—This author is partially supported by the French Direction Générale de l'Armement.

Y. Seurin—This author is partially supported by the French National Agency of Research: ANR-11-INS-011.

C. Cid and C. Rechberger (Eds.): FSE 2014, LNCS 8540, pp. 243–264, 2015.
DOI: 10.1007/978-3-662-46706-0_13

For Feistel networks, most of the provable security work falls in what is usually named the Luby-Rackoff framework, in reference to the seminal work of Luby and Rackoff [11]. In this setting, the round functions of the Feistel scheme are idealized as being uniformly random (and secret). Such results can be directly transposed to the case where the round functions are pseudorandom via a composition theorem (but again proving any lower bound for the pseudorandomness of some concrete function family is out of reach of current techniques). Starting from the Luby-Rackoff result that the 3-round Feistel scheme is a pseudorandom permutation [11], and the proof by Patarin [17] that four rounds yield a strong pseudorandom permutation (where *strong* means that inverse queries to the permutation oracle are allowed), a long series of work established refined bounds for larger number of rounds [8,12,13,18,19,22].

For SPN ciphers, provable security results were for a long time limited to resistance to specific attacks such as differential and linear attacks [3]. Recently though, a number of results have been obtained for the ideal *key-alternating* cipher, *a.k.a.* iterated Even-Mansour cipher. An r-round key-alternating cipher is specified by r public permutations on n bits $P_0, \ldots, P_{r-1}$, and encrypts a plaintext x as

$$y = k_r \oplus P_{r-1}(k_{r-1} \oplus P_{r-2}(\cdots P_0(k_0 \oplus x) \cdots)),$$

where $(k_0, \ldots, k_r)$ are $r+1$ keys of n bits. When $r = 1$, this construction was analyzed and its security established up to $\mathcal{O}(2^{n/2})$ queries by Even and Mansour [6] in the random permutation model for P_0, *i.e.* when the permutation P_0 is a random permutation oracle to which the adversary can make direct and inverse queries. Subsequently, a number of papers improved this seminal result to larger numbers of rounds [1,9,21], culminating with the proof by Chen and Steinberger [2] that the r-round ideal key-alternating cipher is secure up to $\mathcal{O}(2^{\frac{rn}{r+1}})$ adaptive, chosen plaintext and ciphertext queries (which is optimal since it matches the best known attack).

Our Contribution. In this work, we study the security of Feistel networks in a setting where the round functions are random and *public* (meaning that the adversary can make oracle queries to these functions), and an independent round key is xored before each round function. In other words, the state at round i is updated according to $(x_L, x_R) \mapsto (x_R, x_L \oplus F_i(x_R \oplus k_i))$, where x_L and x_R are respectively the left and right n-bit halves of the state, and k_i is an n-bit round key. In a sense, this can be seen as transposing the setting of recent works on the ideal key-alternating cipher (which uses the random permutation model) to Feistel ciphers (in the random function model). For this reason, we call such a design a *key-alternating Feistel cipher* (KAF cipher for short). In fact, one can easily see that two rounds of a key-alternating Feistel cipher can be rewritten as a (single-key) one-round Even-Mansour cipher, where the permutation P is a two-round (public and un-keyed) Feistel scheme (see Fig. 2). When we want to insist that we consider the model where the round functions F_i are uniformly random public functions, we talk of the *ideal* KAF cipher. Hence, the setting we consider departs from the usual Luby-Rackoff framework in two ways: on one

hand, we consider "complex" round functions (random function oracles), but on the other hand we consider the simplest keying procedure, namely xoring.

In this setting, the resources of the adversary are measured by the maximal number q_e of queries to the permutation oracle (and its inverse for strong pseudorandomness), and the maximal number q_f of queries to each round function. In the special case where $q_f = 0$ (*i.e.* the adversary has not access to the random round functions), one exactly recovers the more usual Luby-Rackoff setting, so that our analysis allows to directly derive results for this framework as well by letting q_f be zero.

Our analysis is based on a coupling argument, a well-known tool from the theory of Markov chains. Its use in cryptography has been pioneered by Mironov [15] for the analysis of the shuffle of the RC4 stream cipher, and later by Morris *et al.* for the analysis of maximally unbalanced Feistel schemes [16]. Later use of this technique includes [8,9]. The work of Hoang and Rogaway [8] is particularly relevant to this paper since they analyzed (among other variants) balanced Feistel schemes, although only in the traditional Luby-Rackoff setting.

Our bounds show that an ideal KAF cipher with r rounds ensures security up to $\mathcal{O}(2^{\frac{tn}{t+1}})$ queries of the adversary, where

- $t = \lfloor \frac{r}{3} \rfloor$ for non-adaptive chosen-plaintext (NCPA) adversaries;
- $t = \lfloor \frac{r}{6} \rfloor$ for adaptive chosen-plaintext and ciphertext (CCA) adversaries.

In the Luby-Rackoff setting ($q_f = 0$), we improve on the previous work of Hoang and Rogaway [8] thanks to a more careful analysis of the coupling argument. Namely we show that the ideal LR cipher is CCA-secure up to $\mathcal{O}(2^{\frac{tn}{t+1}})$ queries, where $t = \lfloor \frac{r-1}{4} \rfloor$. The best proven security bound in the Luby-Rackoff setting remains due to Patarin [19], who showed that the 6-round Feistel cipher is secure up to $\mathcal{O}(2^n)$ queries against CCA distinguishers. However his analysis is much more complicated and does not seem to be directly transposable to the case of KAF ciphers. We feel that the simplicity of the coupling argument is an attractive feature in addition to being immediately applicable to KAF ciphers.

Other Related Work. We are only aware of two previous works in a setting similar to ours. The first is a paper by Ramzan and Reyzin [20], who showed that the 4-round Feistel construction remains (strongly) pseudorandom when the adversary is given oracle access to the two middle round functions. This setting is somehow intermediate between the Luby-Rackoff and the KAF setting. The second paper is by Gentry and Ramzan [7], who showed that the public random permutation of the Even-Mansour cipher $x \mapsto k_1 \oplus P(k_0 \oplus x)$ can be replaced by a 4-round public Feistel scheme, and the resulting construction is still a strong pseudorandom permutation. While their result shows how to construct a strong pseudorandom permutation from only four public random functions (while we need six rounds of Feistel and hence six random functions to get the same result in this paper), their analysis only yields a $\mathcal{O}(2^{n/2})$ security bound. On the contrary, our bounds improve asymptotically with the number of rounds, approaching the information-theoretic bound of $\mathcal{O}(2^n)$ queries. In fact, our results are the first ones beyond the birthday bound for KAF ciphers.

Organization. We start with some definitions and preliminaries in Sect. 2. In Sect. 3, we prove a probabilistic lemma which will be useful later to study the coupling probability for Feistel schemes. This result might be of independent interest. Finally, Sect. 4 contains our main results about the security of ideal KAF ciphers and Luby-Rackoff ciphers. Some proofs have been omitted for reasons of space and can be found in the full version of the paper [10].

2 Preliminaries

2.1 General Notation

In all the following, we fix an integer $n \geq 1$. Given an integer $q \geq 1$ and a set S, we denote $(S)^{*q}$ the set of all q-tuples of pairwise distinct elements of S. We denote $[i;j]$ the set of integers k such that $i \leq k \leq j$.

The set of functions of n bits to n bits will be denoted $\mathcal{F}_n$. Let $\boldsymbol{F} = (F_0, \ldots, F_{r-1}) \in (\mathcal{F}_n)^r$ be a tuple of functions, and $u = (u_0, \ldots, u_{r-1})$ and $v = (v_0, \ldots, v_{r-1})$ where for $i = 0, \ldots, r-1$, $u_i = (u_i^1, \ldots, u_i^q) \in (\{0,1\}^n)^q$ and $v_i = (v_i^1, \ldots, v_i^q) \in (\{0,1\}^n)^q$ are q-tuples of n-bit strings. We write $F_i(u_i) = v_i$ as a shorthand to mean that $F_i(u_i^j) = v_i^j$ for all $j = 1, \ldots, q$, and $\boldsymbol{F}(u) = v$ as a shorthand to mean that $F_i(u_i) = v_i$ for all $i = 0, \ldots, r-1$.

2.2 Definitions

Given a function F from $\{0,1\}^n$ to $\{0,1\}^n$ and a n-bit key k, the one-round keyed Feistel permutation is the permutation on $\{0,1\}^{2n}$ defined as:

$$\Psi_k^F(x_L, x_R) = (x_R, x_L \oplus F(x_R \oplus k)),$$

where x_L and x_R are respectively the left and right n-bit halves of the input.

A key-alternating Feistel cipher (KAF cipher for short) with r rounds is specified by r public round functions $F_0, \ldots, F_{r-1}$ from $\{0,1\}^n$ to $\{0,1\}^n$, and will be denoted $\mathtt{KAF}^{F_0,\ldots F_{r-1}}$. It has key-space $(\{0,1\}^n)^r$ and message space $\{0,1\}^{2n}$. It maps a key $(k_0, \ldots, k_{r-1})$ and a plaintext x to the ciphertext defined as:

$$\mathtt{KAF}^{F_0,\ldots F_{r-1}}((k_0, \ldots, k_{r-1}), x) = \Psi_{k_{r-1}}^{F_{r-1}} \circ \cdots \circ \Psi_{k_0}^{F_0}(x).$$

We will denote $\mathtt{KAF}_{k_0,\ldots,k_{r-1}}^{F_0,\ldots F_{r-1}}$ the permutation on $\{0,1\}^{2n}$ mapping a plaintext x to $\mathtt{KAF}^{F_1,\ldots F_r}((k_0, \ldots, k_{r-1}), x)$. When the number of rounds is clear, we simply denote $\boldsymbol{F} = (F_0, \ldots, F_{r-1})$ and $k = (k_0, \ldots, k_{r-1})$, and $\mathtt{KAF}_k^{\boldsymbol{F}}$ the $2n$-bit permutation specified by round functions $\boldsymbol{F}$ and round keys k.

As already noted in [4], a KAF cipher with an even number of rounds can be seen as a special case of a (permutation-based) key-alternating cipher, also known as an iterated Even-Mansour cipher. Indeed, two rounds of a KAF cipher can be rewritten as (see Fig. 2):

$$\Psi_{k_{i+1}}^{F_{i+1}} \circ \Psi_{k_i}^{F_i}(x) = (k_{i+1} \| k_i) \oplus \Psi_0^{F_{i+1}} \circ \Psi_0^{F_i}((k_{i+1} \| k_i) \oplus x).$$

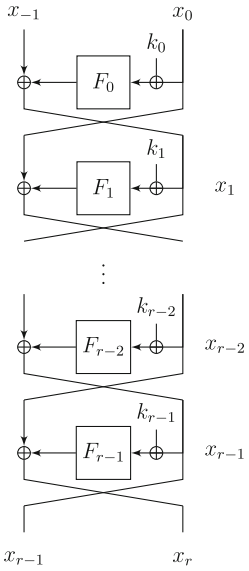


Fig. 1. Notations used for a r-round KAF cipher.

Here $\Psi_0^{F_{i+1}} \circ \Psi_0^{F_i}$ is the un-keyed two-round Feistel permutation with round functions F_i and F_{i+1}. Hence this permutation is public since the two round functions F_i and F_{i+1} are public oracles. Recall that the (single-key) Even-Mansour cipher on $2n$ bits is defined from a public permutation P on $2n$ bits as $E(k, x) = k \oplus P(k \oplus x)$, where k is the $2n$-bit key and x the $2n$-bit plaintext [5,6]. Hence, a $2r'$-round KAF cipher with round functions $(F_0, \ldots, F_{2r'-1})$ and round keys $(k_0, \ldots, k_{2r'-1})$ can be seen as an r'-round key-alternating cipher, where the i-th permutation, $i = 0, \ldots, r'-1$, is the (un-keyed) two-round Feistel scheme with round functions F_{2i} and F_{2i+1}, and the sequence of $2n$-bit keys is $(\tilde{k}_0, \tilde{k}_0 \oplus \tilde{k}_1, \ldots, \tilde{k}_{r'-2} \oplus \tilde{k}_{r'-1}, \tilde{k}_{r'-1})$ with $\tilde{k}_i = k_{2i+1} \| k_{2i}$. (This is more accurately described as the cascade of r' single-key one-round Even-Mansour ciphers.)

As already mentioned in introduction, the iterated Even-Mansour cipher has been subject to extensive security analysis recently (these works often consider the case where all keys are independent, but virtually all the results, in particular [2,9], apply to the cascade of single-key one-round Even-Mansour schemes). However, these results cannot be transposed to the case of KAF ciphers since they are a special sub-case of the general construction, and hence a dedicated analysis is required. In particular, note that even though the single-key one-round Even-Mansour cipher with a $2n$-bit permutation is provably secure up to $\mathcal{O}(2^n)$ queries against CCA distinguishers, the two-round ideal KAF cipher is easily distinguishable from a random permutation with only two chosen plaintext queries (namely: query the encryption oracle on (x_L, x_R) and (x'_L, x_R), and check whether the respective ciphertexts (y_L, y_R) and (y'_L, y'_R) satisfy $y_L \oplus y'_L = x_L \oplus x'_L$).

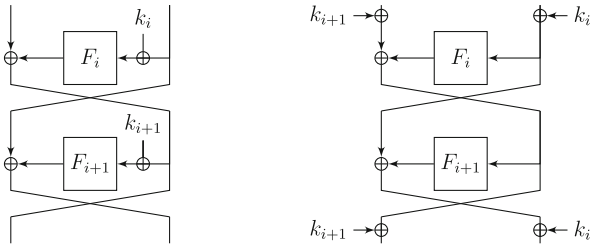


Fig. 2. An alternative view of two rounds of a KAF cipher.

2.3 Security Notions

In order to study the pseudorandomness of KAF ciphers, we will consider distinguishers $\mathcal{D}$ interacting with r function oracles $\boldsymbol{F} = (F_0, \ldots, F_{r-1})$ from n bits to n bits and a $2n$-bit permutation oracle (and potentially its inverse) which is either the KAF cipher $\mathtt{KAF}_k^{\boldsymbol{F}}$ specified by $\boldsymbol{F}$ with a uniformly random key $k = (k_0, \ldots, k_{r-1})$, or a perfectly random permutation P (independent from $\boldsymbol{F}$). A (q_e, q_f)-distinguisher is a distinguisher that makes at most q_e queries to the permutation oracle and at most q_f queries to each round function $F_0, \ldots, F_{r-1}$. We will consider only computationally unbounded distinguishers. As usual we restrict ourself *wlog* to deterministic distinguishers that never make redundant queries and always make the maximal number of allowed queries to each oracle.

As in [9], we will define two types of distinguishers, depending on the way it can make its queries to the oracles, namely non-adaptive chosen-plaintext (NCPA) distinguishers, and (adaptive) chosen-plaintext and ciphertext (CCA) distinguishers. We stress that the distinction adaptive/non-adaptive only refers to the queries to the permutation oracle. We now give the precise definitions of these two classes of distinguishers.

Definition 1. *A (q_e, q_f)-NCPA distinguisher runs in two phases:*

1. *in a first phase, it makes exactly q_f queries to each round function F_i. These queries can be adaptive.*
2. *in a second phase, it chooses a tuple of q_e non-adaptive forward queries $x = (x^1, \ldots, x^{q_e})$ to the permutation oracle, and receives the corresponding answers. By non-adaptive queries, we mean that all queries must be chosen before receiving any answer from the permutation oracle, however these queries may depend on the answers received in the previous phase from the round function oracles F_i.*

A (q_e, q_f)-CCA distinguisher is the most general one: it makes adaptively q_f queries to each round function F_i and q_e forward or backward queries to the permutation oracle, in any order (in particular it may interleave queries to the permutation oracle and to the round function oracles).

In all the following, the probability of an event E when $\mathcal{D}$ interacts with $(\boldsymbol{F}, P)$ where P is a random permutation independent from the uniformly random round

functions $\boldsymbol{F}$ will simply be denoted $\Pr^*[E]$, whereas the probability of an event E when $\mathcal{D}$ interacts with $(\boldsymbol{F}, \mathtt{KAF}_k^{\boldsymbol{F}})$, where the key $k = (k_0, \ldots, k_{r-1})$ is uniformly random, will simply be denoted $\Pr[E]$. With these notations, the advantage of a distinguisher $\mathcal{D}$ is defined as $|\Pr[\mathcal{D}(1^n) = 1] - \Pr^*[\mathcal{D}(1^n) = 1]|$ (we omit the oracles in this notation since they can be deduced from the notation $\Pr[\cdot]$ or $\Pr^*[\cdot]$). The maximum advantage of a (q_e, q_f)-ATK-distinguisher against the ideal r-round KAF cipher with n-bit round functions (where ATK is NCPA or CCA) will be denoted $\mathbf{Adv}^{\mathrm{atk}}_{\mathtt{KAF}[n,r]}(q_e, q_f)$.

When $q_f = 0$, *i.e.* in the setting where the distinguisher is not allowed to query the round functions, it is not hard to see that the round keys $k_0, \ldots, k_{r-1}$ do not add any security, so that they can all be taken equal to zero. Hence we are brought back to the usual security framework *à la* Luby-Rackoff, where the round functions are uniformly random and play the role of the secret key (in other words, the key space in this setting is $(\mathcal{F}_n)^r$, where $\mathcal{F}_n$ is the set of all functions from n bits to n bits). In that case, our definitions of an NCPA and a CCA distinguisher correspond to the usual definitions of pseudorandomness of a blockcipher in the standard model (*i.e.* when no additional oracles are involved). In order to emphasize that this setting is qualitatively different, we will denote $\mathbf{Adv}^{\mathrm{atk}}_{\mathtt{LR}[n,r]}(q_e)$ the advantage of a $(q_e, q_f = 0)$-ATK-distinguisher against the ideal r-round Luby-Rackoff cipher.

To sum up, we consider in a single framework two flavors of Feistel ciphers: Luby-Rackoff ciphers, where the round functions are random and secret, and key-alternating Feistel ciphers, where round functions are of the type $F_i(x \oplus k_i)$, where k_i is a secret round key and F_i a public random function oracle.

2.4 Statistical Distance and Coupling

Given a finite event space Ω and two probability distributions μ and ν defined on Ω, the *statistical distance* (or total variation distance) between μ and ν, denoted $\|\mu - \nu\|$ is defined as:

$$\|\mu - \nu\| = \frac{1}{2} \sum_{x \in \Omega} |\mu(x) - \nu(x)| .$$

A *coupling* of μ and ν is a distribution λ on $\Omega \times \Omega$ such that for all $x \in \Omega$, $\sum_{y \in \Omega} \lambda(x, y) = \mu(x)$ and for all $y \in \Omega$, $\sum_{x \in \Omega} \lambda(x, y) = \nu(y)$. In other words, λ is a joint distribution whose marginal distributions are resp. μ and ν. The fundamental result of the coupling technique is the following one. See *e.g.* [9] for a proof.

Lemma 1 (Coupling Lemma). *Let μ and ν be probability distributions on a finite event space Ω, let λ be a coupling of μ and ν, and let $(X, Y) \sim \lambda$ (i.e. (X, Y) is a random variable sampled according to distribution λ). Then $\|\mu - \nu\| \leq \Pr[X \neq Y]$.*

3 A Useful Probabilistic Lemma

Readers may skip this section at first reading and come back after Lemma 11. In all the following, we interchangeably use the notation A_iA_j to denote the intersection $A_i \cap A_j$ of two events, and more generally $A_{i_1}A_{i_2}\cdots A_{i_k}$ to denote $A_{i_1} \cap A_{i_2} \cap \cdots \cap A_{i_k}$.

In this section, we consider the following problem: for $r \geq 2$, let $A_1, \ldots, A_r$ be events defined over the same probability space Ω, satisfying the following "negative dependence" condition:

Definition 2. *Let $p \in]0,1[$. A sequence of events $A_1, \ldots, A_r$ is said to be p-negatively dependent if for any $i \in [1;r]$ and any subset $S \subseteq [1;i-1]$, one has:*

$$\Pr\left[A_i \middle| \bigcap_{j\in S} A_j\right] \leq p\,,$$

with the convention that an empty intersection is the certain event Ω (hence, in particular $\Pr[A_i] \leq p$ for $i \in [1;r]$).

We denote C_r the event $C_r = \cap_{i=1}^{r-1}(A_i \cup A_{i+1})$, or in a more eloquent form:

$$C_r = (A_1 \cup A_2)(A_2 \cup A_3)\cdots(A_{r-2} \cup A_{r-1})(A_{r-1} \cup A_r)\,.$$

Our goal is to find an upper bound on the probability $\Pr[C_r]$ of this event. Note that C_r is an event in conjunctive normal form, which is not directly amenable to deriving an adequate upper bound. However, once written in disjunctive normal form, one can easily upper bound its probability using the following simple fact:

Lemma 2. *Let $A_1, \ldots, A_r$ be p-negatively dependent events. Then for any $k \in [1;r]$ and any distinct integers $i_1, \ldots, i_k$ in $[1;r]$ one has:*

$$\Pr[A_{i_1}\cdots A_{i_k}] \leq p^k\,.$$

Proof. By induction on k. □

In the following, for a sequence $\alpha \in \{0,1\}^{r-1}$, we denote α_i the i-th bit of α. By developing straightforwardly event C_r, one obtains the following expression.

Lemma 3.

$$\bigcap_{i=1}^{r-1}(A_i \cup A_{i+1}) = \bigcup_{\alpha\in\{0,1\}^{r-1}} \bigcap_{i=1}^{r-1} A_{i+\alpha_i}\,.$$

Proof. By induction on r. □

For any sequence $\alpha \in \{0,1\}^{r-1}$, we will denote $B_{r,\alpha} = \cap_{i=1}^{r-1} A_{i+\alpha_i}$, so that $C_r = \cup_{\alpha\in\{0,1\}^{r-1}} B_{r,\alpha}$. Depending on α, $B_{r,\alpha}$ may be the intersection of strictly less than $r-1$ events (*e.g.* as soon as $\alpha_i = 1$ and $\alpha_{i+1} = 0$ for some i). Moreover, for two distinct sequences α and α', it may happen that $B_{r,\alpha} \subset B_{r,\alpha'}$. Consider

for example the simple case $r = 3$. Then $B_{3,00} = A_1 \cap A_2$ and $B_{3,10} = A_2 \cap A_2 = A_2$, so that $B_{3,00} \subset B_{3,10}$ (see Table 1 for the developed and "reduced" disjunctive form of C_r for r up to 8). This motivates the following definition of *irreducible* sequences, which informally characterize the "minimal" set of events $B_{r,\alpha}$ covering C_r.

Definition 3. *We define the set of* irreducible *sequences as the following regular language (λ denotes the empty string):*

$$\mathcal{I} = \{\lambda, 0\}\{10, 100\}^*\{\lambda, 1\}\,.$$

In other words, irreducible sequences are obtained by concatenating possibly a single 0, then the two patterns 10 and 100 arbitrarily, and finally possibly a single 1. Sequences in $\{0,1\}^* \setminus \mathcal{I}$ are called reducible. *We denote $\mathcal{I}_r$ the set of irreducible sequences of length r.*

It is easy to see that irreducible sequences are exactly sequences α such that 0α does not contain three consecutive zeros or two consecutive ones, but we will not need this characterization here.

The usefulness of irreducible sequences comes from the following lemma.

Lemma 4. $\Pr[C_r] \leq \sum_{\alpha \in \mathcal{I}_{r-1}} \Pr[B_{r,\alpha}]$.

Proof. We show by induction on r that $C_r \subseteq \cup_{\alpha \in \mathcal{I}_{r-1}} B_{r,\alpha}$, from which the lemma follows by the union bound. We first show it directly for $r = 2, 3, 4$. This trivially holds for $r = 2$ since $C_2 = A_1 \cup A_2 = B_{2,0} \cup B_{2,1}$ and the two sequences 0 and 1 are irreducible. For $r = 3$, we have:

$$C_3 = (A_1 \cup A_2)(A_2 \cup A_3) \subseteq A_1A_3 \cup A_2 = B_{3,01} \cup B_{3,10}\,,$$

from which the result follows since 01 and 10 are irreducible while 00 and 11 are reducible. For $r = 4$, we have

$$\begin{aligned} C_4 = (A_1 \cup A_2)(A_2 \cup A_3)(A_3 \cup A_4) &\subseteq A_1A_3 \cup A_2A_3 \cup A_2A_4 \\ &\subseteq B_{4,010} \cup B_{4,100} \cup B_{4,101}\,, \end{aligned}$$

from which the result follows since 010, 100, and 101 are the only irreducible sequences of length 3.

Let us now show the result for $r \geq 5$, assuming that the result holds for $r-1$. We have:

$$\begin{aligned} C_r = C_{r-1} \cap (A_{r-1} \cup A_r) &\subseteq \left(\cup_{\alpha \in \mathcal{I}_{r-2}} B_{r-1,\alpha}\right) \cap (A_{r-1} \cup A_r) \\ &\subseteq \left(\cup_{\alpha \in \mathcal{I}_{r-2}} B_{r,\alpha 0}\right) \cup \left(\cup_{\alpha \in \mathcal{I}_{r-2}} B_{r,\alpha 1}\right) \end{aligned}$$

Hence, it suffices to show that for any irreducible $\alpha \in \mathcal{I}_{r-2}$ such that $\alpha 0$, resp. $\alpha 1$, is reducible, there is an irreducible $\bar{\alpha} \in \mathcal{I}_{r-1}$ such that $B_{r,\alpha 0} \subseteq B_{r,\bar{\alpha}}$, resp. $B_{r,\alpha 1} \subseteq B_{r,\bar{\alpha}}$. We distinguish three cases depending on the form of $\alpha \in \mathcal{I}_{r-2}$. Note that since we assume $r - 2 \geq 3$, α contains at least a pattern 10 or 100, so that either $\alpha = \alpha'10$, or $\alpha = \alpha'100$, or $\alpha = \alpha'1$, with $\alpha' \in \{\lambda, 0\}\{10, 100\}^*$ in each case.

– Case 1: $\alpha = \alpha'10$; in that case, we see that both $\alpha 0 = \alpha'100$ and $\alpha 1 = \alpha'101$ are irreducible, so there is nothing to prove.
– Case 2: $\alpha = \alpha'100$; in that case, $\alpha 1 = \alpha'1001$ is irreducible, so there is nothing to prove for $\alpha 1$. On the other hand, $\alpha 0 = \alpha'1000$ is reducible. Let $\bar{\alpha} = \alpha'1010$. Note that $\bar{\alpha}$ is irreducible. Moreover:

$$B_{r,\alpha 0} = B_{r,\alpha'1000} = B_{r-4,\alpha'} \cap A_{r-3}A_{r-2}A_{r-1}$$
$$B_{r,\bar{\alpha}} = B_{r,\alpha'1010} = B_{r-4,\alpha'} \cap A_{r-3}A_{r-1}\,,$$

so that $B_{r,\alpha 0} \subseteq B_{r,\bar{\alpha}}$.
– Case 3: $\alpha = \alpha'1$; in that case, $\alpha 0 = \alpha'10$ is irreducible, so there is nothing to prove for $\alpha 0$. On the other hand, $\alpha 1 = \alpha'11$ is reducible. Let $\bar{\alpha} = \alpha'10$. Note that $\bar{\alpha}$ is irreducible. Moreover:

$$B_{r,\alpha 1} = B_{r,\alpha'11} = B_{r-2,\alpha'} \cap A_{r-1}A_r$$
$$B_{r,\bar{\alpha}} = B_{r,\alpha'10} = B_{r-2,\alpha'} \cap A_{r-1}\,,$$

so that $B_{r,\alpha 1} \subseteq B_{r,\bar{\alpha}}$.

Hence $C_r \subseteq \cup_{\alpha \in \mathcal{I}_{r-1}} B_{r,\alpha}$, which concludes the proof. □

We now give an upper bound for the probability of events $B_{r,\alpha}$ for irreducible sequences α. For this, we introduce the following definition.

Definition 4. *The weight of a sequence $\alpha \in \{0,1\}^*$, denoted $\mathbf{w}(\alpha)$, is the number of patterns* 10 *it contains (i.e. the number of integers i such that $\alpha_i = 1$ and $\alpha_{i+1} = 0$).*

Lemma 5. *Let $\alpha \in \{0,1\}^{r-1}$ be an irreducible sequence. Then:*

$$\Pr[B_{r,\alpha}] \leq p^{r-1-\mathbf{w}(\alpha)}\,.$$

Proof. Let $k = \mathbf{w}(\alpha)$. By definition, there are exactly k distinct integers $i_1 < \ldots < i_k$ such that for each $i \in \{i_1, \ldots, i_k\}$ we have $\alpha_i = 1$ and $\alpha_{i+1} = 0$, which implies $A_{i+\alpha_i}A_{i+1+\alpha_{i+1}} = A_{i+1} = A_{i+\alpha_i}$. Hence we see that:

$$B_{r,\alpha} \subseteq \bigcap_{\substack{i=1 \\ i \neq i_1+1,\ldots,i_k+1}}^{r-1} A_{i+\alpha_i}\,,$$

which implies the result by Lemma 2 since the event on the right hand side is the intersection of exactly $r-1-k$ distinct events A_j. □

It remains to count the number of irreducible sequences of a given weight.

Lemma 6. *The number of irreducible sequences of length r and weight k is $\binom{k+2}{r-2k}$. Moreover the minimal and maximal weights of an irreducible sequence are respectively $k_{\min} = \lceil \frac{r-2}{3} \rceil$ and $k_{\max} = \lfloor \frac{r}{2} \rfloor$.*

Proof. Let a and b denote respectively the number of patterns 10 and 100 in an irreducible sequence. Clearly the weight k of the sequence satisfies $k = a + b$. Moreover, depending on whether the sequence starts with a single 0 and ends with a single 1, we have the following relation between a and b and the length r of the sequence:

- for sequences of the form $\lambda\{10, 100\}^*\lambda$, one has $2a + 3b = r$
- for sequences of the form $0\{10, 100\}^*\lambda$ or $\lambda\{10, 100\}^*1$, one has $2a+3b = r-1$
- for sequences of the form $0\{10, 100\}^*1$, one has $2a + 3b = r - 2$

Denoting $r' = r$, $r-1$ or $r-2$ depending on the case, we always have $2a+3b = r'$, which combined with $a + b = k$ yields $b = r' - 2k$. For each case the number of possible sequences is $\binom{a+b}{b} = \binom{k}{r'-2k}$. Hence the total number of irreducible sequences of length r and weight k is:

$$\binom{k}{r-2k} + 2\binom{k}{r-1-2k} + \binom{k}{r-2-2k} = \binom{k+2}{r-2k}.$$

The minimal and maximal weights of an irreducible sequence directly follows from the condition $0 \leq r - 2k \leq k + 2$ for $\binom{k+2}{r-2k}$ to be non-zero. This concludes the proof. □

We are now ready to state and prove the main result of this section, namely the following upper bound for $\Pr[C_r]$.

Lemma 7. *Let $A_1, \ldots, A_r$ be p-negatively dependent events. Then:*

$$\Pr\left[\bigcap_{i=1}^{r-1}(A_i \cup A_{i+1})\right] \leq \sum_{k=\lfloor \frac{r}{2} \rfloor}^{\lfloor \frac{2r}{3} \rfloor} \binom{r+1-k}{2r-3k} p^k .$$

Proof. Combining Lemmas 4, 5, and 6 (note that we apply this last lemma to sequences of length $r - 1$), we have:

$$\Pr[C_r] \leq \sum_{k=\lceil \frac{r-3}{3} \rceil}^{\lfloor \frac{r-1}{2} \rfloor} \binom{k+2}{r-1-2k} p^{r-1-k} .$$

which after the change of variable $r - 1 - k \leftarrow k'$ yields the desired bound. □

We checked Lemma 7 by directly expanding and reducing the conjunctive normal form of C_r for small values of r (see Table 1 for the upper bound obtained for values of r up to 8).

4 Application to the Security of Key-Alternating Feistel Ciphers

4.1 Coupling for Non-adaptive Distinguishers

We will first bound the advantage against the r-round ideal KAF cipher $\texttt{KAF}[n, r]$ of any NCPA distinguisher making at most q_e queries to the cipher and q_f queries

Table 1. Disjunctive normal form of event C_r and upper bound on $\Pr[C_r]$ for r up to 8.

r	C_r (developed and reduced)	$\Pr[C_r]$ upper bound
2	$A_1 \cup A_2$	$2p$
3	$A_1A_3 \cup A_2$	$p + p^2$
4	$A_1A_3 \cup A_2A_3 \cup A_2A_4$	$3p^2$
5	$A_1A_3A_4 \cup A_1A_3A_5 \cup A_2A_3A_5 \cup A_2A_4$	$p^2 + 3p^3$
6	$A_1A_3A_4A_6 \cup A_1A_3A_5 \cup A_2A_3A_5 \cup A_2A_4A_5 \cup A_2A_4A_6$	$4p^3 + p^4$
7	$A_1A_3A_4A_6 \cup A_1A_3A_5A_6 \cup A_1A_3A_5A_7 \cup A_2A_3A_5A_6 \cup A_2A_3A_5A_7 \cup A_2A_4A_5A_7 \cup A_2A_4A_6$	$p^3 + 6p^4$
8	$A_1A_3A_4A_6A_7 \cup A_1A_3A_4A_6A_8 \cup A_1A_3A_5A_6A_8 \cup A_1A_3A_5A_7 \cup A_2A_3A_5A_6A_8 \cup A_2A_3A_5A_7 \cup A_2A_4A_5A_7 \cup A_2A_4A_6A_7 \cup A_2A_4A_6A_8$	$5p^4 + 4p^5$

to each round function. For this we will upper bound the statistical distance between the outputs of the KAF cipher, conditioned on partial information about round functions obtained through the oracle queries to $F_0, \ldots, F_{r-1}$, and the uniform distribution on $(\{0,1\}^{2n})^{*q_e}$.

For any tuples $u = (u_0, \ldots, u_{r-1})$ and $v = (v_0, \ldots, v_{r-1})$ with $u_i, v_i \in (\{0,1\}^n)^{q_f}$, and $x \in (\{0,1\}^{2n})^{*q_e}$, we denote $\mu_{x,u,v}$ the distribution of the q_e-tuple $y = \mathtt{KAF}_k^{\boldsymbol{F}}(x)$ when the key $k = (k_0, \ldots, k_{r-1})$ is uniformly random, and the round functions $\boldsymbol{F} = (F_0, \ldots, F_{r-1})$ are uniformly random among functions satisfying $\boldsymbol{F}(u) = v$. In the Luby-Rackoff setting ($q_f = 0$), we sometimes simply denote this distribution μ_x. We also denote μ^* the uniform distribution over $(\{0,1\}^{2n})^{*q_e}$. Then we have the following lemma. Its proof is standard and very similar to the proof of [9, Lemma 4], and therefore omitted.

Lemma 8. *Let q_e, q_f be positive integers. Assume that there exists α such that for any tuples $u = (u_0, \ldots, u_{r-1})$, $v = (v_0, \ldots, v_{r-1})$ with $u_i, v_i \in (\{0,1\}^n)^{q_f}$, and $x \in (\{0,1\}^{2n})^{*q_e}$, we have $\|\mu_{x,u,v} - \mu^*\| \leq \alpha$. Then $\mathbf{Adv}^{\mathrm{ncpa}}_{\mathtt{KAF}[n,r]}(q_e, q_f) \leq \alpha$.*

In the remainder of this section, we will establish an upper bound α on $\|\mu_{x,u,v} - \mu^*\|$ by using a coupling argument similar to the one of Hoang and Rogaway [8] (and an improved analysis of this coupling in the Luby-Rackoff setting). In all the following, we fix tuples $u = (u_0, \ldots, u_{r-1})$, $v = (v_0, \ldots, v_{r-1})$ with $u_i = (u_i^1, \ldots, u_i^{q_f}) \in (\{0,1\}^n)^{q_f}$ and $v_i = (v_i^1, \ldots, v_i^{q_f}) \in (\{0,1\}^n)^{q_f}$, and $x = (x^1, \ldots, x^{q_e}) \in (\{0,1\}^{2n})^{*q_e}$.

For $0 \leq \ell \leq q_e - 1$, we denote ν_ℓ the distribution of the $(\ell+1)$ outputs of the KAF cipher when it receives inputs $(x^1, \ldots, x^\ell, x^{\ell+1})$, and ν_ℓ^* the distribution of the $(\ell+1)$ outputs of the KAF cipher when it receives inputs $(x^1, \ldots, x^\ell, z^{\ell+1})$, where $z^{\ell+1}$ is uniformly distributed over $\{0,1\}^{2n} \setminus \{x^1, \ldots, x^\ell\}$ (in both cases the key $k = (k_0, \ldots, k_{r-1})$ is uniformly random, and the round functions $\boldsymbol{F} = (F_0, \ldots, F_{r-1})$ are uniformly random among functions satisfying $\boldsymbol{F}(u) = v$). Then we have the following lemma, whose proof is similar to the one of [16, Lemma 2] (this lemma is not specific to our setting, and applies to any block cipher).

Lemma 9. $\|\mu_{x,u,v} - \mu^*\| \leq \sum_{\ell=0}^{q_e-1} \|\nu_\ell - \nu_\ell^*\|$.

Proof. Deferred to the full version of the paper [10]. □

We now turn to upper bounding $\|\nu_\ell - \nu_\ell^*\|$ for $0 \leq \ell \leq q_e - 1$. Our goal is to describe a coupling of ν_ℓ and ν_ℓ^*, *i.e.* a joint distribution on pairs of $(\ell+1)$-tuples of $2n$-bit strings, whose marginal distributions are ν_ℓ and ν_ℓ^*. For this, we consider two KAF ciphers in parallel. The first one, $\mathtt{KAF}_k^{\boldsymbol{F}}$, takes as inputs $(x^1, \ldots, x^\ell, x^{\ell+1})$, while the second one, $\mathtt{KAF}_{k'}^{\boldsymbol{F}'}$, where $\boldsymbol{F}' = (F'_0, \ldots, F'_{r-1})$, takes as inputs $(x^1, \ldots, x^\ell, z^{\ell+1})$, where $z_{\ell+1}$ is any value in $\{0,1\}^{2n} \setminus \{x^1, \ldots, x^\ell\}$ (we upper bound the statistical distance between the outputs of the two systems for any $z_{\ell+1}$, from which it follows that the same upper bound holds when $z^{\ell+1}$ is uniformly random in $\{0,1\}^{2n} \setminus \{x^1, \ldots, x^\ell\}$). We assume that k is uniformly random and $\boldsymbol{F}$ is uniformly random among function tuples satisfying $\boldsymbol{F}(u) = v$, and we will define k' and $\boldsymbol{F}'$ so that they also satisfy these properties. This will ensure that the marginal distribution of the outputs of the first KAF cipher is ν_ℓ, and the marginal distribution of the outputs of the second KAF cipher is ν_ℓ^*.

The coupling. We now explain how the coupling of the two KAF ciphers is defined. First, the round keys in the second KAF cipher are the same as in the first one, namely $k' = k$. For $1 \leq j \leq \ell+1$, let x_{-1}^j and x_0^j denote respectively the left and right n-bit halves of x^j and for $1 \leq i \leq r$ let x_i^j be recursively defined as $x_i^j = x_{i-2}^j \oplus F_{i-1}(x_{i-1}^j \oplus k_{i-1})$ (see Fig. 1). For any $1 \leq j \leq \ell$ and any $0 \leq i \leq r-1$, we simply set $F'_i(x_i^j \oplus k_i) = F_i(x_i^j \oplus k_i)$ (note that this is consistent with the condition $\boldsymbol{F}'(u) = v$ in case some value $x_i^j \oplus k_i$ belongs to $u_i = (u_i^1, \ldots, u_i^{q_f})$, the set of queries of the distinguisher to the i-th round function). Since the ℓ first queries to the second KAF cipher are the same as the queries made to the first KAF cipher, this ensures that the ℓ first outputs of both ciphers are equal. It remains to explain how the $(\ell+1)$-th queries are coupled. Let $z_{-1}^{\ell+1}$ and $z_0^{\ell+1}$ be respectively the left and right n-bit halves of $z^{\ell+1}$. We will define recursively for $1 \leq i \leq r$ the round values $z_i^{\ell+1} = z_{i-2}^{\ell+1} \oplus F'_{i-1}(z_{i-1}^{\ell+1} \oplus k_{i-1})$. For this, we define two bad events which may happen at round $0 \leq i \leq r-1$ in each KAF cipher. We say that $\mathtt{XColl}_i$ happens if $x_i^{\ell+1} \oplus k_i$ is equal to $x_i^j \oplus k_i$ for some $1 \leq j \leq \ell$ (*i.e.* the input value to the i-th round function when enciphering $x^{\ell+1}$ collides with the input value to the i-th round function when enciphering some previous query x^j). We say that $\mathtt{FColl}_i$ happens if $x_i^{\ell+1} \oplus k_i \in u_i$ (*i.e.* the input value to the i-th round function when enciphering $x^{\ell+1}$ is equal to one of the oracle queries made to F_i by the distinguisher). We simply denote $\mathtt{Coll}_i = \mathtt{XColl}_i \cup \mathtt{FColl}_i$. Similarly, we say that $\mathtt{XColl}'_i$ happens if $z_i^{\ell+1} \oplus k_i$ is equal to $x_i^j \oplus k_i$ for some $1 \leq j \leq \ell$, that $\mathtt{FColl}'_i$ happens if $z_i^{\ell+1} \oplus k_i \in u_i$, and we denote $\mathtt{Coll}'_i = \mathtt{XColl}'_i \cup \mathtt{FColl}'_i$. Then, for $i = 0, \ldots, r-1$, we define $F'_i(z_i^{\ell+1} \oplus k_i)$ as follows:

(1) if $\mathtt{Coll}'_i$ happens, then $F'_i(z_i^{\ell+1} \oplus k_i)$ is already defined (either because $z_i^{\ell+1} \oplus k_i = x_i^j \oplus k_i$ for some $j \leq \ell$, or by the constraint $\boldsymbol{F}'(u) = v$);
(2) if $\mathtt{Coll}'_i$ does not happen but $\mathtt{Coll}_i$ happens, $F'_i(z_i^{\ell+1} \oplus k_i)$ is chosen uniformly at random;

(3) if neither $\texttt{Coll}_i$ nor $\texttt{Coll}'_i$ happens, then we define $F'_i(z_i^{\ell+1} \oplus k_i)$ so that $z_{i+1}^{\ell+1} = x_{i+1}^{\ell+1}$, namely:

$$F'_i(z_i^{\ell+1} \oplus k_i) = z_{i-1}^{\ell+1} \oplus x_{i-1}^{\ell+1} \oplus F_i(x_i^{\ell+1} \oplus k_i)\,.$$

One can check that the round functions $\boldsymbol{F}'$ in the second KAF cipher are uniformly random among functions tuples satisfying $\boldsymbol{F}'(u) = v$. This is clear when $F'_i(z_i^{\ell+1} \oplus k_i)$ is defined according to rule (1) or (2). When $F'_i(z_i^{\ell+1} \oplus k_i)$ is defined according to rule (3), then $F_i(x_i^{\ell+1} \oplus k_i)$ is uniformly random since $\texttt{Coll}_i$ does not happen, so that $F'_i(z_i^{\ell+1} \oplus k_i)$ is uniformly random as well. This implies that the outputs of the second KAF cipher are distributed according to ν_ℓ^* as wanted.

We say that the coupling is successful if all the outputs of both KAF ciphers are equal. Since the ℓ first outputs are aways equal by definition of the coupling, this is simply equivalent to having $z_{r-1}^{\ell+1} = x_{r-1}^{\ell+1}$ and $z_r^{\ell+1} = x_r^{\ell+1}$.

The following lemma simply states the key idea of a coupling argument: if the states just after round i when enciphering $x^{\ell+1}$ in the first cipher and $z^{\ell+1}$ in the second cipher, namely $(x_i^{\ell+1}, x_{i+1}^{\ell+1})$ and $(z_i^{\ell+1}, z_{i+1}^{\ell+1})$, are equal, then they remain equal after any subsequent round so that the coupling is successful.

Lemma 10. *If there exists $i \leq r-1$ such that $z_i^{\ell+1} = x_i^{\ell+1}$ and $z_{i+1}^{\ell+1} = x_{i+1}^{\ell+1}$, then the coupling is successful.*

Proof. We proceed by reverse induction. If $i = r-1$, there is nothing to prove. Fix $i < r-1$, and assume that the property is satisfied for $i+1$. Then, if $z_i^{\ell+1} = x_i^{\ell+1}$ and $z_{i+1}^{\ell+1} = x_{i+1}^{\ell+1}$, we simply have to prove that $z_{i+2}^{\ell+1} = x_{i+2}^{\ell+1}$ and the coupling will be successful by the induction hypothesis.

Assume first that $\texttt{Coll}'_{i+1}$ happens, namely $z_{i+1}^{\ell+1} \oplus k_{i+1}$ is equal to $x_{i+1}^{j} \oplus k_{i+1}$ for some $1 \leq j \leq \ell$ or to $u_{i+1}^{j'}$ for some $1 \leq j' \leq q_f$. In both cases we see that $F'_{i+1}(z_{i+1}^{\ell+1} \oplus k_{i+1}) = F_{i+1}(x_{i+1}^{\ell+1} \oplus k_{i+1})$, so that

$$z_{i+2}^{\ell+1} = z_i^{\ell+1} \oplus F'_{i+1}(z_{i+1}^{\ell+1} \oplus k_{i+1}) = x_i^{\ell+1} \oplus F_{i+1}(x_{i+1}^{\ell+1} \oplus k_{i+1}) = x_{i+2}^{\ell+1}\,.$$

When $\texttt{Coll}'_{i+1}$ does not happen, then $\texttt{Coll}_{i+1}$ does not happen either since we assume $x_{i+1}^{\ell+1} = z_{i+1}^{\ell+1}$, so that by definition of the coupling $F'_{i+1}(z_{i+1}^{l+1} \oplus k_{i+1})$ is chosen such that $z_{i+2}^{\ell+1} = x_{i+2}^{\ell+1}$. □

The following lemma states that if neither $\texttt{Coll}_i$ nor $\texttt{Coll}'_i$ happen for two consecutive rounds, then the coupling is successful. Note that in general we cannot use round 0 to try to couple since we cannot prevent the distinguisher from choosing $x^{\ell+1}$ such that $x_0^{\ell+1} = x_0^j$ for some $j \leq \ell$, in which case $\texttt{Coll}_0$ happens with probability 1.

Lemma 11. *For $i \in [1; r-1]$, define $A_i = \texttt{Coll}_i \cup \texttt{Coll}'_i$. Let* Fail *be the event that the coupling does not succeed. Then:*

$$\Pr[\texttt{Fail}] \leq \Pr\left[\bigcap_{i=1}^{r-2} (A_i \cup A_{i+1})\right]\,.$$

Proof. Fix $i \in [1; r-2]$. We will show that $\neg(A_i \cup A_{i+1}) \implies \neg\texttt{Fail}$. Indeed, if none of the events $\texttt{Coll}_i$, $\texttt{Coll}'_i$, $\texttt{Coll}_{i+1}$, and $\texttt{Coll}'_{i+1}$ happens, then by definition of the coupling $F'_i(z_i^{\ell+1} \oplus k_i)$ and $F'_{i+1}(z_{i+1}^{\ell+1} \oplus k_{i+1})$ are chosen such that one has $z_{i+1}^{\ell+1} = x_{i+1}^{\ell+1}$ and $z_{i+2}^{\ell+1} = x_{i+2}^{\ell+1}$. By Lemma 10, this implies that the coupling is successful. We just proved that $\neg\texttt{Fail} \supset \cup_{i=1}^{r-2} \neg(A_i \cup A_{i+1})$, which yields the result by negation. □

Hence, the probability that the coupling fails is exactly the probability of event C_{r-1} that we studied in Sect. 3. At this point, the analysis differs for the KAF and the Luby-Rackoff settings. Indeed, in the LR setting, we can show that events A_i are p-negatively dependent, whereas this does not hold in the KAF setting.

4.2 The KAF Setting

In the KAF setting, we cannot show that events A_i are p-negatively dependent. However, they satisfy some weaker form of negative dependence.

Lemma 12. *For any $i \in [1; r-1]$ and any subset $S \subseteq [1; i-2]$, one has:*

$$\Pr\left[A_i \middle| \cap_{s\in S} A_s\right] \leq \frac{2(\ell + 2q_f)}{2^n}\,.$$

Proof. We need to prove that for any $i \in [1; r-1]$ and any subset $S \subseteq [1; i-2]$, one has:

$$\Pr\left[\texttt{Coll}_i \cup \texttt{Coll}'_i \middle| \cap_{s\in S} A_s\right] \leq \frac{2(\ell + 2q_f)}{2^n}\,.$$

We upper bound the conditional probability of $\texttt{Coll}_i$, the reasoning for $\texttt{Coll}'_i$ being similar. Recall that $\texttt{XColl}_i$ is the event that $x_i^{\ell+1} \oplus k_i$ is equal to $x_i^j \oplus k_i$ for some $j \in [1; \ell]$, and $\texttt{FColl}_i$ is the event that $x_i^{\ell+1} \oplus k_i$ is equal to $u_i^{j'}$ for some $j' \in [1; q_f]$, and that $\texttt{Coll}_i = \texttt{XColl}_i \cup \texttt{FColl}_i$.

We first consider the probability of $\texttt{FColl}_i$. Since k_i is uniformly random and independent from $\cap_{s\in S} A_s$, this probability is at most $q_f/2^n$.

We now consider the probability of $\texttt{XColl}_i$, *i.e.* that $x_i^{\ell+1} \oplus k_i = x_i^j \oplus k_i$ for some $j \in [1; \ell]$. Note that this is equivalent to

$$x_{i-2}^{\ell+1} \oplus F_{i-1}(x_{i-1}^{\ell+1} \oplus k_{i-1}) = x_{i-2}^{j} \oplus F_{i-1}(x_{i-1}^{j} \oplus k_{i-1})\,. \tag{1}$$

Here, we face the problem that conditioned on $\texttt{FColl}_{i-1}$, $F_{i-1}(x_{i-1}^{\ell+1} \oplus k_{i-1})$ is not random because of the constraint $\boldsymbol{F}(u) = v$. Hence, denoting $B = \cap_{s\in S} A_s$, we write:

$$\begin{aligned}\Pr\left[\texttt{XColl}_i | B\right] &= \Pr[\texttt{XColl}_i | B \cap \texttt{FColl}_{i-1}] \Pr\left[\texttt{FColl}_{i-1} | B\right] \\ &\quad + \Pr\left[\texttt{XColl}_i | B \cap \overline{\texttt{FColl}_{i-1}}\right] \Pr\left[\overline{\texttt{FColl}_{i-1}} | B\right] \\ &\leq \Pr[\texttt{FColl}_{i-1} | B] + \Pr\left[\texttt{XColl}_i | B \cap \overline{\texttt{FColl}_{i-1}}\right]\,.\end{aligned}$$

Since k_{i-1} is random and independent from $B = \cap_{s \in S} A_s$ (recall that $S \subseteq [1; i-2]$), we have $\Pr[\mathtt{FColl}_{i-1} | B] \leq q_f/2^n$. To upper bound the second probability, note that if $x_{i-1}^{\ell+1} = x_{i-1}^{j}$, then necessarily $x_i^{\ell+1} \neq x_i^j$ since otherwise this would contradict the hypothesis that queries $x^{\ell+1}$ and x^j are distinct. If $x_{i-1}^{\ell+1} \neq x_{i-1}^{j}$, then conditioned on $\overline{\mathtt{FColl}_{i-1}}$, $F_{i-1}(x_{i-1}^{\ell+1} \oplus k_{i-1})$ is uniformly random and equation (1) is satisfied with probability at most 2^{-n} for each j, so that summing over $j \in [1; \ell]$ we obtain $\Pr\left[\mathtt{XColl}_i | B \cap \overline{\mathtt{FColl}_{i-1}}\right] \leq \ell/2^n$. Hence we have that $\Pr[\mathtt{Coll}_i] \leq (\ell + 2q_f)/2^n$. The reasoning and the bound are the same for the probability that $\mathtt{Coll}'_i$ happens, hence the result. □

Lemma 13. *Let q_e, q_f be positive integers. Then for any tuples $x \in (\{0,1\}^{2n})^{*q_e}$ and $u = (u_0, \ldots, u_{r-1})$, $v = (v_0, \ldots, v_{r-1})$ with $u_i, v_i \in (\{0,1\}^n)^{q_f}$, one has:*

$$\|\mu_{x,u,v} - \mu^*\| \leq \frac{4^t}{t+1} \frac{(q_e + 2q_f)^{t+1}}{2^{tn}} \quad \textit{with} \quad t = \left\lfloor \frac{r}{3} \right\rfloor .$$

Proof. Using successively the Coupling Lemma (Lemmas 1), Lemmas 11, and 12, one has:

$$\begin{aligned} \|\nu_\ell - \nu_\ell^*\| \leq \Pr[\mathtt{Fail}] &\leq \Pr\left[\bigcap_{i=1}^{r-2} (A_i \cup A_{i+1})\right] \\ &\leq \Pr\left[(A_1 \cup A_2)(A_4 \cup A_5) \cdots (A_{3\cdot\lfloor \frac{r}{3} \rfloor - 2} \cup A_{3\cdot\lfloor \frac{r}{3} \rfloor - 1})\right] \\ &\leq \left(\frac{4(\ell + 2q_f)}{2^n}\right)^t \quad \text{with} \quad t = \left\lfloor \frac{r}{3} \right\rfloor . \end{aligned}$$

Hence, by Lemma 9, we have for any tuples x, u, v:

$$\begin{aligned} \|\mu_{x,u,v} - \mu^*\| \leq \sum_{\ell=0}^{q_e-1} \|\nu_\ell - \nu_\ell^*\| &\leq \frac{4^t}{2^{tn}} \sum_{\ell=0}^{q_e-1} (\ell + 2q_f)^t \\ &\leq \frac{4^t}{2^{tn}} \int_{\ell=0}^{q_e} (\ell + 2q_f)^t \, d\ell \leq \frac{4^t}{t+1} \frac{(q_e + 2q_f)^{t+1}}{2^{tn}} , \end{aligned}$$

which concludes the proof. □

Finally, combining Lemmas 8 and 13, we obtain the following bound for the NCPA-security of the ideal KAF cipher.

Theorem 1. *Let q_e, q_f be positive integers. Then:*

$$\mathbf{Adv}^{\mathrm{ncpa}}_{\mathtt{KAF}[n,r]}(q_e, q_f) \leq \frac{4^t}{t+1} \frac{(q_e + 2q_f)^{t+1}}{2^{tn}} \quad \text{with} \quad t = \left\lfloor \frac{r}{3} \right\rfloor .$$

Hence, the ideal KAF cipher with r rounds ensures NCPA-security up to $\mathcal{O}(2^{\frac{tn}{t+1}})$ queries of the adversary for $t = \lfloor \frac{r}{3} \rfloor$.

4.3 The Luby-Rackoff Setting

In the Luby-Rackoff setting, events A_i can be shown to be p-negatively dependent. This will allow to use the results of Sect. 3 to upper bound the probability that the coupling fails.

Lemma 14. *In the Luby-Rackoff setting ($q_f = 0$), events $A_1, \ldots, A_{r-1}$ are p-negatively dependent for $p = \frac{2\ell}{2^n}$.*

Proof. We need to prove that for any $i \in [1; r-1]$ and any subset $S \subseteq [1; i-1]$, one has:

$$\Pr\left[\texttt{Coll}_i \cup \texttt{Coll}'_i \middle| \cap_{s\in S} A_s\right] \leq \frac{2\ell}{2^n}\,.$$

In the Luby-Rackoff setting, $q_f = 0$ so that events $\texttt{FColl}_i$ and $\texttt{FColl}'_i$ cannot happen. Hence, we simply have to consider events $\texttt{XColl}_i$ and $\texttt{XColl}'_i$. Event $\texttt{XColl}_i$ happens if $x_i^{\ell+1} \oplus k_i = x_i^j \oplus k_i$ for some $j \in [1; \ell]$. Note that this is equivalent to

$$x_{i-2}^{\ell+1} \oplus F_{i-1}(x_{i-1}^{\ell+1} \oplus k_{i-1}) = x_{i-2}^{j} \oplus F_{i-1}(x_{i-1}^{j} \oplus k_{i-1})\,.$$

If $x_{i-1}^{\ell+1} \neq x_{i-1}^j$, then this happens with probability at most 2^{-n} since in the LR setting F_{i-1} is uniformly random and independent of $\cap_{s\in S} A_s$. If $x_{i-1}^{\ell+1} = x_{i-1}^j$, then necessarily $x_i^{\ell+1} \neq x_i^j$ since otherwise this would contradict the hypothesis that queries $x^{\ell+1}$ and x^j are distinct.[1] Summing over $j \in [1; \ell]$, the probability of $\texttt{XColl}_i$ is at most $\ell/2^n$. The reasoning is similar for the probability that $\texttt{XColl}'_i$ happens, hence the result. □

This allows to use Lemma 7 to upper bound the probability that the coupling fails.

Lemma 15. *Let q_e be a positive integer. Then for any tuple $x \in (\{0,1\}^{2n})^{*q_e}$, one has:*

$$\|\mu_x - \mu^*\| \leq \sum_{t=\lfloor \frac{r-1}{2} \rfloor}^{\lfloor \frac{2r-2}{3} \rfloor} \frac{2^t}{t+1} \binom{r-t}{2r-2-3t} \frac{q_e^{t+1}}{2^{tn}}\,.$$

Proof. Using successively the Coupling Lemma (Lemma 1), Lemmas 11, and 7 combined with Lemma 14, one has (note that we apply Lemma 7 with $r-1$ rather than r):

$$\|\nu_\ell - \nu_\ell^*\| \leq \Pr[\texttt{Fail}] \leq \Pr\left[\bigcap_{i=1}^{r-2} (A_i \cup A_{i+1})\right] \leq \sum_{t=\lfloor \frac{r-1}{2} \rfloor}^{\lfloor \frac{2r-2}{3} \rfloor} \binom{r-t}{2r-2-3t} \left(\frac{2\ell}{2^n}\right)^t\,.$$

[1] Note that whether $x_{i-1}^{\ell+1}$ and x_{i-1}^j are distinct or not depends on $\cap_{s\in S} A_s$, so that the event $x_i^{\ell+1} = x_i^j$ is not independent from $\cap_{s\in S} A_s$.

Hence, by Lemma 9, we have for any tuple $x \in (\{0,1\}^{2n})^{*q_e}$:

$$\begin{aligned}\|\mu_x - \mu^*\| \le \sum_{\ell=0}^{q_e-1} \|\nu_\ell - \nu_\ell^*\| &\le \sum_{t=\lfloor \frac{r-1}{2} \rfloor}^{\lfloor \frac{2r-2}{3} \rfloor} \binom{r-t}{2r-2-3t} \sum_{\ell=0}^{q_e-1} \left(\frac{2\ell}{2^n}\right)^t \\ &\le \sum_{t=\lfloor \frac{r-1}{2} \rfloor}^{\lfloor \frac{2r-2}{3} \rfloor} \binom{r-t}{2r-2-3t} \left(\frac{2}{2^n}\right)^t \int_{\ell=0}^{q_e} \ell^t d\ell \\ &\le \sum_{t=\lfloor \frac{r-1}{2} \rfloor}^{\lfloor \frac{2r-2}{3} \rfloor} \frac{2^t}{t+1} \binom{r-t}{2r-2-3t} \frac{q_e^{t+1}}{2^{tn}},\end{aligned}$$

which concludes the proof. □

Finally, combining Lemmas 8 and 15, we obtain the following bound for the NCPA-security of the ideal LR cipher.

Theorem 2. *Let q_e be a positive integer. Then:*

$$\mathbf{Adv}^{\mathrm{ncpa}}_{\mathtt{LR}[n,r]}(q_e) \le \sum_{t=\lfloor \frac{r-1}{2} \rfloor}^{\lfloor \frac{2r-2}{3} \rfloor} \frac{2^t}{t+1} \binom{r-t}{2r-2-3t} \frac{q_e^{t+1}}{2^{tn}}.$$

The bound in this theorem is dominated by the term corresponding to $t = \lfloor (r-1)/2 \rfloor$. In particular, when $r = 2r'+1$, the coefficient of this leading term is simply $2^{r'}$, so that the dominating term is simply $2^{r'} q_e^{r'+1}/2^{r'n}$. (Incidentally, this is exactly the bound that was proved in [9] for the r'-round Even-Mansour cipher with n-bit permutations.) In other words, against NCPA-distinguishers, the ideal LR cipher is secure up to $\mathcal{O}(2^{\frac{tn}{t+1}})$ queries of the adversary with $t = \lfloor (r-1)/2 \rfloor$.

Comparison with the Hoang-Rogaway (HR) bound. In [8], Hoang and Rogaway proved the following bound for the security of the ideal Luby-Rackoff cipher $\mathtt{LR}[n,r]$:

$$\mathbf{Adv}^{\mathrm{ncpa}}_{\mathtt{LR}[n,r]}(q_e) \le \frac{4^t}{t+1} \frac{q_e^{t+1}}{2^{tn}} \quad \text{with} \quad t = \left\lfloor \frac{r}{3} \right\rfloor.$$

In a nutshell, their analysis of the coupling probability proceeds as follows: they show that the probability not to couple over three rounds is at most $4\ell/2^n$, and then iterate the process for the next three rounds, etc. In effect, they prove an additional security margin only every three rounds. Our analysis of the coupling probability is tighter: we roughly get the same bonus every two rounds, hence substantially ameliorating the security bound. For example, for three rounds, both the HR bound and our bound show that the advantage is upper bounded by $2q_e^2/2^n$ (which is exactly the original Luby-Rackoff bound). While for five rounds the HR bound does not improve, ours already shows that the advantage is upper bounded by $4q_e^3/2^{2n}$, while the HR bound yields a $\mathcal{O}(q_e^3/2^{2n})$-security bound only for six rounds. See also Fig. 3 for a concrete comparison of the two bounds once leveraged to CCA-security.

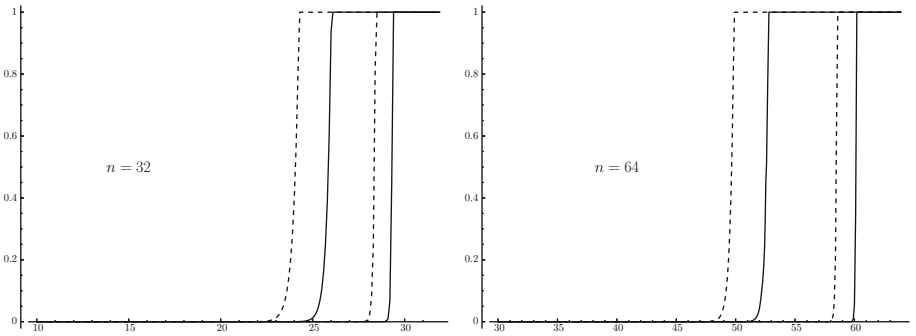


Fig. 3. Proven CCA-security for the ideal Luby-Rackoff cipher $\mathtt{LR}[n, r]$ as a function of $\log_2(q_e)$, the log of the number of adversary's queries (left: $n = 32$, right: $n = 64$). The dashed lines depict the Hoang-Rogaway bound [8], while the solid lines depict the bound proven in this paper. On each graph, the two leftmost curves are for $r = 24$ while the two rightmost curves are for $r = 96$.

4.4 Adaptive Distinguishers

In order to prove security against CCA distinguishers, we use the classical strategy (which was already used in all previous works using a coupling argument [8,9,16]) of composing two NCPA-secure ciphers. This is justified by the following lemma.

Lemma 16 ([14]). *If G and H are two blockciphers with the same message space, then for any q:*

$$\mathbf{Adv}^{\mathrm{cca}}_{H^{-1}\circ G}(q) \leq \mathbf{Adv}^{\mathrm{ncpa}}_{G}(q) + \mathbf{Adv}^{\mathrm{ncpa}}_{H}(q)\,,$$

where in $H^{-1} \circ G$ the two block ciphers are independently keyed.

Unfortunately, this result was only proved in the standard model (*i.e.* when the block ciphers do not depend on additional oracles), which allows us to use it only in the Luby-Rackoff setting.

Theorem 3. *Let q_e be a positive integer. Then:*

$$\mathbf{Adv}^{\mathrm{cca}}_{\mathtt{LR}[n,2r'-1]}(q_e) \leq \sum_{t=\lfloor \frac{r'-1}{2} \rfloor}^{\lfloor \frac{2r'-2}{3} \rfloor} \frac{2^{t+1}}{t+1} \binom{r'-t}{2r'-2-3t} \frac{q_e^{t+1}}{2^{tn}}\,.$$

Proof. Let $\mathtt{Rev}$ be the operation defined as $\mathtt{Rev}(x_L, x_R) = (x_R, x_L)$. Then, as already noticed in [13], a $(2r'-1)$-round Feistel scheme with round functions $F_0, \ldots, F_{2r'-2}$ can be written as $\mathtt{Rev} \circ H^{-1} \circ G$, where G and H are r'-round Feistel schemes. This can be seen by writing the middle round function $F_{r'-1}$ as the xor of two independent round functions $F'_{r'-1} \oplus F''_{r'-1}$ (clearly, this does not change the distribution of the outputs of the system): then G is the Feistel scheme with round functions $F_0, \ldots, F_{r'-2}, F'_{r'-1}$, while H is the Feistel scheme with round functions $F_{2r'-2}, \ldots, F_{r'}, F''_{r'-1}$. The result then follows from Lemma 16 and Theorem 2 (clearly composing with $\mathtt{Rev}$ does not change the advantage). □

For a $2r'$-round Luby-Rackoff cipher, we get the same bound as for $2r'-1$ rounds. Again, the bound in this theorem is dominated by the term corresponding to $t = \lfloor (r'-1)/2 \rfloor$. Hence, this shows that an r-round Luby-Rackoff cipher ensures CCA-security up to $\mathcal{O}(2^{\frac{tn}{t+1}})$ queries, where $t = \left\lfloor \frac{\lfloor (r+1)/2 \rfloor - 1}{2} \right\rfloor = \lfloor \frac{r-1}{4} \rfloor$.

For KAF ciphers, since we cannot apply Lemma 16 directly because the cipher depends on additional oracles, we will appeal to the same strategy as in [9], which relies on the following lemma, a refinement to Lemma 8.

Lemma 17. *Let $G^{\boldsymbol{F}}$ and $H^{\boldsymbol{F}'}$ be two block ciphers with the same message space, where $G^{\boldsymbol{F}}$ and $H^{\boldsymbol{F}'}$ depend respectively on oracles $\boldsymbol{F} = (F_0, \ldots, F_{r-1})$ and $\boldsymbol{F}' = (F'_0, \ldots, F'_{r'-1})$ (this might be arbitrary oracles, not necessarily random functions). Assume that there exists α_G such that for any tuple $x \in (\mathtt{MsgSp}(G))^{*q_e}$ and any tuples $u = (u_0, \ldots, u_{r-1})$ and $v = (v_0, \ldots, v_{r-1})$ where $u_i \in (\mathtt{Dom}(F_i))^{q_f}$ and $v_i \in (\mathtt{Rng}(F_i))^{q_f}$, one has $\|\mu^G_{x,u,v} - \mu^*\| \leq \alpha_G$, and that there exists α_H such that for any tuple $x' \in (\mathtt{MsgSp}(H))^{*q_e}$ and any tuples $u' = (u'_0, \ldots, u'_{r-1})$ and $v' = (v'_0, \ldots, v'_{r-1})$ where $u'_i \in (\mathtt{Dom}(F'_i))^{q_f}$ and $v'_i \in (\mathtt{Rng}(F'_i))^{q_f}$, one has $\|\mu^H_{x',u',v'} - \mu^*\| \leq \alpha_H$.*

(Here, $\mathtt{MsgSp}(E)$ is the message space of block cipher E, $\mathtt{Dom}(F)$ and $\mathtt{Rng}(F)$ are respectively the domain and the range of the oracle F, and the distributions are defined as in Sect. 4.1, namely $\mu^G_{x,u,v}$ is the distribution of the outputs of $G^{\boldsymbol{F}}$ when receiving inputs x, conditioned on $\boldsymbol{F}(u) = v$, and $\mu^H_{x',u',v'}$ is the distribution of the outputs of $H^{\boldsymbol{F}'}$ when receiving inputs x', conditioned on $\boldsymbol{F}'(u') = v'$.)

Then:

$$\mathbf{Adv}^{\mathrm{cca}}_{(H^{\boldsymbol{F}'})^{-1} \circ G^{\boldsymbol{F}}}(q_e, q_f) \leq 2(\sqrt{\alpha_G} + \sqrt{\alpha_H}) .$$

Proof. Deferred to the full version of the paper [10]. □

Theorem 4. *Let q_e, q_f be positive integers. Then:*

$$\mathbf{Adv}^{\mathrm{cca}}_{\mathtt{KAF}[n,2r']}(q_e, q_f) \leq 4 \left(\frac{4^t}{t+1} \frac{(q_e + 2q_f)^{t+1}}{2^{tn}} \right)^{1/2} \quad \text{with} \quad t = \left\lfloor \frac{r'}{3} \right\rfloor .$$

Proof. Since in this context the distinguisher has oracle access to the round functions, we cannot use the same trick as in the proof of Theorem 3 of writing the middle round function of a $(2r'-1)$-round Feistel scheme as the xor of two independent functions. Hence, we consider a $2r'$-round KAF cipher. First, we note that all the results of Sect. 4.1 apply *mutatis mutandis* to the inverse of a KAF cipher, *i.e.* when the state at round i is updated according $(x_L, x_R) \mapsto (x_R \oplus F_i(x_L \oplus k_i), x_L)$. Hence, we can see this $2r'$-round KAF cipher as the cascade of an r'-round KAF cipher and the inverse of the inverse of an independent r'-round KAF cipher. The result then follows directly by combining Lemmas 13 and 17. □

For a $(2r'+1)$-round KAF cipher, we get the same bound as for a $2r'$-round KAF cipher. Hence, a r-round KAF cipher ensures CCA-security up to $\mathcal{O}(2^{\frac{tn}{t+1}})$ queries in total, where $t = \left\lfloor \frac{\lfloor r/2 \rfloor}{3} \right\rfloor = \lfloor \frac{r}{6} \rfloor$.

References

1. Bogdanov, A., Knudsen, L.R., Leander, G., Standaert, F.-X., Steinberger, J., Tischhauser, E.: Key-alternating ciphers in a provable setting: encryption using a small number of public permutations. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT 2012. LNCS, vol. 7237, pp. 45–62. Springer, Heidelberg (2012)
2. Chen, S., Steinberger, J.: Tight security bounds for key-alternating ciphers. In: Nguyen, P.Q., Oswald, E. (eds.) EUROCRYPT 2014. LNCS, vol. 8441, pp. 327–350. Springer, Heidelberg (2014)
3. Daemen, J., Rijmen, V.: The Design of Rijndael: AES - The Advanced Encryption Standard. Springer, Berlin (2002)
4. Daemen, J., Rijmen, V.: Probability distributions of correlation and differentials in block ciphers. J. Math. Cryptol. **1**(3), 221–242 (2007)
5. Dunkelman, O., Keller, N., Shamir, A.: Minimalism in cryptography: the even-mansour scheme revisited. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT 2012. LNCS, vol. 7237, pp. 336–354. Springer, Heidelberg (2012)
6. Even, S., Mansour, Y.: A Construction of a Cipher from a Single Pseudorandom Permutation. J. Cryptol. **10**(3), 151–162 (1997)
7. Gentry, C., Ramzan, Z.: Eliminating random permutation oracles in the even-mansour cipher. In: Lee, P.J. (ed.) ASIACRYPT 2004. LNCS, vol. 3329, pp. 32–47. Springer, Heidelberg (2004)
8. Hoang, V.T., Rogaway, P.: On generalized feistel networks. In: Rabin, T. (ed.) CRYPTO 2010. LNCS, vol. 6223, pp. 613–630. Springer, Heidelberg (2010)
9. Lampe, R., Patarin, J., Seurin, Y.: An asymptotically tight security analysis of the iterated even-mansour cipher. In: Wang, X., Sako, K. (eds.) ASIACRYPT 2012. LNCS, vol. 7658, pp. 278–295. Springer, Heidelberg (2012)
10. Lampe, R., Seurin, Y.: Security Analysis of Key-Alternating Feistel Ciphers. Full version of this paper. http://eprint.iacr.org/2014
11. Luby, M., Rackoff, C.: How to Construct Pseudorandom Permutations from Pseudorandom Functions. SIAM J. Comput. **17**(2), 373–386 (1988)
12. Maurer, U.M.: A simplified and generalized treatment of luby-rackoff pseudorandom permutation generators. In: Rueppel, R.A. (ed.) EUROCRYPT 1992. LNCS, vol. 658, pp. 239–255. Springer, Heidelberg (1993)
13. Maurer, U.M., Pietrzak, K.: The security of many-roundluby-rackoff pseudorandom permutations. In: Biham, E. (ed.) EUROCRYPT 2003. LNCS, vol. 2656, pp. 544–561. Springer, Heidelberg (2003)
14. Maurer, U.M., Pietrzak, K., Renner, R.S.: Indistinguishability amplification. In: Menezes, A. (ed.) CRYPTO 2007. LNCS, vol. 4622, pp. 130–149. Springer, Heidelberg (2007)
15. Mironov, I.: (Not So) random shuffles of RC4. In: Yung, M. (ed.) CRYPTO 2002. LNCS, vol. 2442, pp. 304–319. Springer, Heidelberg (2002)
16. Morris, B., Rogaway, P., Stegers, T.: How to encipher messages on a small domain. In: Halevi, S. (ed.) CRYPTO 2009. LNCS, vol. 5677, pp. 286–302. Springer, Heidelberg (2009)
17. Patarin, J.: Pseudorandom permutations based on the DES scheme. In: Charpin, P., Cohen, G. (eds.) EUROCODE 1990. LNCS, vol. 514, pp. 193–204. Springer, Heidelberg (1991)
18. Patarin, J.: Security of random feistel schemes with 5 or more rounds. In: Franklin, M. (ed.) CRYPTO 2004. LNCS, vol. 3152, pp. 106–122. Springer, Heidelberg (2004)

19. Patarin, J.: Security of balanced and unbalanced Feistel Schemes with Linear Non Equalities (2010). http://eprint.iacr.org/2010/293
20. Ramzan, Z., Reyzin, L.: On the round security of symmetric-key cryptographic primitives. In: Bellare, M. (ed.) CRYPTO 2000. LNCS, vol. 1880, pp. 376–393. Springer, Heidelberg (2000)
21. Steinberger, J.: Improved Security Bounds for Key-Alternating Ciphers via Hellinger Distance. IACR Cryptology ePrint Archive, Report 2012/481 (2012). http://eprint.iacr.org/2012/481
22. Vaudenay, S.: Decorrelation: A Theory for Block Cipher Security. J. Cryptol. **16**(4), 249–286 (2003)

The Related-Key Analysis of Feistel Constructions

Manuel Barbosa[1(✉)] and Pooya Farshim[2]

[1] HASLab – INESC TEC and Universidade do Minho, Braga, Portugal
mbb@di.uminho.pt
[2] Fachbereich Informatik, Technische Universität Darmstadt, Darmstadt, Germany
farshim@cased.de

Abstract. It is well known that the classical three- and four-round Feistel constructions are provably secure under chosen-plaintext and chosen-ciphertext attacks, respectively. However, irrespective of the number of rounds, no Feistel construction can resist related-key attacks where the keys can be offset by a constant. In this paper we show that, under suitable reuse of round keys, security under related-key attacks can be provably attained. Our modification is simpler and more efficient than alternatives obtained using generic transforms, namely the PRG transform of Bellare and Cash (CRYPTO 2010) and its random-oracle analogue outlined by Lucks (FSE 2004). Additionally we formalize Luck's transform and show that it does not *always* work if related keys are derived in an oracle-dependent way, and then prove it sound under appropriate restrictions.

Keywords: Feistel construction · Luby–rackoff · Related-key attack · Pseudorandom permutation · Random oracle

1 Introduction

Cryptographic algorithms deployed in the real world are subject to a multitude of threats. Many of these threats are accounted for in the theoretical security analysis carried out by cryptographers, but not all. Indeed, many documented cases [14,15,32,39] show that theoretically secure cryptographic algorithms can be vulnerable to relatively simple physical attacks, when these exploit implementation aspects that were abstracted away in the security analysis. For this reason, an enormous research effort has been undertaken in recent years to bridge the gap between physical security and theoretical security.

An important part of this effort has been dedicated to *related-key attacks* (RKA), which were first identified by Knudsen and Biham [9,27] as an important risk on implementations of block ciphers and symmetric-key cryptosystems. The idea behind these attacks is as follows. The security of cryptographic algorithms depends fundamentally on keeping secret keys hidden from attackers for extended periods of time. For this reason, secret keys are typically stored and

C. Cid and C. Rechberger (Eds.): FSE 2014, LNCS 8540, pp. 265–284, 2015.
DOI: 10.1007/978-3-662-46706-0_14

manipulated in protected memory areas and dedicated hardware components. If these mechanisms can be influenced by intrusive techniques (such as fault injection [2]) an adversary may be able to disturb the value of a secret key and observe results computed using the manipulated (likely correlated) key value.

Since the original work of Knudsen and Biham, there have been many reported cases of successful related-key cryptanalysis [8,10,28], and notably of the Advanced Encryption Standard (AES) [11,12]. These results led to the consensual view that RKA resilience should be a standard design goal for low-level cryptographic primitives such as block ciphers and hash functions. For example, in the recent SHA-3 competition, candidates were analyzed with respect to such attacks (c.f. the work of Khovratovich et al. [26]), which played an important role in the selection process.

The importance of including RKA security as a design goal for basic cryptographic components is further heightened by the fact that such low-level primitives are often *assumed* to provide RKA security when used in higher-level protocols. Prominent examples are the key derivation procedures in standard protocols such as EMV [16] and the 3GPP integrity and confidentiality algorithms [25], where efficiency considerations lead to the use of the same block cipher under closely related keys. Similar assumptions arise in constructions of *tweakable ciphers* [29], where a block cipher is called on keys which are offset by XOR-ing tweak values.

Provable RKA security. Bellare and Kohno [6] initiated the theoretical treatment of security under related-key attacks by proposing definitions for RKA-secure pseudorandom functions (PRFs) and pseudorandom permutations (PRPs), and presenting possibility and impossibility results for these primitives. The models proposed in [6] were extended by Albrecht et al. [1] to address the possibility of oracle-dependent attacks in idealized models of computation.

Various important positive results for provably RKA-secure constructions of complex cryptographic primitives were subsequently published in the literature. Bellare and Cash [4] obtained a breakthrough result by presenting a concrete construction of an RKA-secure pseudorandom function based on standard computational assumptions and in the standard model. Bellare, Cash, and Miller [5] present a comprehensive treatment of RKA security for various cryptographic primitives, focusing on the problem of leveraging the RKA resilience of one primitive to construct RKA-secure instances of another. In particular, Bellare et al. present a generic transformation in which an RKA-secure pseudorandom generator can be used to convert instances of standard primitives such as digital signatures and identity-based encryption into RKA-secure ones. Concrete constructions of RKA-secure public-key primitives were given by Wee and by Bellare et al. in [7,42].

Feistel networks. A Feistel network [17,18] is a construction that permits obtaining an efficiently computable and invertible permutation from an efficiently computable function. The network is a cascade of simple Feistel permutations, each relying on a *round function* (f, g, and h) mapping bit strings of length n to outputs of the same length. Here the input and output are shown as tuples

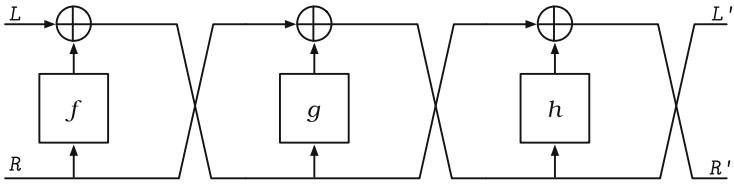


Fig. 1. A three-round Feistel network.

(L, R) and (L', R'), where each component is a string of length n. For any number of rounds, these networks provide an invertible permutation over bit strings of length $2n$. Figure 1 shows an example of a Feistel network with three rounds.

Feistel networks (and generalized variants such as those discussed by Hoang and Rogaway in [23]) have been extensively used in the construction of symmetric cryptosystems (and even asymmetric ones such as RSA-OAEP), since the notable case of the Data Encryption Standard (DES) in the 1970s [18]. In particular, a multitude of block ciphers include Feistel-like constructions in their design, including GOST, MYSTY1, Skipjack, BEAR / LION, CAST-256, RC6, and MARS [38]. For this reason, the security properties of Feistel networks received significant attention in the last decades.

Security of the Feistel construction. In their seminal paper, Luby and Rackoff [30] showed that instantiating the round functions in a Feistel construction with independently keyed secure PRFs is sufficient to obtain a secure PRP. For three rounds of cascading, this result applies when the adversary has access to results of forward computations (i.e., under chosen-plaintext attacks), and for four rounds, the result holds even if the adversary can additionally observe the results of inverse computations (i.e., under chosen-ciphertext attacks).

Following Luby and Rackoff's result, many subsequent works looked at the security of Feistel networks and generalized variants thereof. Important results were obtained with respect to the efficiency of the construction, for example by reducing the necessary key material (c.f. the work of Patarin [36]) and by weakening the security assumptions for some of the round functions as in the work of Naor and Reingold in [35]. In a different direction, the security offered by Feistel networks with increasing numbers of rounds was precisely characterized in a sequence of works by Vaudenay [41], Maurer and Pietrzak [33], Patarin [37] and Hoang and Rogaway [23]. Holenstein, Künzler, and Tessaro [24] used the Feistel construction with fourteen rounds to establish the equivalence of the random-oracle and the ideal-cipher models in a broad range of applications via the indifferentiability framework.

RKA security of Feistel networks. Despite this large body of work on the provable security of the Feistel construction and the positive results on the RKA security of advanced cryptographic primitives referred above, the RKA security of the Feistel construction has received little attention. Indeed, to the best of our knowledge, only the work of Bellare and Kohno [6] touches upon this topic, where a strong negative result is shown: the Feistel construction *irrespective of*

the number of rounds is vulnerable to related-key attacks, provided that the attacker is able to modify as little as a single bit in the key used in the last round function.[1]

Referring to Fig. 1, the attacker would proceed as follows. It would first observe the output (L'_1, R'_1) of the permutation computed on an input (L, R). Then, the adversary would modify round function h to some other function h' by manipulating its key, and observe the output (L'_2, R'_2) computed over the same input. The adversary can now determine whether it is interacting with an ideal permutation or not: If interacting with Feistel, the outputs will *always* satisfy $L'_1 = L'_2$, whereas in for an ideal (keyed) permutation the two outputs will be different with overwhelming probability. This attack is possible whenever the adversary is able to independently tweak the round function of the output stage in the network, independently of the number of rounds, and even if the round functions are instantiated with RKA-secure PRFs.

This vulnerability is relevant for practical applications of Feistel constructions, since many important cryptanalytic results such as those presented by Biryukov et al. [11,12] can be described as utilizing related keys that are derived by XOR-ing the original key with a constant. This in particular permits an attacker to selectively modify the secret key for the output round in a Feistel network and break the security of the construction. In this work we initiate the treatment of provable RKA security of the Feistel constructions. Our main result is to prove is that specific instances of Feistel networks that reuse round keys offer *intrinsic* RKA security against practically relevant classes of RKD functions, and thus overcome the negative result by Bellare and Kohno described above. We now present our contributions in more detail.

CONTRIBUTIONS. Lucks [31] proposes a general solution to the RKA security of any cryptographic primitive in the random-oracle model: hash the secret key before applying it to the cryptosystem. The intuition is that, modeling the hash function as a random oracle, any modification to the secret key will result in a new independent key to be used in the cryptosystem, confining the RKA adversary to standard attacks. The RKA-secure PRG transform of Bellare, Cash, and Miller (BCM) [5] that we discussed above can be seen as a special standard-model analogue of this transform. Somewhat surprisingly, we show that the original random oracle transform does not *always* result in an RKA-secure construction. We amend this by first showing that, under certain restrictions on the RKD set, the random oracle is an RKA-secure PRG, and then extending the BCM result to the random-oracle model. The set of necessary restrictions is permissive enough to include offsetting keys by constants (even if those keys were hashed!) as a particular case. This solution, however, in addition to relying on strong assumptions on the hash function, gives rise to decreased efficiency with respect to the original primitive.

Moreover, the above result only applies to a transformed construction and says nothing about the RKA security of Feistel constructions (which could be

[1] Note that this does not contradict the aforementioned fourteen-round indifferentiability result as the RKA security game is multi-stage.

present in the construction of the hash function itself!). We therefore revisit the Bellare–Kohno (BK) negative result and complement it by characterizing the class of RKA-attacks that *can* be sustained by three and four rounds Feistel networks with independent round keys (i.e., the original Luby–Rackoff constructions). The class of tolerated attacks is highly restrictive and, in particular, it excludes the XOR-with-constants set. (This was to be expected, since the BK attack can be launched using these RKD functions.)

We next consider variants of Feistel constructions in which the keys to round functions in different stages of the network may be *reused*. These variants were already proposed in the literature (c.f. the work by Patarin [36]) due to the efficiency and security benefits of reducing the necessary secret key material. However, we observe that key reuse has the added effect of *limiting* the power of an RKA-adversary in targeting individual round keys. We build on this intuition to obtain our main results: we show that Feistel networks with three (respectively four) rounds can be proven CPA (respectively CCA) RKA secure by relying on an RKA-secure PRF and using specific key assignments that reuse some of the round keys.

Intuitively, our selection of key reusing assignments can be described as follows. It is well known that reusing the same keys in all rounds of the Feistel network or, more generally, any palindromic assignment of the keys, leads to totally insecure constructions. Also, the BK attack rules out key assignments where the key to the output round (in both forward and inverse computations) can be independently thwarted. These restrictions leave few plausible key assignments for intrinsic RKA security of three- and four-round Feistel networks. From these candidates we selected two specific assignments based on *two* PRF keys K_1 and K_2: we consider the key assignment (K_1, K_2, K_2) for the three-round variant, and the (K_1, K_2, K_1, K_2) key assignment for the four-round variant. We prove that the three-round variant is CPA secure and that the four-round variant is CCA secure, both in the RKA setting, assuming that the underlying PRF is RKA secure, and that the RKD set satisfies natural restrictions akin to those adopted, e.g., in [6].

Our results require no other modification to the original constructions in addition to the key assignment and therefore come with minimal modifications to deployed implementations.[2] Put differently, we are able to prove the RKA security of the three-stage (CPA) and four-stage (CCA) Luby–Rackoff constructions, whilst *reducing* the amount of key material and therefore potentially improving the efficiency of the resulting implementations.

For practical applications, the most important aspect of our results is perhaps that they cover the standard classes RKD functions considered in literature, namely those which offset the key by XOR-ing a constant. However, for the sake of generality our presentation relies on a slightly more abstract framework, where we characterize the covered classes of covered RKD functions by defining a set of sufficient restrictions that they must satisfy. This approach also enables a clearer and more modular presentation. For example, as an intermediate step,

[2] Albeit imposing a stronger security assumption on the underlying PRF.

we formalize a notion of multi-key RKA security that may be of independent interest, and relate it to the standard single-key variant.

From a foundational perspective, our result can be seen as one bringing RKA security analysis to the classical constructions of pseudorandom objects. Goldberg and Liskov [19] study this question for building RKA-secure pseudorandom generators (where the seed is interpreted as the key) from one-way functions via Goldreich–Levin [20]. However, the natural questions of transforming RKA-secure PRGs to RKA-secure PRFs via the GGM construction [21] or RKA-secure PRFs to PRPs via the Luby–Rackoff constructions [30] have not been addressed yet. Our results can been seen as giving a positive answer to the latter question.

2 Preliminaries

NOTATION. We write $x \leftarrow y$ for the action of assigning the value y to the variable x. We write $x_1, \ldots, x_n \leftarrow\!\!\$\ \mathsf{X}$ for sampling $x_1, \ldots, x_n$ from a finite set X uniformly at random. If $\mathcal{A}$ is a probabilistic algorithm we denote the action of running $\mathcal{A}$ on inputs $x_1, \ldots, x_n$ with independently chosen coins, and assigning the result to $y_1, \ldots, y_n$ by $y_1, \ldots, y_n \leftarrow\!\!\$\ \mathcal{A}(x_1, \ldots, x_n)$. For a vector $\mathbf{x} = (x_1, \ldots, x_n)$, we define $\mathbf{x}|_i = x_i$. We let $[n] := \{1, \ldots, n\}$. A function $\epsilon(\lambda)$ is negligible if $|\epsilon(\lambda)| \in \lambda^{-\omega(1)}$. PPT as usual abbreviates probabilistic polynomial-time.

KEYED FUNCTIONS AND PERMUTATIONS. Let Dom_λ, Rng_λ, and KSp_λ be three families of finite sets parametrized by a security parameter $\lambda \in \mathbb{N}$. We denote the set of all functions $\rho : \mathsf{Dom}_\lambda \longrightarrow \mathsf{Rng}_\lambda$ by $\mathrm{Func}(\mathsf{Dom}_\lambda, \mathsf{Rng}_\lambda)$. A keyed function is a set of functions in $\mathrm{Func}(\mathsf{Dom}_\lambda, \mathsf{Rng}_\lambda)$ indexed by the elements of the key space KSp_λ. We denote the set of all keyed functions by $\mathrm{Func}(\mathsf{KSp}_\lambda, \mathsf{Dom}_\lambda, \mathsf{Rng}_\lambda)$. By the ideal keyed function, we mean the family of distributions corresponding to choosing a function uniformly at random from $\mathrm{Func}(\mathsf{KSp}_\lambda, \mathsf{Dom}_\lambda, \mathsf{Rng}_\lambda)$. The random oracle is the ideal keyed function where KSp_λ for each $\lambda \in \mathbb{N}$ contains a single key. We denote the set of all permutations on Dom_λ by $\mathrm{Perm}(\mathsf{Dom}_\lambda)$. Note that each permutation uniquely defines its inverse permutation (which is also a member of this set). We define a family of keyed permutations analogously by indexing a set of permutations according to keys in some space KSp_λ. We denote the set of all such keyed permutations by $\mathrm{Perm}(\mathsf{KSp}_\lambda, \mathsf{Dom}_\lambda)$. The ideal keyed permutation (a.k.a. the ideal cipher) is defined as the family of distributions that choose a random element of $\mathrm{Perm}(\mathsf{KSp}_\lambda, \mathsf{Dom}_\lambda)$.

PSEUDORANDOM FUNCTION AND PERMUTATION FAMILY. A pseudorandom function family $\mathsf{PRF} := \{\mathsf{PRF}_\lambda\}_{\lambda \in \mathbb{N}}$ is a family of efficiently implementable keyed functions, i.e., functions $\mathsf{PRF}_\lambda : \mathsf{KSp}_\lambda \times \mathsf{Dom}_\lambda \longrightarrow \mathsf{Dom}_\lambda$, where PRP_λ can be computed in polynomial time in λ, together with an efficient procedure for sampling of keys and domain points which by a slight abuse of notation we denote by $\mathsf{KSp}(1^\lambda)$ and $\mathsf{Dom}(1^\lambda)$, respectively. A pseudorandom permutation family is defined analogously with the extra requirement that the inverse of each permutation in the family is also efficiently computable.

3 RKA-Secure Pseudorandom Functions and Permutations

In this section we introduce the formal framework in which we will analyze the RKA security of Feistel constructions. We begin by formalizing the notion of a family of related-key deriving (RKD) functions, which will parametrize our RKA security notions. Subsequently we introduce a generalization of the standard security model for RKA-secure pseudorandom functions and permutations to a scenario where multiple secret keys may be present in the system and influence the secret key derived by an RKD function. This is the natural setting for analyzing Feistel networks, as they use multiple instances of the same PRF.

Family of RKD sets. A family of n-ary related-key deriving (RKD) sets Φ is a family of RKD sets $\{\Phi_\lambda\}$ consisting of RKD functions ϕ (viewed as circuits) which map an n-tuple of keys in some key space KSp_λ to a new key in KSp_λ, i.e., $\phi : \mathsf{KSp}_\lambda^n \to \mathsf{KSp}_\lambda$. Throughout the paper we assume that membership in any RKD set can be efficiently decided.

Multi-key RKA security. Let $\mathsf{PRP} := \{\mathsf{PRP}_\lambda : \mathsf{KSp}_\lambda \times \mathsf{Dom}_\lambda \longrightarrow \mathsf{Dom}_\lambda\}$ be a PRP family and let $\Phi := \{\Phi_\lambda\}$ be a family of n-ary RKD sets where the implicit key space of the RKD functions in Φ_λ is KSp_λ. Let game $\mathsf{RKCCA}_{\mathsf{PRP},\mathcal{A},\Phi}(1^\lambda)$ be as shown in Fig. 2. We say that PRP is Φ-RKCCA secure if the advantage of any legitimate ppt adversary $\mathcal{A}$ defined as

$$\mathbf{Adv}^{\mathsf{rkcca}}_{\mathsf{PRP},\mathcal{A},\Phi}(\lambda) := 2 \cdot \Pr\left[\mathsf{RKCCA}_{\mathsf{PRP},\mathcal{A},\Phi}(1^\lambda)\right] - 1$$

is negligible as a function of λ. An adversary is legitimate if it queries the RKFn and RKFn^{-1} oracles with functions ϕ in Φ_λ only.[3] We say PRP is Φ-RKCPA secure if the above advantage is negligible for any legitimate ppt adversary $\mathcal{A}$ that never queries its RKFn^{-1} oracle.

In the full version [3] of this paper we prove that under the following natural (but strong) restriction on RKD sets, the single-key and multi-key RKA models are equivalent: we impose that any $\phi \in \Phi_\lambda$ is of the form $\phi : (K_1, \ldots, K_n) \mapsto \psi(K_i)$, where $i \in [n]$ and $\psi : \mathsf{KSp}_\lambda \longrightarrow \mathsf{KSp}_\lambda$ is a unary RKD function.

$\underline{\mathsf{RKCCA}_{\mathsf{PRP},\mathcal{A},\Phi}(1^\lambda)}$:	$\underline{\mathrm{RKFn}(\phi, x)}$:	$\underline{\mathrm{RKFn}^{-1}(\phi, x)}$:
$b \leftarrow\$ \{0,1\}$	$K' \leftarrow \phi(K_1, \ldots, K_n)$	$K' \leftarrow \phi(K_1, \ldots, K_n)$
$\pi \leftarrow\$ \mathrm{Perm}(\mathsf{KSp}_\lambda, \mathsf{Dom}_\lambda)$	If $b = 0$ Return $\pi(K', x)$	If $b = 0$ Return $\pi^{-1}(K', x)$
$K_1, \ldots, K_n \leftarrow\$ \mathsf{KSp}(1^\lambda)$	Return $\mathsf{PRP}(K'x)$	Return $\mathsf{PRP}^{-1}(K', x)$
$b' \leftarrow\$ \mathcal{A}^{\mathrm{RKFn}, \mathrm{RKFn}^{-1}}(1^\lambda)$		
Return $(b' = b)$		

Fig. 2. Game defining the Φ-RKCCA security of a PRP.

[3] Throughout the paper, we assume all the adversaries are, in this sense, legitimate.

REMARK. The multi-key RKA model for PRFs (under chosen-plaintext attacks) is recovered when π is sampled from $\mathrm{Func}(\mathsf{KSp}_\lambda, \mathsf{Dom}_\lambda, \mathsf{Rng}_\lambda)$ and oracle RKFN^{-1} is no longer present. When $n = 1$, we recover the single-key RKA model for PRPs and PRFs as in [6]. The standard model for PRPs/PRFs is one where the RKD sets Φ_λ contain the identity functions $id_\lambda : \mathsf{KSp}_\lambda \longrightarrow \mathsf{KSp}_\lambda; K \mapsto K$ only. The above definition is not the strongest multi-key security model that one can envision. (For instance consider a model where the adversary can choose the arity n.) However, since the applications that we will be considering in this paper have a fixed number of keys, the simpler definition above is sufficient for our purposes.

4 The Random-Oracle Transform

One way to transform a standard pseudorandom permutation to one which resists related-key attacks is to hash the PRP key before using it in the construction [31]. We call this the "Hash-then-PRP" transform. Bellare and Cash [4, Theorem 6.1] prove the soundness of this approach in the *standard* model for a restricted class of RKD functions, when the hash function is replaced by an RKA-secure pseudorandom generator. At first sight it appears that an ideal hash function (i.e., the random oracle) should be a valid instantiation of this construction. However, in the random-oracle model (ROM) the security proof should be carried out in a setting where *all* parties have access to the random oracle (which models the hash function). In this section we consider the implications of this observation, and show that the random oracle does *not* always give rise to a good instantiation of the construction. We provide a set of sufficient conditions that allows us to formally prove that the heuristic transform is sound in the ROM.

RKA-SECURE PRG IN ROM.[4] We define an *oracle* RKD function to be a circuit which contains special oracle gates, and we write an n-ary oracle RKD function as $\phi^H : \mathsf{KSp}^n \to \mathsf{KSp}$. Families of oracle RKD sets are defined in the obvious way.

Let $\mathsf{PRG}^H : \mathsf{Dom} \longrightarrow \mathsf{Rng}$ be a pseudorandom generator in the ROM. Let game $\mathsf{RKA}_{\mathsf{PRG},\mathcal{A},\Phi}$ be as shown in Fig. 3. We say that PRG is Φ-RKA secure if

$\underline{\mathrm{RKA}_{\mathsf{PRG},\mathcal{A},\Phi}(1^\lambda)}$:	$\underline{\mathrm{RKFN}(\phi)}$:
$\rho \leftarrow\$ \mathrm{Func}(\mathsf{Dom}, \mathsf{Rng})$	$K' \leftarrow \phi^H(K_1, \ldots, K_n)$
$H \leftarrow\$ \mathrm{Func}(\mathsf{Dom}', \mathsf{Rng}')$	If $b = 0$ Return $\rho(K')$
$K_1, \ldots, K_n \leftarrow\$ \mathsf{Dom}(1^\lambda)$	Return $\mathsf{PRG}^H(K')$
$b \leftarrow\$ \{0, 1\}$	
$b' \leftarrow\$ \mathcal{A}^{\mathrm{RKFN},\mathrm{RO}}(1^\lambda)$	$\underline{\mathrm{RO}(X)}$:
Return $(b' = b)$	Return $H(X)$

Fig. 3. Game defining the Φ-RKA security of a PRG. An adversary is legitimate if it queries RKFN with a $\phi \in \Phi_\lambda$ only.

[4] We remark that this game can also be seen as extension of correlated-input secure hashing [22] to the random-oracle model.

the advantage of any PPT adversary $\mathcal{A}$ as defined below is negligible in λ.

$$\mathbf{Adv}^{\mathsf{rka}}_{\mathsf{PRG},\Phi,\mathcal{A}}(\lambda) := 2 \cdot \Pr\left[\mathsf{RKA}_{\mathsf{PRF},\mathcal{A},\Phi}(1^\lambda)\right] - 1 \ .$$

The question that we wish to answer is under which conditions does the random oracle itself (i.e., when $\mathsf{PRG}^H(X) := H(X)$) constitute an RKA-secure PRG. The attack we now show and the ensuing discussion demonstrate that this is only the case if we exclude certain forms of *oracle-dependent* related-key attacks.

The attack. Consider a unary RKD set containing the identity function and an oracle-dependent RKD function ϕ^H [1]: $\Phi := \{id : K \mapsto K,\ \phi^H : K \mapsto H(K)\}$. Here, H denotes the random oracle. Now consider an adversary that first requests a PRG value of the seed by querying id to the RKFn oracle. It receives as response a value y which is either $H(K)$, when $b = 1$, or $\rho(K)$ when $b = 0$, where ρ is an independent random oracle. The adversary now queries y to RO to get a new value z which is either $H(H(K))$ or $H(\rho(K))$. Finally, the adversary queries ϕ^H to RKFn to get a value z' which is either $H(H(K))$ or $\rho(H(K))$. Now, when $b = 1$, then $z = z'$ with probability 1. When $b = 0$ the values z and z' would only match if $H(\rho(K)) = \rho(H(K))$. The probability of this event is negligible, so the adversary wins with overwhelming probability by returning $(z = z')$.

We now define a sufficient set of restrictions on oracle RKD sets that allow us to prove a ROM analogue of the result by Bellare and Cash [4]. Intuitively the restrictions are strong enough to rule out attacks that follow the above pattern.

Output unpredictability. A family of oracle RKD sets Φ is output unpredictable (UP) if the following definition of advantage is negligible in λ for any PPT adversary $\mathcal{A}$ outputting a list of RKD functions and a list of keys.

$$\mathbf{Adv}^{\mathsf{up}}_{\mathcal{A},\Phi}(\lambda) := \Pr\left[\exists\ (\phi, \mathbf{K}^*) \in \mathsf{L}_1 \times \mathsf{L}_2 \ \text{ s.t. } \ \phi^H(\mathbf{K}) = \mathbf{K}^* : H \leftarrow\!\!\$\ \mathsf{Func}(\mathsf{KSp}, \mathsf{KSp}); \mathbf{K} \leftarrow\!\!\$\ \mathsf{KSp}^n; (\mathsf{L}_1, \mathsf{L}_2) \leftarrow\!\!\$\ \mathcal{A}^H(1^\lambda)\right]$$

Claw-freeness. A family of oracle RKD sets Φ is claw-free (CF) if the following definition of advantage is negligible in λ for any PPT adversary $\mathcal{A}$ outputting a list of RKD functions.

$$\mathbf{Adv}^{\mathsf{cf}}_{\mathcal{A},\Phi}(\lambda) := \Pr\left[\exists\ \phi_1^H, \phi_2^H \in \mathsf{L} \ \text{ s.t. } \ \phi_1^H(\mathbf{K}) = \phi_2^H(\mathbf{K}) \wedge \phi_1^H \neq \phi_2^H : H \leftarrow\!\!\$\ \mathsf{Func}(\mathsf{KSp}, \mathsf{KSp}); \mathbf{K} \leftarrow\!\!\$\ \mathsf{KSp}^n; \mathsf{L} \leftarrow\!\!\$\ \mathcal{A}^H(1^\lambda)\right]$$

Query independence. A family of oracle RKD sets Φ is query independent (QI) if the following definition of advantage is negligible in λ for any PPT adversary $\mathcal{A}$ outputting a list of RKD functions.

$$\mathbf{Adv}^{\mathsf{qi}}_{\mathcal{A},\Phi}(\lambda) := \Pr\left[\exists \phi_1^H, \phi_2^H \in \mathsf{L} \ \text{ s.t. } \ \phi_1^H(\mathbf{K}) \in \mathsf{Qry}[\phi_2^H(\mathbf{K})] : H \leftarrow\!\!\$\ \mathsf{Func}(\mathsf{KSp}, \mathsf{KSp}); \mathbf{K} \leftarrow\!\!\$\ \mathsf{KSp}^n; \mathsf{L} \leftarrow\!\!\$\ \mathcal{A}^H(1^\lambda)\right]$$

Here, $\mathsf{Qry}[\phi_2^H(\mathbf{K})]$ denotes the set of queries placed to H by ϕ_2^H when run on a vector of keys $\mathbf{K}$. Note that RKD functions ϕ_1^H and ϕ_2^H need not be distinct.

We recover the standard (non-oracle) definition of output unpredictability and claw-freeness [6], when the RKD functions do not make any oracle queries: the random oracle can be simulated using lazy sampling. Query independence is trivially satisfied for such non-oracle RKD functions.

We now prove that the random oracle is an RKA-secure pseudorandom generator under the above restrictions on the oracle RKD set, and then build on this result to establish security of the Hash-then-PRP transform in the random oracle model. Looking ahead, this result allows us to take a Luby–Rackoff PRP and generically transform it to obtain an RKA-secure PRP. In subsequent sections we will explore less intrusive, more efficient alternatives that take advantage of the inner structure of the Feistel construction.

Theorem 1 (RKA Security of the Random Oracle). *Let Φ be a family of oracle RKD sets. For any Φ-RKCCA adversary $\mathcal{A}$ against the pseudorandom generator $\mathsf{PRG}^H(K) := H(K)$, there are adversaries $\mathcal{A}_1$, $\mathcal{A}_2$, and $\mathcal{A}_3$ such that*

$$\mathbf{Adv}^{\mathsf{rkcpa}}_{\mathsf{PRG},\mathcal{A},\Phi}(\lambda) \leq \mathbf{Adv}^{\mathsf{up}}_{\mathcal{A}_1,\Phi}(\lambda) + 2 \cdot \mathbf{Adv}^{\mathsf{cf}}_{\mathcal{A}_2,\Phi}(\lambda) + \mathbf{Adv}^{\mathsf{qi}}_{\mathcal{A}_3,\Phi}(\lambda) \ ,$$

Proof (Sketch). We give only the intuition; the details of the proof can be found in the full version. Assume, without loss of generality, that the adversary never places repeat queries to its RKFn and RO oracles. Let Game_0 denote the RKA game where H is used in the RKFn oracle (i.e., the challenge bit is 1).

We modify Game_0 to Game_1 by implementing the H oracle in the RKFn oracle in a forgetful way (i.e., we won't keep track of repetitions), but leaving it unchanged for the explicit queries made through RO and the indirect queries placed by the oracle RKD functions. Note that in this game the adversary receives independently and uniformly distributed strings from either of its oracles.

Games Game_0 and Game_1 are identical unless one of the following events takes place: (1) A repeat H query is placed as a result of an explicit RO query and an output of an oracle RKD function queried to RKFn: this leads to a violation of the output unpredictability. (2) There is a repeat query to H as a result of two distinct RKFn queries: this leads to a claw-freeness break. (3) There is a repeat H query as a result of a query to RKFn and an indirect query placed by an oracle RKD function to H: this breaks the query-independence property.

We now modify Game_1 to Game_2 by changing the forgetful oracle and implementing it using an independently chosen (non-forgetful) random oracle. The games are identical unless there is a claw among the RKD functions queried to RKFn, which by the above analysis happens with negligible probability. Finally note that Game_2 is identical to the RKA game conditioned on $b = 0$.[5] □

In the full version we state and prove the analogue of the RKA-secure PRG transform of Bellare, Cash, and Miller [5], which in combination with Theorem 1 establishes security of the Hash-then-PRP transform in the random oracle model.

[5] This transition my be avoided by observing that Game_0 and Game_2 are also identical until the same bad events which separate Game_0 and Game_1.

5 The Feistel Construction

In this section we recall the formal definitions related to the Feistel constructions and introduce the notion of key assignment. We also establish a general result that permits shifting the analysis of Feistel networks with any number of rounds where the round functions are instantiated with an RKA-secure PRF to a more convenient setting where the round functions are instantiated with the ideal keyed function.

Feistel networks. The one-round Feistel construction and its inverse with respect to a function f is defined as

$$\mathbf{F}[f](L,R) := (R, L \oplus f(R)) \quad \text{and} \quad \mathbf{F^{-1}}[f](L,R) := (R \oplus f(L), L) \ .$$

The n-round Feistel construction with respect to functions $f_1, \ldots, f_n$ is defined recursively via the following equations (see Fig. 1 for a pictorial representation).

$$\begin{aligned} \mathbf{F}[f_1, \ldots, f_n](L,R) &:= \mathbf{F}[f_2, \ldots, f_n](\mathbf{F}[f_1](L,R)) \ , \\ \mathbf{F^{-1}}[f_1, \ldots, f_n](L,R) &:= \mathbf{F^{-1}}[f_1, \ldots, f_{n-1}](\mathbf{F^{-1}}[f_n](L,R)) \end{aligned}$$

Typically, functions $f_i(\cdot)$ are implemented using a PRF under independently generated keys $K_1, \ldots, K_n$. In our analysis we will also consider the conceptual setting in which these functions are instantiated by an ideal keyed function ρ, again under independently generated keys $K_1, \ldots, K_n$. In this case we denote the constructions by $\mathbf{F}^{\mathsf{PRF}}[K_1, \ldots, K_n]$ and $\mathbf{F}^{\rho}[K_1, \ldots, K_n]$, respectively.

Key assignment. A key assignment is a family of circuits $\kappa_\lambda : \overline{\mathsf{KSp}}_\lambda \longrightarrow \mathsf{KSp}^n$, where $\overline{\mathsf{KSp}}$ is an arbitrary key space. Given $\kappa := \{\kappa_\lambda\}$ and $K \in \overline{\mathsf{KSp}}_\lambda$, we consider the associated n-round Feistel construction $\mathbf{F}^{\mathsf{PRF}}[\kappa(K)]$. When the key $K \in \overline{\mathsf{KSp}}_\lambda$ is randomly generated, we denote the construct by $\mathbf{F}^{\mathsf{PRF}}[\kappa]$. For example, the Hash-then-PRP transform of the previous section can be viewed as $\mathbf{F}^{\mathsf{PRF}}[H]$. We are, however, interested in simple key assignments of the form $\kappa : (K_1, \ldots, K_m) \mapsto (K_{i_1}, \ldots, K_{i_n})$, where $i_1, \ldots, i_n$ are fixed indices in $[m]$. We will therefore compactly write the Feistel construction associated to the simple key assignment above by $\mathbf{F}^{\mathsf{PRF}}[i_1, \ldots, i_n]$. For example, when $\kappa(K_1, K_2) := (K_1, K_2, K_2)$, the associated Feistel construction is written as $\mathbf{F}^{\mathsf{PRF}}[1, 2, 2]$.

When the round functions in a 3-round Feistel construction are instantiated with a PRF under independent keys, we obtain the classic CPA-secure Luby–Rackoff pseudorandom permutation. When 4 rounds are used, we obtain its CCA-secure counterpart. As stated in the introduction, Bellare and Kohno [6] observed that if an adversary can arbitrarily tamper with the key used in the last round of any Feistel network, then a successful related-key attack is possible (even if the underlying PRF is RKA secure).

As discussed in the previous section, by applying the Hash-then-PRP transform to the Luby–Rackoff construction, we can obtain a PRP which resists related-key attacks. The underlying PRG can be instantiated in the standard model via an RKA-secure PRF (e.g., that used in the Luby–Rackoff construction) as suggested in [4] or, outside the standard model, using random oracles.

Both transformations, however, come with two major drawbacks. The first drawback is the performance penalty. The standard-model approach incurs a total of six PRF computations in the 3-round network: 3 calls to generate the keys and another 3 to compute the PRP.[6] (The total number of calls is eight for the CCA case.) Note that the amortized complexity of the construction cannot be brought down back to 3 by storing the generated keys, as related-key attacks can be applied to these keys. In the ROM transform (on top of strong assumptions) the penalty will be smaller if the hash function is more efficient than the PRF. However, this leads to a second drawback: the transform is software/hardware intrusive, as extra circuitry for the implementation key-derivation procedure need to be added.

For these reasons, in the remainder of the paper, we will consider more efficient alternatives to obtaining RKA-secure PRPs by exploring directly the structure of Feistel constructions via simple key assignments. Before doing so, we prove a general theorem that allows us to move from the security analysis of a Feistel construction with respect to an RKA-secure PRF to a setting in which the round functions are instantiated by the ideal keyed function. Our result holds for any number of rounds and any key assignment.

Theorem 2 (Computational RKA Transition). *Let Φ be a family of RKD sets containing functions of the form $\mathsf{KSp}^m \longrightarrow \mathsf{KSp}^m$ and let $\kappa : \mathsf{KSp}^m \longrightarrow \mathsf{KSp}^n$ be a key assignment. Define $\Psi := \cup_i (\kappa \circ \Phi)_i$, where $(\kappa \circ \Phi)_i$ is the RKD set obtained by composing function in Φ by κ on the right and then projecting to i-th component for $1 \leq i \leq n$. Let ρ denote the ideal keyed function, and let* PRF *denote be a pseudorandom function. Then for any* PPT *adversary $\mathcal{A}$ against the* Φ-RKCCA *security of $\mathbf{F}^{\mathsf{PRF}}[\kappa]$, there is an adversary $\mathcal{B}$ against the* Ψ-RKCPA *security of* PRF *such that*

$$\mathbf{Adv}^{\mathsf{rkcca}}_{\mathbf{F}^{\mathsf{PRF}}[\kappa],\mathcal{A},\Phi}(\lambda) \leq \mathbf{Adv}^{\mathsf{rkcca}}_{\mathbf{F}^{\rho}[\kappa],\mathcal{A},\Phi}(\lambda) + \mathbf{Adv}^{\mathsf{rkcpa}}_{\mathsf{PRF},\mathcal{B},\Psi}(\lambda).$$

An analogous result holds for Φ-RKCPA *adversaries.*

Proof (Sketch). We start with the Φ-RKCCA game for $\mathbf{F}^{\mathsf{PRF}}[\kappa]$ and replace all n rounds function in the Feistel construction with an ideal keyed function. Any change in an adversary $\mathcal{A}$'s advantage in the two games can be used to break the (multi-key) Ψ-RKCCA security of PRF via an adversary $\mathcal{B}$. Algorithm $\mathcal{B}$ runs $\mathcal{A}$ and answers its forward queries to the Feistel construction as follows. On input (ϕ, x) where $\phi \in \Phi$, algorithm $\mathcal{B}$ sets $\psi_1 := (\kappa \circ \phi)|_1$ and calls the RKFn oracle on (ψ_1, x) to get x_1. It then sets $\psi_2 := (\kappa \circ \phi)|_2$, queries RKFn on (ψ_2, x_1) to get x_2. Algorithm $\mathcal{B}$ continues in this way for all n rounds and returns the final output. Backward queries can be also handled similarly using RKFn in the reverse direction. Clearly, according to the challenge bit b used in the Ψ-RKCPA game, $\mathcal{B}$ simulates the Φ-RKCCA game with the same challenge bit b for algorithm $\mathcal{A}$. □

[6] The overall tightness of security obtained via [4, Theorem 6.1] is also worse than what we obtain here, although it is possible that it can be improved via a direct analysis.

6 CPA Security: The 3-Round Constructions

As we discussed in the Introduction, no palindromic assignment of keys in a three-round Feistel construction can result in a CPA-secure PRP, since the construction in the forward direction can be used to compute inverses, and a trivial distinguishing attack emerges. Moreover, if the key used in the third round is independent of those used in first and second rounds, then the BK attack applies. Under these restriction, for simple key assignments and up to relabeling of the indices, we are left with only one 3-round construction which can potentially achieve CPA security under related-key attacks: $\mathbf{F}^{\mathsf{PRF}}[1,2,2]$.

The main proof of this section is an information-theoretic argument showing that $\mathbf{F}^{\rho}[1,2,2]$ is Φ-RKCPA secure for Φ's which are claw-free and switch-free. Combined with Theorem 2 in the previous section, this implies that $\mathbf{F}^{\mathsf{PRF}}[1,2,2]$ offers *intrinsic* RKA-resilience, in the sense that it permits leveraging the RKA-security properties of its underlying PRF.

For the security proof in this and the next sections we need to rely on an additional restriction on RKD sets.

Switch-freeness. A family of RKD sets Φ with arity $n > 2$ is called switch-free (SF) if the advantage of any PPT adversary $\mathcal{A}$ as defined below is negligible as a function of λ.

$$\mathbf{Adv}^{\mathsf{sf}}_{\mathcal{A},\Phi}(\lambda) := \Pr\left[(\exists \phi_1, \phi_2 \in \mathsf{L})(\exists i \neq j \in [n])\ \ \phi_1(\mathbf{K})|_i = \phi_2(\mathbf{K})|_j : \mathbf{K} \leftarrow\!\!\$\ \mathsf{KSp}^n; \mathsf{L} \leftarrow\!\!\$\ \mathcal{A}(1^\lambda)\right]$$

We note that the switch-free and claw-free properties are in general incomparable. Consider, for example, the set consisting of *id* and a function which agrees with *id* on all but one point. This set is switch-free but not claw-free. Conversely, consider the set consisting of *id* and the map $(K_1, K_2) \mapsto (K_2, K_1)$. This set is claw-free but not switch-free.

Theorem 3 ($\mathbf{F}^{\rho}[1,2,2]$ Security). *Let Φ be a family of RKD sets. The $\mathbf{F}^{\rho}[1,2,2]$ construction is Φ-RKCPA secure in the ideal keyed function model if Φ is claw-fee and switch-free. More precisely, for every Φ-RKCPA adversary $\mathcal{A}$ placing at most $Q(\lambda)$ queries to* RKFn*, there exist adversaries $\mathcal{B}_1$ and $\mathcal{B}_2$ such that*

$$\mathbf{Adv}^{\mathsf{rkcpa}}_{\mathbf{F}^{\rho}[1,2,2],\mathcal{A},\Phi}(\lambda) \leq \mathbf{Adv}^{\mathsf{rf/rp}}_{\mathcal{A},\Phi}(\lambda) + 2\mathbf{Adv}^{\mathsf{sf}}_{\mathcal{B}_1,\Phi}(\lambda) + 4\mathbf{Adv}^{\mathsf{cf}}_{\mathcal{B}_2,\Phi}(\lambda) + \frac{2^5 Q(\lambda)^2}{|\mathsf{Dom}_\lambda|}\,.$$

Proof (Intuition). We give a high-level description of the proof and refer the reader to the full version for the full details. We assume, without loss of generality, that the adversary is non-repeating in the sense that it not place redundant repeat queries to its oracle. We start with the Φ-RKCPA game, and consider an modified game where the round functions are implemented as follows. The first round is implemented using a consistent ideal keyed function (as in the original construction). The second and third round functions, however, will be forgetful and return independent random values on each invocation irrespective of the input values.

Note that the outputs of the network computed according to this game are random, and, by an appropriate strengthening of the classical PRP/PRF switching lemma (given in the full version), they are also indistinguishable from an ideal keyed permutation. Furthermore, in this game the values of the outputs of the first round function remain hidden as they are masked by random values generated in the third round.

Now the game above differs from the original CPA game due to inconsistencies occurring in computing round function values both across and within the same round, when the adversary is able to cause collisions in round function inputs in the original CPA game that are ignored in the game above. There are five such pairs of inconsistencies possible (we keep track of queries to the first round, so inconsistencies wont happen here). If there is a collision in inputs, which include the keys, to the first and second or first and third rounds, then the keys collide and this event leads to a violation of switch-freeness. Now suppose the inconsistency is due to a collision between the inputs to the third round function. Since the outputs of the second round function are randomly chosen at each invocation, this event happens with probability roughly $Q(\lambda)^2/|\mathsf{Dom}_\lambda|$ by the birthday bound. Collisions between the inputs to the second and third rounds also happen with negligible probably as the outputs of the first round remain hidden from the adversary. Finally, we are left with collisions in the inputs to the second round function. Note that this means that the keys input to this function are identical. Now if the keys or right halves of the inputs used in the first round in the two colliding queries were different, then the outputs of the first round function would be random and independent, and a collision would happen with a negligible probability (as first-round outputs are hidden). If the keys and right halves were identical, a collision can only take place if the left halves are also identical. However, due to the non-repeating condition, in this case we must have that the queried RKD functions are distinct, and consequently a claw in the RKD set is discovered. □

We emphasize that we do not claim the switch-free and claw-free restrictions are *necessary* for non-existence of attacks. On the other hand, these restrictions are akin to those adopted in previous works on RKA security, and do not overly constrain the practical applicability of our results. For example, the n-ary RKD sets for XOR-ing with constants defined by

$$\Phi_m^{\oplus} := \{\phi_{C_1,\ldots,C_m} : (K_1,\ldots,K_m) \mapsto (K_1 \oplus C_1,\ldots,K_m \oplus C_m) \ : \\ (C_1,\ldots,C_m) \in \mathsf{KSp}^m\}$$

can be easily shown to satisfy these restrictions. Unpredictability follows from the fact that each map in the set induces a permutation over the keys (and hence output distribution is uniform). For claw-freeness suppose we are given two distinct RKD functions. Suppose they differ in their i-th component, i.e., $C_i \neq C_i'$. Then, since the keys K_i and K_j are chosen independently and uniformly at random, the probability that the i-th output keys match, i.e., that $K_i \oplus C_i = K_j \oplus C_i'$, is negligible. Switch-freeness follows from a similar argument.

Note finally that the restrictions needed for the reduction to the RKA security of the underlying PRF are easily shown to be satisfied by the above set, as the key assignment is simple. We obtain the following corollary.

Corollary 1. $\mathbf{F}^{\mathsf{PRF}}[1,2,2]$ *is a* $\Phi_2^{\oplus}$-RKCPA-*secure pseudorandom permutation, if* PRF *is a* $\Phi_1^{\oplus}$-RKCPA-*secure PRF.*

In the full version we characterize the RKA security of the original three-round Luby–Rackoff construction, where three independent round keys are used.

7 CCA Security: The 4-Round Constructions

It is well known that the $\mathbf{F}^{\rho}[1,2,3]$ construction is CCA insecure. For example, the attacker can proceed as follows: 1) Choose arbitrary L, R, L', query $\mathrm{RKF}_N(L,R)$ to obtain C_1 and query $\mathrm{RKF}_N(L',R)$ to obtain C_2; 2) Query $\mathrm{RKF}_N^{-1}(C_2 \oplus (0, L \oplus L'))$ to obtain C_3; 3) Check if $(C_1 \oplus C_2 \oplus \mathsf{Swap}(C_3))$ is same as R. The same attack applies to all Feistel networks with three rounds, independently of the key assignment, and so there is no hope that such constructions can achieve any form of CCA security.

In this section we investigate the CCA security of 4-round constructions under related-key attacks. Due to the generic related-key attacks that we listed in the previous section (insecurity of palindromic key assignment and tampering with the last key), and the fact the in the CCA model the construction can be accessed in both the forward and backward directions, the only candidates than can potentially satisfy RKCCA security are: $\mathbf{F}^{\rho}[1,1,2,1]$, its inverse $\mathbf{F}^{\rho}[1,2,1,1]$, $\mathbf{F}^{\rho}[1,1,2,2]$, $\mathbf{F}^{\rho}[1,2,1,2]$, and $\mathbf{F}^{\rho}[1,2,3,1]$. In this work, we look at $\mathbf{F}^{\rho}[1,2,1,2]$.

The proof of RKCCA security for the $\mathbf{F}[1,2,1,2]$ construction, as in the RKCPA case, has two components: a computational part allowing transition from PRFs to ideal keyed functions, and an information-theoretic argument that establishes security when the construction is instantiated with an ideal keyed function. The first part of the proof follows from Theorem 2. We now prove the second part.

Theorem 4 ($\mathbf{F}[1,2,1,2]$ Security). *Let* Φ *be a family of RKD sets. Suppose* Φ *is claw-fee and switch-free. Then the* $\mathbf{F}^{\rho}[1,2,1,2]$ *construction is* Φ-RKCPA *secure in the ideal keyed function model. More precisely, for every* Φ-RKCCA *adversary* $\mathcal{A}$ *placing at most* $Q(\lambda)$ *queries to* RKF_N *or* RKF_N^{-1}*, there are* $\mathcal{B}_1$ *and* $\mathcal{B}_2$ *such that*

$$\mathbf{Adv}^{\mathsf{rkcca}}_{\mathbf{F}^{\rho}[1,2,1,2],\mathcal{A},\Phi}(\lambda) \leq \mathbf{Adv}^{\mathsf{rf/rp}}_{\mathcal{A},\Phi}(\lambda) + 2\mathbf{Adv}^{\mathsf{sf}}_{\mathcal{B}_1,\Phi}(\lambda) + 8\mathbf{Adv}^{\mathsf{cf}}_{\mathcal{B}_2,\Phi}(\lambda) + \frac{2^8 Q(\lambda)^2}{|\mathsf{Dom}_\lambda|}.$$

Proof (Intuition). We give a high-level description of the proof and refer the reader to the full version for the full details. The proof follows the same structure as Theorem 3, but it is slightly more complex due to the possibility of collisions occurring in the inputs of the round functions when they are used in the RKF$_N$

and RKF_N^{-1} oracles. We assume, without loss of generality, that the adversary is non-repeating in the sense that it does not place repeat queries to either of its oracles, does not decipher an enciphered value, and does not enciphered a deciphered value.

We start with the Φ-RKCCA game where the round functions faithfully implement an ideal keyed function. We then consider a game where all round functions are implemented in a *forgetful* way except that (1) the input round function in RKF_N is consistent and also keeps track of the entries contributed from RKF_N^{-1}'s output round; and (2) the input round function in RKF_N^{-1} is consistent and also keeps track of the entries contributed from RKF_N's output round. In this game the output values of the construction are random and hence indistinguishable from those from an ideal keyed permutation by the PRP/PRF switching lemma. Furthermore, the outputs of the input round functions in the RKF_N and RKF_N^{-1} oracles remain hidden as they are masked by the forgetful action of the remaining round functions.

As in the CPA setting, we need to keep track of collisions in the inputs to various pairs of round functions with lead to inconsistencies, as follows. (1) First forward and fourth backward rounds are consistent with previous queries due to their implementation. (2) Collisions between even and odd numbered round functions in both directions happen with negligible probability due to switch-freeness. (3) Inputs to the third and fourth forward rounds collide with negligible probability with the previous inputs of all other round functions due to the randomness of their respective inputs. A similar argument applies to the first and second backward rounds. (4) Collisions between first forward and third forward/backward rounds happen with negligible probability as the outputs of the fourth backward round are random and remain hidden from the adversary. A similar argument applies to the fourth/second rounds in the backward direction. (5) Collisions between second forward and fourth forward/backward rounds happen with negligible probability as outputs of the first forward round are random and remain hidden. A similar argument applies to the second round in the backward direction. (6) Finally, collisions between the second forward round and itself or second backward can be bounded using the fact that outputs of the first forward round are random remain hidden, combined with claw-freeness, similarly to the CPA case. A similar argument applies to the third backward round. □

As in the CPA setting, the family $\Phi_4^{\oplus}$ satisfies all the prerequisites required for the reduction to the RKA security of the underlying PRF and we obtain the following corollary.

Corollary 2. $\mathbf{F}^{\mathsf{PRF}}[1,2,1,2]$ *is a* $\Phi_2^{\oplus}$-RKCCA-*secure PRP, if the underlying* PRF *is a* $\Phi_1^{\oplus}$-RKCCA-*secure PRF.*

In the full version we give a positive result for the RKA security of the original 4-round Luby–Rackoff construction.

8 Directions for Further Research

This works takes a first step in the construction of RKA-secure symmetric cryptosystems based on Feistel networks, and leaves open a number of directions for future research. From a conceptual point of view, the RKA-security of many-round Feistel networks (including beyond-birthday-type concrete security) are important open questions. From a practical point of view, the RKA security of alternative constructions of PRPs such as generalized Feistel networks [23] and key-alternating ciphers [13], along with their potential (dis)advantages over Feistel networks are another interesting direction for future work.

We conclude the paper with a conjecture about the RKA security of Feistel networks with respect to arbitrary numbers of rounds and key assignments, which generalizes the CCA characterization studied in [34], and generalizes our result in Sect. 7 to the other plausible key assignments.

Conjecture. Let $n > 3$ be an integer, $\kappa : \mathsf{KSp}^m \longrightarrow \mathsf{KSp}^n$ be a simple key assignment, and Φ be a family of RKD sets consisting of functions $\phi : \mathsf{KSp}^m \longrightarrow \mathsf{KSp}^m$. Suppose that the following requirements are satisfied.

1. $\kappa \circ \Phi$ is output unpredictable and claw-free.
2. (κ, Φ) is palindrome-fee: for any $\phi, \phi' \in \Phi$ the probability over a random $(K_1, \ldots, K_m)$ that $\kappa \circ \phi'(K_1, \ldots, K_m) = \sigma \circ \kappa \circ \phi(K_1, \ldots, K_m)$ is negligible, where $\sigma(K_1, \ldots, K_m) := (K_m, \ldots, K_1)$.
3. (κ, Φ) is first-key repeating: for any distinct $\phi, \phi' \in \Phi$ the probability over a random $(K_1, \ldots, K_m)$ that $[\kappa \circ \phi(K_1, \ldots, K_m)]_1 \neq [\kappa \circ \phi'(K_1, \ldots, K_m)]_1$ and $[\kappa \circ \phi(K_1, \ldots, K_m)]_i = [\kappa \circ \phi'(K_1, \ldots, K_m)]_i$ for all $1 < i \leq n$ is small.
4. (κ, Φ) is last-key repeating: for any distinct $\phi, \phi' \in \Phi$ the probability over a random $(K_1, \ldots, K_m)$ that $[\kappa \circ \phi(K_1, \ldots, K_m)]_n \neq [\kappa \circ \phi'(K_1, \ldots, K_m)]_n$ and $[\kappa \circ \phi(K_1, \ldots, K_m)]_i = [\kappa \circ \phi'(K_1, \ldots, K_m)]_i$ for all $1 \leq i < n$ is small.

Then the $\mathbf{F}^{\rho}[\kappa]$ construction is Φ-RKCCA secure in the ideal keyed function model and hence, combined with Theorem 2, the $\mathbf{F}^{\mathsf{PRF}}[\kappa]$ construction is Φ-RKCCA secure for a Ψ-RKCPA-secure PRF, for Ψ as in the statement of Theorem 2.

We note that among the above restrictions claw-freeness is the only requirement which is not known to be necessary. Hence we obtain an "almost" characterization. Note, however, that the RKA security of a deterministic cryptosystems seems difficult to be established without assuming claw-freeness (nevertheless, cf. [5] for a weaker ICR notion). The conjecture strengthens and extends some of the results presented in the previous sections.

Acknowledgements. Manuel Barbosa was supported by Project Best Case, co-financed by the North Portugal Regional Operational Programme (ON.2 – O Novo Norte), under the National Strategic Reference Framework (NSRF), through the European Regional Development Fund (ERDF). Pooya Farshim was supported by grant Fi 940/4-1 of the German Research Foundation (DFG).

References

1. Albrecht, M.R., Farshim, P., Paterson, K.G., Watson, G.J.: On cipher-dependent related-key attacks in the ideal-cipher model. In: Joux, A. (ed.) FSE 2011. LNCS, vol. 6733, pp. 128–145. Springer, Heidelberg (2011)
2. Anderson, R., Kuhn, M.: Low cost attacks on tamper resistant devices. In: Christianson, B., Lomas, M., Crispo, B., Roe, M. (eds.) Security Protocols 1997. LNCS, vol. 1361, pp. 125–136. Springer, Heidelberg (1998)
3. Barbosa, M., Farshim, P.: The Related-key analysis of feistel constructions. In: Cryptology ePrint Archive, Report 2014/093 (2014)
4. Bellare, M., Cash, D.: Pseudorandom functions and permutations provably secure against related-key attacks. In: Cryptology ePrint Archive, Report 2010/397 (2013)
5. Bellare, M., Cash, D., Miller, R.: Cryptography secure against related-key attacks and tampering. In: Lee, D.H., Wang, X. (eds.) ASIACRYPT 2011. LNCS, vol. 7073, pp. 486–503. Springer, Heidelberg (2011)
6. Bellare, M., Kohno, T.: A theoretical treatment of related-key attacks: RKA-PRPs, RKA-PRFs, and applications. In: Biham, E. (ed.) EUROCRYPT 2003. LNCS, vol. 2656, pp. 491–506. Springer, Heidelberg (2003)
7. Bellare, M., Paterson, K.G., Thomson, S.: RKA security beyond the linear barrier: IBE, encryption and signatures. In: Wang, X., Sako, K. (eds.) ASIACRYPT 2012. LNCS, vol. 7658, pp. 331–348. Springer, Heidelberg (2012)
8. Biham, E.: How to decrypt or even substitute DES-encrypted messages in 228 steps. Inf. Process. Lett. **84**(3), 117–124 (2002)
9. Biham, E.: New types of cryptoanalytic attacks using related keys. J. Cryptol. **7**(4), 229–246 (1994)
10. Biham, E., Dunkelman, O., Keller, N.: Related-key boomerang and rectangle attacks. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 507–525. Springer, Heidelberg (2005)
11. Biryukov, A., Khovratovich, D.: Related-key cryptanalysis of the full AES-192 and AES-256. In: Matsui, M. (ed.) ASIACRYPT 2009. LNCS, vol. 5912, pp. 1–18. Springer, Heidelberg (2009)
12. Biryukov, A., Khovratovich, D., Nikolić, I.: Distinguisher and related-key attack on the full AES-256. In: Halevi, S. (ed.) CRYPTO 2009. LNCS, vol. 5677, pp. 231–249. Springer, Heidelberg (2009)
13. Bogdanov, A., Knudsen, L.R., Leander, G., Standaert, F.-X., Steinberger, J., Tischhauser, E.: Key-alternating ciphers in a provable setting: encryption using a small number of public permutations. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT 2012. LNCS, vol. 7237, pp. 45–62. Springer, Heidelberg (2012)
14. Brier, E., Clavier, C., Olivier, F.: Correlation power analysis with a leakage model. In: Joye, M., Quisquater, J.-J. (eds.) CHES 2004. LNCS, vol. 3156, pp. 16–29. Springer, Heidelberg (2004)
15. Chari, S., Rao, J.R., Rohatgi, P.: Template attacks. In: Kaliski Jr., B.S., Koç, Ç.K., Paar, C. (eds.) Cryptographic Hardware and Embedded Systems - CHES 2002. LNCS, vol. 2523, pp. 13–28. Springer, Heidelberg (2002)
16. EMV integrated circuit card specifications for payment systems. Book 2 Security and Key Management, Version 4.2, June 2008
17. Feistel, H.: Cryptography and computer privacy. Sci. Am. **228**, 15–23 (1973)
18. Feistel, H., Notz, W.A., Lynn Smithm, J.: Some cryptographic techniques for machine-to-machine data communications. Proc. of the IEEE **63**(11), 1545–1554 (1975)

19. Goldenberg, D., Liskov, M.: On related-secret pseudorandomness. In: Micciancio, D. (ed.) TCC 2010. LNCS, vol. 5978, pp. 255–272. Springer, Heidelberg (2010)
20. Goldreich, O., Levin, L.: A hard-core predicate for all one-way functions. In: Vitter, J.S. (ed.) STOC, pp. 25–32. ACM (1989)
21. Goldreich, O., Goldwasser, S., Micali, S.: How to construct random functions. J. ACM **33**(4), 792–807 (1986)
22. Goyal, V., O'Neill, A., Rao, V.: Correlated-input secure hash functions. In: Ishai, Y. (ed.) TCC 2011. LNCS, vol. 6597, pp. 182–200. Springer, Heidelberg (2011)
23. Hoang, V.T., Rogaway, P.: On generalized feistel networks. In: Rabin, T. (ed.) CRYPTO 2010. LNCS, vol. 6223, pp. 613–630. Springer, Heidelberg (2010)
24. Holenstein, T., Künzler, R., Tessaro, S.: The equivalence of the random oracle model and the ideal cipher model, revisited. In Fortnow, L., Vadhan, S.P. (eds.) STOC 2011, pp. 89–98. ACM (2011)
25. Iwata, T., Kohno, T.: New security proofs for the 3gpp confidentiality and integrity algorithms. In: Roy, B., Meier, W. (eds.) FSE 2004. LNCS, vol. 3017, pp. 427–445. Springer, Heidelberg (2004)
26. Khovratovich, D., Nikolić, I., Rechberger, C.: Rotational rebound attacks on reduced skein. In: Abe, M. (ed.) ASIACRYPT 2010. LNCS, vol. 6477, pp. 1–19. Springer, Heidelberg (2010)
27. Knudsen, L.R.: Cryptanalysis of LOKI91. In: Seberry, J., Zheng, Y. (eds.) Advances in Cryptology – AUSCRYPT '92. LNCS, vol. 718, pp. 196–208. Springer, Heidelberg (1993)
28. Knudsen, L.R., Kohno, T.: Analysis of RMAC. In: Johansson, T. (ed.) FSE 2003. LNCS, vol. 2887, pp. 182–191. Springer, Heidelberg (2003)
29. Liskov, M., Rivest, R.L., Wagner, D.: Tweakable block ciphers. In: Yung, M. (ed.) CRYPTO 2002. LNCS, vol. 2442, p. 31. Springer, Heidelberg (2002)
30. Luby, M., Rackoff, C.: How to construct pseudo-random permutations from pseudo-random functions. SIAM J. Comput. **17**(2), 373–386 (1988)
31. Lucks, S.: Ciphers secure against related-key attacks. In: Roy, B., Meier, W. (eds.) FSE 2004. LNCS, vol. 3017, pp. 359–370. Springer, Heidelberg (2004)
32. Kocher, P.C., Jaffe, J., Jun, B.: Differential power analysis. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, p. 388. Springer, Heidelberg (1999)
33. Maurer, U., Pietrzak, K.: The security of many-round luby-rackoff pseudo-random permutations. In: Biham, Eli (ed.) EUROCRYPT 2003. LNCS, vol. 2656, pp. 544–561. Springer, Heidelberg (2003)
34. Nandi, M.: The characterization of luby-rackoff and its optimum single-key variants. In: Gong, G., Gupta, K.C. (eds.) INDOCRYPT 2010. LNCS, vol. 6498, pp. 82–97. Springer, Heidelberg (2010)
35. Naor, M., Reingold, O.: On the construction of pseudorandom permutations: Luby-Rackoff revisited. J. Cryptol. **12**(1), 29–66 (1999)
36. Patarin, J.: How to construct pseudorandom permutations and super pseudorandom permutations from one single pseudorandom functions. In: Rueppel, R.A. (ed.) Advances in Cryptology – EUROCRYPT 1992. LNCS, vol. 658, pp. 256–266. Springer, Heidelberg (1992)
37. Patarin, J.: Security of random feistel schemes with 5 or more rounds. In: Franklin, M. (ed.) CRYPTO 2004. LNCS, vol. 3152, pp. 106–122. Springer, Heidelberg (2004)
38. Piret, G.: Block Ciphers: Security Proofs, Cryptanalysis, Design, and Fault Attacks. Ph.D. Thesis, Université Catholique de Louvain (2005)
39. Sarkar, S., Maitra, S.: Side channel attack to actual cryptanalysis: breaking crt-rsa with low weight decryption exponents. In: Prouff, E., Schaumont, P. (eds.) CHES 2012. LNCS, vol. 7428, pp. 476–493. Springer, Heidelberg (2012)

40. Shoup, V.: Sequences of games: a tool for taming complexity in security proofs. In: Cryptology ePrint Archive, Report 2004/332 (2004)
41. Vaudenay, S.: Feistel ciphers with L_2-decorrelation. In: Tavares, S., Meijer, H. (eds.) SAC 1998. LNCS, vol. 1556, p. 1. Springer, Heidelberg (1999)
42. Wee, H.: Public key encryption against related key attacks. In: Fischlin, M., Buchmann, J., Manulis, M. (eds.) PKC 2012. LNCS, vol. 7293, pp. 262–279. Springer, Heidelberg (2012)

The Indistinguishability of the XOR of k Permutations

Benoit Cogliati(✉), Rodolphe Lampe, and Jacques Patarin

University of Versailles, Paris, France
benoit.cogliati@ens.uvsq.fr

Abstract. Given k independent pseudorandom permutations $f_1, \ldots, f_k$ over $\{0,1\}^n$, it is natural to define a pseudorandom function by XORing the permutations: $f_1 \oplus \ldots \oplus f_k$. In [9] Stefan Lucks studied the security of this PRF. In this paper we improve the security bounds of [9] by using different proof techniques.

Keywords: Pseudorandom functions · Pseudorandom permutations · Security beyond the birthday bound · Luby-Rackoff backwards

1 Introduction

Much research dealt with constructing cryptographic operations from other ones: Levin [6] got "pseudorandom bit generators" from "one-way functions", then Goldreich, Goldwasser and Micali [4] constructed pseudorandom functions (PRFs) from "pseudorandom bit generators". In [1], Aiello and Venkatesan studied how to construct PRFs from smaller PRFs. Luby and Rackoff [7] dealt with the problem of getting pseudorandom permutations (PRPs) from PRFs; further work about their construction can be found in [8,11]. Our article focuses on the reverse problem of converting PRPs into PRFs named "Luby-Rackoff backwards" which was first considered in [3]. This problem is obvious if we are interested in an asymptotical polynomial versus non polynomial security model (since a PRP is then a PRF), but not if we are interested in achieving more optimal and concrete security bounds. More precisely, the loss of security when regarding a PRP as a PRF comes from the "birthday attack" which can distinguish a random permutation from a random function of n bits to n bits in $2^{\frac{n}{2}}$ operations and $2^{\frac{n}{2}}$ queries. Therefore different ways to build PRF from PRP with a security above $2^{\frac{n}{2}}$ and by performing very few computations have been suggested (see [2,3,5,9]). One of the simplest way is to XOR k independent pseudorandom permutations with $k \geq 2$. In [9] (Theorem 2, p.474) Stefan Lucks proved, with a simple proof, that the XOR of k independant PRPs gives a PRF with security at least in $\mathcal{O}\left(2^{\frac{k}{k+1}} n\right)$. In [2,12] difficult analyses of $k = 2$ are given, with proofs that the security is good when the number of queries is lower than $\mathcal{O}\left(\frac{2^n}{n^{2/3}}\right)$ or $\mathcal{O}\left(2^n\right)$. For $k \geq 3$ there is a significant gap between the proven security of [9] and the best attacks of [13].

C. Cid and C. Rechberger (Eds.): FSE 2014, LNCS 8540, pp. 285–302, 2015.
DOI: 10.1007/978-3-662-46706-0_15

In this paper we reduce this gap by improving the proven security for the XOR of k permutations, $k \geq 3$. Constructions with $k \geq 3$ instead of $k = 2$ are interesting for various reasons. First, our proofs are much simpler than the proofs of [2,12]. Second, in many cryptographic applications the size n of the blocks cannot be chosen by the designer of the algorithm since it is imposed by the application. Then it is interesting to have another parameter to decrease the proven advantage of any adversary to a value as small as wanted with a simple construction. Our proof technique is based on the "coefficient H technique" of Patarin (cf [14]). However we only use the first steps (and not all the refinements) in order to keep very simple proofs with still better security results than previously known; we could achieve tighter bounds by using the full technique, but it would require more computations (such as [15]).

Related Problems. In [10] the security of the XOR of two **public** permutations are studied (i.e. indifferentiability instead of indistinguishability).

Organisation of the Paper. Section. 2 presents the notations and basic definitions that are used in this paper. In Sects. 3 and 4, two security bounds are shown with different techniques (respectively the "H_σ coefficient" technique and the "H coefficient" technique). Then both these results are compared to the one from [9] in the last section.

2 Preliminaries

We denote I_n the set of $n-$bits strings and J_n^q the subset of I_n^q of values $(x_i)_{1 \leq i \leq q}$ satisfying $x_i \neq x_j, \forall i \neq j$. We denote F_n the set of functions from I_n to I_n and B_n the set of permutations of I_n. The notation $x \in_R E$ stands for "x is chosen randomly with a uniform distribution in E".

An adversary A trying to distinguish between $f_1 \oplus \ldots \oplus f_k$, where $f_i \in_R B_n$ for each $i \in \{1, \ldots, k\}$, from a random function $F \in_R F_n$ is considered to have access to an oracle Q. This oracle either simulates F or $f_1 \oplus \ldots \oplus f_k$. A chooses inputs $x \in \{0,1\}^n$; then Q responds $Q(x) \in \{0,1\}^n$. After at most q queries, A outputs $A(Q) \in \{0,1\}$. $A(Q)$ is then seen as a random variable over $\{0,1\}$. This is an adaptative chosen plaintext attack (cpa). To measure the pseudo-randomness of the XOR of k permutations one must evaluate the advantage $\mathbf{Adv}^{\mathrm{cpa}}_{A, f_1 \oplus \ldots \oplus f_k}$ of an adversary A which is defined as

$$\mathbf{Adv}^{\mathrm{cpa}}_{A, f_1 \oplus \ldots \oplus f_k} = |Pr[A(f_1 \oplus \ldots \oplus f_k) = 1] - Pr[A(F) = 1]|.$$

We write $\mathbf{Adv}^{\mathrm{cpa}}_{f_1 \oplus \ldots \oplus f_k}$ for the maximal advantage any adversary can get when trying to distinguish the XOR of k random permutations from a random function.

3 Security Bound from the H_σ Technique

3.1 Linking the Advantage to a Combinatorial Problem

Let $k \geq 2$. We use Theorem 3 from [14]:

Theorem 1. *Let $\alpha, \beta \in \mathbb{R}^+$ and $q \in \mathbb{N} \setminus \{0\}$. Let E be a subset of I_n^q such that $|E| \geq (1-\beta)2^{nq}$. Suppose that, for each sequence $(a_i)_{1\leq i\leq q}, (b_i)_{1\leq i\leq q} \in J_n^q$, with $(b_i)_{1\leq i\leq q} \in E$:*

$$H(a,b) \geq (1-\alpha)\frac{|B_n|^k}{2^{nq}},$$

with $H(a,b)$ the number of $(f_1,\ldots,f_k) \in B_n^k$ such that:

$$\forall i, 1 \leq i \leq q, (f_1 \oplus \ldots \oplus f_k)(a_i) = b_i.$$

Then:

$$\mathbf{Adv}^{\mathrm{cpa}}_{f_1\oplus\ldots\oplus f_k} \leq \alpha + \beta.$$

For every $b \in J_n^q$, let $h_q(b)$ be the number of sequences $x^1, x^2, \ldots, x^{k-1} \in J_n^q$ such that $x^1 \oplus \ldots \oplus x^{k-1} \oplus b \in J_n^q$ then

Lemma 1. *For all $a, b \in J_n^q$:*

$$H(a,b) = h_q(b)\frac{|B_n|^k}{\left(2^n \times \cdots \times (2^n - q + 1)\right)^k}.$$

Proof. The number $H(a,b)$ can be seen as the sum, over the sequences $x^1, x^2, \ldots, x^{k-1} \in J_n^q$ such that $x^1 \oplus \ldots \oplus x^{k-1} \oplus b \in J_n^q$, of the number of $f_1, \ldots, f_k \in B_n$ satisfying the equations $f_j(a_i) = x_i^j$ for all $j \leq k-1, i \leq q$ and $f_k(a_i) = x_i^1 \oplus \ldots \oplus x_i^{k-1} \oplus b_i, \forall i \leq q$. Then, for each choices of $x^1, \ldots, x^{k-1}$, each f_j is a uniformly random permutation fixed on q points so $H(a,b) = h_q(b)\left(\frac{|B_n|}{2^n\times\cdots\times(2^n-q+1)}\right)^k$, which also shows that $H(a,b)$ does not depend of a. □

We now see h_q as a random variable over $b \in_R I_n^q$. The security of the XOR of k permutations is closely related to the variance and the expectancy of this random variable:

Lemma 2. *The advantage satisfies:*

$$\mathbf{Adv}^{\mathrm{cpa}}_{f_1\oplus\ldots\oplus f_k} \leq 2\left(\frac{\mathbb{V}[h_q]}{\mathbb{E}[h_q]^2}\right)^{1/3}. \tag{1}$$

Proof. For all a, we define $H(a)$ the random variable over b equal to $H(a,b)$. The Bienayme-Chebyshev's inequality yields:

$$\forall \epsilon > 0, \Pr\left[|H(a) - \mathbb{E}[H(a)]| \leq \epsilon\right] \geq 1 - \frac{\mathbb{V}[H(a)]}{\epsilon^2}.$$

Taking $\epsilon = \alpha\mathbb{E}[H(a)]$:

$$\forall \alpha > 0, \Pr\left[|H(a) - \mathbb{E}[H(a)]| \leq \alpha\mathbb{E}[H(a)]\right] \geq 1 - \frac{\mathbb{V}[H(a)]}{\alpha^2\mathbb{E}[H(a)]^2}.$$

Then

$$\forall \alpha > 0, \Pr\left[H(a) \geq (1-\alpha)\mathbb{E}\left[H(a)\right]\right] \geq 1 - \frac{\mathbb{V}\left[H(a)\right]}{\alpha^2 \mathbb{E}\left[H(a)\right]^2}.$$

Thus, defining $E = \{(b_i)_{1 \leq i \leq q} | H(a,b) \geq (1-\alpha)\mathbb{E}\left[H(a)\right]\}$, Theorem 1 yields:

$$\forall \alpha > 0, \mathbf{Adv}^{\mathrm{cpa}}_{f_1 \oplus \ldots \oplus f_k} \leq \alpha + \frac{\mathbb{V}\left[H(a)\right]}{\alpha^2 \mathbb{E}\left[H(a)\right]^2}.$$

Then, with $\alpha = \left(\frac{\mathbb{V}[H(a)]}{\mathbb{E}[H(a)]^2}\right)^{1/3}$:

$$\mathbf{Adv}^{\mathrm{cpa}}_{f_1 \oplus \ldots \oplus f_k} \leq 2\left(\frac{\mathbb{V}\left[H(a)\right]}{\mathbb{E}\left[H(a)\right]^2}\right)^{1/3} = 2\left(\frac{\mathbb{V}\left[h_q\right]}{\mathbb{E}\left[h_q\right]^2}\right)^{1/3}.$$

□

Lemma 3. *The mean of h_q satisfies:*

$$\mathbb{E}\left[h_q\right] = \frac{\left[2^n(2^n-1)\ldots(2^n-q+1)\right]^k}{2^{nq}}.$$

Proof. This result generalizes a theorem found in [12]. We define δ_x, with $x = (x^1, \ldots, x^{k-1}) \in (J_n^q)^{k-1}$, a random variable over b such that $\delta_x = 1$ if $x^1, \ldots, x^{k-1}, b \oplus x^1 \oplus \cdots \oplus x^{k-1} \in J_n^q$ and $\delta_x = 0$ otherwise. It's clear that $h_q = \sum_{x \in (J_n^q)^{k-1}} \delta_x$, then

$$\begin{aligned}
\mathbb{E}\left[h_q\right] &= \sum_{x \in (J_n^q)^{k-1}} \mathbb{E}\left[\delta_x\right] \\
&= \sum_{x \in (J_n^q)^{k-1}} \Pr\left[\text{the } b_i \oplus x_i^1 \oplus \ldots \oplus x_i^{k-1} \text{ are pairwise distinct}\right] \\
&= \sum_{x \in (J_n^q)^{k-1}} \frac{2^n(2^n-1)\ldots(2^n-q+1)}{2^{nq}} \\
&= |J_n^q|^{k-1} \times \frac{2^n(2^n-1)\ldots(2^n-q+1)}{2^{nq}} \\
&= \frac{\left[2^n(2^n-1)\ldots(2^n-q+1)\right]^k}{2^{nq}}.
\end{aligned}$$

□

We now focus on the variance of h_q.

3.2 Study of $\mathbb{V}\left[h_q\right]$

We denote λ_q the number of sequences $g^1, \ldots, g^{2k} \in J_n^q$ such that $g^1 \oplus \cdots \oplus g^{2k} = 0$. These conditions will be referred to as the λ_q conditions. This is $2k$ sequences of q

pairwise distinct elements and q equations so, we could expect λ_q to be close to

$$U_q := \frac{(2^n(2^n-1)(2^n-q+1))^{2k}}{2^{2nq}}.$$

We see in the next lemma that the problem of knowing how close λ_q is from U_q is at the core of the computation of the advantage.

Lemma 4. *The advantage satisfies:*

$$\mathbf{Adv}^{\mathrm{cpa}}_{f_1\oplus\ldots\oplus f_k} \leq 2\left(\frac{\lambda_q}{U_q}-1\right)^{1/3}.$$

Proof. We know that $h_q = \sum_x \delta_x$ with the sum being over $x \in (J_n^q)^{k-1}$, so the linearity of the expected value operator yields:

$$\begin{aligned}
\mathbb{V}[h_q] &= \mathbb{E}\left[\left(\sum_x \delta_x - \mathbb{E}[h_q]\right)^2\right] \\
&= \mathbb{E}\left[\left(\sum_x \delta_x\right)^2 - 2\left(\sum_x \delta_x\right)\mathbb{E}[h_q] + \mathbb{E}[h_q]^2\right] \\
&= \mathbb{E}\left[\left(\sum_x \delta_x\right)\left(\sum_{x'} \delta_{x'}\right)\right] - 2\mathbb{E}\left[\sum_x \delta_x\right]\mathbb{E}[h_q] + \mathbb{E}[h_q]^2 \\
&= \mathbb{E}\left[\sum_{x,x'} \delta_x\delta_{x'}\right] - \mathbb{E}[h_q]^2,
\end{aligned}$$

the sum being over $x, x' \in (J_n^q)^{k-1}$. Then:

$$\mathbb{E}\left[\sum_{x,x'} \delta_x\delta_{x'}\right] = \frac{1}{2^{nq}}\sum_{b,x,x'} \delta_x(b)\delta_{x'}(b).$$

We know that $\delta_x(b)\delta_{x'}(b)$, with $x, x' \in (J_n^q)^{k-1}$, equals 1 if and only if $b\oplus x^1\oplus\cdots\oplus x^{k-1} \in J_n^q$ and $b\oplus x'^1\oplus\cdots\oplus x'^{k-1} \in J_n^q$. If we change variables like this: $g^i := x^i$ and $g^{i+k-1} := x'^i$ for all $1 \leq i \leq k-1$ and $g^{2k-1} := b \oplus x^1 \oplus \cdots \oplus x^{k-1}, g^{2k} := b \oplus x'^1 \oplus \cdots \oplus x'^{k-1}$, we see that $\sum_{b,x,x'} \delta_x(b)\delta_{x'}(b)$ is equal to λ_q. Then:

$$\begin{aligned}
\mathbb{V}[h_q] &= \frac{\lambda_q}{2^{nq}} - \mathbb{E}[h_q]^2 \\
&= \frac{\lambda_q - U_q}{2^{nq}} \text{ since } \mathbb{E}[h_q]^2 = \frac{U_q}{2^{nq}}.
\end{aligned}$$

Moreover, using Lemma 2:

$$\begin{aligned}\mathbf{Adv}^{\mathrm{cpa}}_{f_1\oplus\ldots\oplus f_k} &\leq 2\left(\frac{\mathbb{V}\left[h_q\right]}{\mathbb{E}\left[h_q\right]^2}\right)^{1/3}\\ &\leq 2\left(\frac{\lambda_q - U_q}{U_q}\right)^{1/3}\\ &\leq 2\left(\frac{\lambda_q}{U_q} - 1\right)^{1/3}.\end{aligned}$$

□

The strategy we follow is to evaluate recursively, more and more accurately, the coefficients λ_α for $1 \leq \alpha \leq q$.

3.3 First Evaluation of λ_α

By definition, $\lambda_{\alpha+1}$ is the number of tuples $g^1, \ldots, g^{2k} \in J_n^{\alpha+1}$ such that:

1. the λ_α conditions hold,
2. for all $1 \leq j \leq 2k$, $g^j_{\alpha+1} \notin \{g^j_i, 1 \leq i \leq \alpha\}$,
3. $g^1_{\alpha+1} \oplus \cdots \oplus g^{2k}_{\alpha+1} = 0$. $\quad (E_{\alpha+1})$

Hence there are $2k\alpha$ equations that should not be verified. For $1 \leq i \leq 2k\alpha$, we denote β_i the i-th such equation. Let B_i be the set of tuples $(g^1, \ldots, g^{2k})$ which satisfy the λ_α conditions, the equation $(E_{\alpha+1})$ and the equation β_i, for $1 \leq i \leq 2k\alpha$. Then:

$$\lambda_{\alpha+1} = 2^{(2k-1)n}\lambda_\alpha - \left|\bigcup_{i=1}^{2k\alpha} B_i\right|.$$

Using the inclusion-exclusion principle:

$$\lambda_{\alpha+1} = 2^{(2k-1)n}\lambda_\alpha + \sum_{l=1}^{2k\alpha}(-1)^l \sum_{i_1<\ldots<i_l} |B_{i_1} \cap \ldots \cap B_{i_l}|.$$

When more than $2k+1$ equations β_i are considered, at least two of them use the same variable, for example $g^1_{\alpha+1} = g^1_1$ and $g^1_{\alpha+1} = g^1_2$, which is impossible according to the λ_α conditions. Thus:

$$\lambda_{\alpha+1} = 2^{(2k-1)n}\lambda_\alpha + \sum_{l=1}^{2k}(-1)^l \sum_{i_1<\ldots<i_l} |B_{i_1} \cap \ldots \cap B_{i_l}|. \tag{2}$$

Now, we study every kind of intersection.

- 1 **equation:**

The β_i equation fixes the value of one new variable, whereas the others are free, so:

$$|B_i| = 2^{(2k-2)n}\lambda_\alpha$$

and there exists $2k\alpha$ such sets.

- l **equations** $(2 \leq l \leq 2k-1)$:

Such an intersection is non-empty if every equation β_i uses a different new variable. In this case, l new variables are fixed and the others remain free. Thus,

$$|B_{i_1} \cap \ldots \cap B_{i_l}| = 2^{(2k-1-l)n}\lambda_\alpha$$

and there are $\binom{2k}{\ell}\alpha^k$ such non-empty intersections.

- $2k$ **equations:**

Like before, such a set is non-empty if every equation β_i uses a different new variable. In this case, the set $B_{i_1} \cap \ldots \cap B_{i_{2k}}$ is composed of tuples such that $g^1_{\alpha+1} = g^1_{i_1}, \ldots, g^{2k}_{\alpha+1} = g^{2k}_{i_{2k}}$ and the equation $(E_{\alpha+1})$ implies that:

$$g^1_{i_1} \oplus \cdots \oplus g^{2k}_{i_{2k}} = 0.$$

We denote X this equation and $\lambda'_\alpha(X)$ the size of $|B_{i_1} \cap \ldots \cap B_{i_{2k}}|$. There are 3 possible cases:

- If the $2k$ indexes in X are equal then X is always true. There are α possibilities and $\lambda'_\alpha(X) = \lambda_\alpha$.
- If $2k-1$ indexes are equal and the last is different, then $\lambda'_\alpha(X) = 0$ since X is in contradiction with λ_α. There are $2k\alpha(\alpha-1)$ possibilities.
- We denote S the set of equations X that are not of the previous types. We denote $\lambda'_\alpha = \max_S \lambda'_\alpha(X)$.

Hence, thanks to (2), one has:

$$\begin{aligned}
\lambda_{\alpha+1} &= 2^{(2k-1)n}\lambda_\alpha - 2k\alpha\lambda_\alpha + \sum_{\ell=2}^{2k-1}\binom{2k}{\ell}(-1)^l\alpha^l 2^{(2k-1-\ell)n}\lambda_\alpha + \sum_X \lambda'_\alpha(X) \\
&= \left(2^{2kn} - 2k\alpha 2^n + \sum_{\ell=2}^{2k-1}\binom{2k}{\ell}(-1)^l\alpha^l 2^{(2k-\ell)n}\right)\frac{\lambda_\alpha}{2^n} + \alpha\lambda_\alpha + \sum_{X\in S}\lambda'_\alpha(X) \\
&\leq \frac{\left((2^n-\alpha)^{2k} - \alpha^{2k} + 2^n\alpha\right)\lambda_\alpha}{2^n} + \left(\alpha^{2k} - \alpha - 2k\alpha(\alpha-1)\right)\lambda'_\alpha
\end{aligned}$$

We denote $\epsilon_\alpha = \frac{2^n\lambda'_\alpha}{\lambda_\alpha} - 1$, so:

$$\begin{aligned}
\frac{2^n\lambda_{\alpha+1}}{\lambda_\alpha} &\leq (2^n-\alpha)^{2k} - \alpha^{2k} + 2^n\alpha + \frac{2^n\lambda'_\alpha}{\lambda_\alpha} \times (\alpha^{2k} - \alpha - 2k\alpha(\alpha-1)) \\
&\leq (2^n-\alpha)^{2k} + 2^n\alpha - \alpha - 2k\alpha(\alpha-1) + \epsilon_\alpha \times (\alpha^{2k} - \alpha - 2k\alpha(\alpha-1)) \\
&\leq (2^n-\alpha)^{2k} - 2k\alpha^2 + \alpha(2^n + 2k - 1) + \epsilon_\alpha \times (\alpha^{2k} - 2k\alpha^2 + \alpha(2k-1))
\end{aligned}$$

3.4 Relation Between the Advantage and ϵ_α

Lemma 5. *For every $m \geq 1$, the advantage satisfies:*

$$\mathbf{Adv}^{\mathrm{cpa}}_{f_1 \oplus \ldots \oplus f_k} \leq 2 \left(\prod_{\alpha=1}^{m-1} \left(1 + \frac{-2k\alpha^2 + \alpha(2^n + 2k - 1) + \epsilon_\alpha \times (\alpha^{2k} - 2k\alpha^2 + \alpha(2k-1))}{(2^n - \alpha)^{2k}} \right) - 1 \right)^{1/3}.$$

Proof. We know that

$$\frac{2^n U_{\alpha+1}}{U_\alpha} = (2^n - \alpha)^{2k},$$

and the result of the previous section yields:

$$\begin{aligned} \frac{\lambda_{\alpha+1}}{U_{\alpha+1}} &\leq \frac{\lambda_\alpha}{U_\alpha} \left(\frac{(2^n - \alpha)^{2k} - 2k\alpha^2 + \alpha(2^n + 2k - 1) + \epsilon_\alpha \times (\alpha^{2k} - 2k\alpha^2 + \alpha(2k-1))}{(2^n - \alpha)^{2k}} \right) \\ &\leq \frac{\lambda_\alpha}{U_\alpha} \left(1 + \frac{-2k\alpha^2 + \alpha(2^n + 2k - 1) + \epsilon_\alpha \times (\alpha^{2k} - 2k\alpha^2 + \alpha(2k-1))}{(2^n - \alpha)^{2k}} \right) \end{aligned}$$

Since $U_1 = \lambda_1 = 2^{(2k-1)n}$:

$$\frac{\lambda_m}{U_m} \leq \prod_{\alpha=1}^{m-1} \left(1 + \frac{-2k\alpha^2 + \alpha(2^n + 2k - 1) + \epsilon_\alpha \times (\alpha^{2k} - 2k\alpha^2 + \alpha(2k-1))}{(2^n - \alpha)^{2k}} \right)$$

And Lemma 4 ends the proof. □

3.5 First Approximation of ϵ_α

Before evaluating ϵ_α, we need a technical lemma:

Lemma 6. *For every $\alpha \in \{2, \ldots, m\}$, one has:*

$$1 - \frac{2k\alpha}{2^n} \leq \frac{\lambda_\alpha}{2^{(2k-1)n} \lambda_{\alpha-1}} \leq 1. \tag{3}$$

Proof. We consider $g^1, \ldots, g^{2^k} \in J_n^\alpha$ satisfying the conditions $\lambda_{\alpha-1}$. To satisfy the conditions λ_α, there are $(2^n - (\alpha - 1))$ possibilities for each $g_\alpha^1, \ldots, g_\alpha^{2k-2}$ and there are $2(\alpha - 1)$ non-equalities left: $g_\alpha^{2k-1} \neq g_i^{2k-1}$ and $g_\alpha^{2k} \neq g_i^{2k}$ for all $i \leq \alpha - 1$. Since $g_\alpha^{2k} = g_\alpha^1 \oplus \cdots \oplus g_\alpha^{2k-1}$, one sees these $2(\alpha-1)$ non-equalities as equations on g_α^{2k-1}. So, there are between $2^n - 2(\alpha-1)$ and $2^n - (\alpha-1)$ possible choices for g_α^{2k-1} and 1 choice for g_α^{2k}. Then:

$$\lambda_{\alpha-1}(2^n - (\alpha-1))^{2k-2}(2^n - 2(\alpha-1)) \leq \lambda_\alpha \leq \lambda_{\alpha-1}(2^n - (\alpha-1))^{2k-1}$$

which is equivalent to:

$$\left(1 - \frac{\alpha-1}{2^n}\right)^{2k-2} \left(1 - \frac{2(\alpha-1)}{2^n}\right) \leq \frac{\lambda_\alpha}{2^{(2k-1)n}\lambda_{\alpha-1}} \leq \left(1 - \frac{\alpha-1}{2^n}\right)^{2k-1}.$$

Since the left term is bigger than $1 - \frac{2k\alpha}{2^n}$ and the right term is inferior to 1, it ends the proof. □

Lemma 7. *Every value $\lambda'_\alpha(X)$ with $X \in S$ satisfies:*

$$\frac{2^n \lambda'_\alpha(X)}{\lambda_\alpha} \leq 1 + \frac{2k\alpha}{\left(1 - \frac{2k\alpha}{2^n}\right) 2^n}.$$

Proof. We now express λ'_α in terms of $\lambda_{\alpha-1}$. Without loss of generality, we suppose that X involves g^1_α, otherwise we can just reorder the variables. Let i be any index such that g^i_α is not involved in X (this is possible since $X \in S$). Let $g^1, \ldots, g^{2k} \in J^\alpha_n$ such that the $\lambda_{\alpha-1}$ conditions are satisfied. We now count $\lambda'_\alpha(X)$. There are at most $2^n - (\alpha - 1)$ possible choices for each $g^j_\alpha, j \neq 1, i$. After we made these choices, there are two variables left: g^1_α and g^i_α. Since g^i_α is not involved in X, there is only, at most, one possible choice for g^1_α and there is, at most, one possible choice for g^i_α using the equation $g^1_\alpha \oplus \cdots \oplus g^{2k}_\alpha = 0$. Then:

$$\lambda'_\alpha(X) \leq (2^n - (\alpha - 1))^{2k-2} \lambda_{\alpha-1}.$$

Applying Lemma 6, one finds that:

$$\lambda'_\alpha(X) \leq (2^n - (\alpha - 1))^{2k-2} \left(\frac{1}{1 - \frac{2k\alpha}{2^n}}\right) \frac{\lambda_\alpha}{2^{(2k-1)n}}$$

Since $2^n - \alpha - 1 \leq 2^n$ and $\frac{1}{1-\frac{2k\alpha}{2^n}} = 1 + \frac{2k\alpha}{\left(1-\frac{2k\alpha}{2^n}\right)2^n}$, this ends the proof. □

Remark: These two technical lemmas formalize the intuition that, when one equation is added to the system, one degree of freedom is lost and this divides the number of possible solutions by around 2^n.

Finally

$$\epsilon_\alpha \leq \frac{2k\alpha}{\left(1 - \frac{2k\alpha}{2^n}\right) 2^n}.$$

First notice that if $q \leq \frac{2^n}{2k}$, $-2k\alpha^2 + \alpha(2^n) \geq 0$. Then, from Lemma 5,

$$\mathbf{Adv}^{\mathrm{cpa}}_{f_1 \oplus \ldots \oplus f_k} \leq 2 \left(\prod_{\alpha=1}^{q-1} \left(1 + \frac{-2k\alpha^2 + \alpha(2^n + 2k - 1) + \epsilon_\alpha \times (\alpha^{2k} - 2k\alpha^2 + \alpha(2k-1))}{(2^n - \alpha)^{2k}} \right) - 1 \right)^{1/3}.$$

If $q \leq \frac{2^n}{2k}$, all the terms of the product are greater than 1 and

$$\begin{aligned}
\mathbf{Adv}^{\mathrm{cpa}}_{f_1 \oplus \ldots \oplus f_k} &\leq 2 \left(\prod_{\alpha=1}^{q-1} \left(1 + \frac{-2k\alpha^2 + \alpha(2^n + 2k - 1)}{(2^n - \alpha)^{2k}} + \frac{2k\alpha \times (\alpha^{2k} - 2k\alpha^2 + \alpha(2k-1))}{\left(1 - \frac{2k\alpha}{2^n}\right) 2^n \times (2^n - \alpha)^{2k}} \right) - 1 \right)^{1/3} \\
&\leq 2 \left(\prod_{\alpha=1}^{q-1} \left(1 + \frac{\alpha 2^n}{(2^n - \alpha)^{2k}} + \frac{2k\alpha^{2k+1}}{\left(1 - \frac{2k\alpha}{2^n}\right) 2^n (2^n - \alpha)^{2k}} \right) - 1 \right)^{1/3} \\
&\leq 2 \left(\left(1 + \frac{q 2^n}{(2^n - q)^{2k}} + \frac{2kq^{2k+1}}{\left(1 - \frac{2kq}{2^n}\right) 2^n (2^n - q)^{2k}} \right)^q - 1 \right)^{1/3}.
\end{aligned}$$

Thus we have proven that:

Theorem 2 (Upper Bound of the Advantage Using H_σ). *The maximal advantage an adversary can get using q queries, with $q \leq \frac{2^n}{2k}$ verifies:*

$$\mathbf{Adv}^{\mathrm{cpa}}_{f_1 \oplus \ldots \oplus f_k} \leq 2 \left(\left(1 + \frac{q 2^n}{(2^n - q)^{2k}} + \frac{2kq^{2k+1}}{\left(1 - \frac{2kq}{2^n}\right) 2^n (2^n - q)^{2k}} \right)^q - 1 \right)^{1/3}.$$

Notice that

$$\mathbf{Adv}^{\mathrm{cpa}}_{f_1 \oplus \ldots \oplus f_k} \lesssim 2 \left(\frac{q^2}{2^{(2k-1)n}(1 - \frac{q}{2^n})^{2k}} + \frac{2kq^{2k+2}}{2^{(2k+1)n}(1 - \frac{6kq}{2^n})} \right)^{1/3}.$$

Since $k \geq 3$ and $q \leq 2^n$, the first term is negligible in front of 1. Moreover, when $q^{2k+2} \ll 2^{(2k+1)n}$, $\mathbf{Adv}^{\mathrm{cpa}}_{f_1 \oplus \ldots \oplus f_k} \ll 1$. Hence we have proven that the XOR of k permutations is safe as long as $q \ll 2^{\frac{2k+1}{2k+2} n}$ with this first technique.

4 Security Bound from the Standard *H* Technique

We now use the "standard H technique", i.e. proofs from the general result (the Corollary 8) below. In this section, $\mathbb{E}[h_q]$ is noted $\tilde{h}_q$ to lighten the notations.

Corollary 8. *Let $\alpha > 0$. If, for every sequence $b = (b_i)_{1 \leq i \leq q} \in I_n^q$*

$$h_q(b) \geq (1 - \alpha)\tilde{h}_q,$$

then

$$\mathbf{Adv}^{\mathrm{cpa}}_{f_1 \oplus \ldots \oplus f_k} \leq \alpha.$$

Proof. This result comes immediately from Theorem 1 with $\beta = 0$ and Lemmas 1 and 3. □

4.1 First Approximation

Let us study $\frac{h_\alpha}{\tilde{h}_\alpha}$.

One has:

$$\tilde{h}_{\alpha+1} = \tilde{h}_\alpha \frac{(2^n - \alpha)^k}{2^n}.$$

We now evaluate $h_{\alpha+1}$ from h_α. From the definition of h_α (see Sect. 3.1), we see that $h_{\alpha+1}$ is the number of sequence $(P_i^j)_{1 \leq i \leq m, 1 \leq j \leq k}$ such that:

- the h_α conditions hold;
- $P^1_{\alpha+1} \oplus \ldots \oplus P^k_{\alpha+1} = b_{\alpha+1}$, this equation will be called X;
- $P^j_{\alpha+1} \neq P^j_i$ for every $1 \leq i \leq \alpha$, $1 \leq j \leq k$.

Let β_i, $1 \leq k\alpha$ be the $k\alpha$ equations which should be false. Let, for $1 \leq i \leq k\alpha$, B_i be the set of the $\left(P_i^j\right)_{1\leq i\leq \alpha+1, 1\leq j\leq k}$ for which the h_α conditions and the equation β_i hold.

From the inclusion-exclusion principle, we get:

$$\begin{aligned} h_{\alpha+1} &= 2^{(k-1)n}h_\alpha - |\cup_{i=1}^{k\alpha} B_i| \\ &= 2^{(k-1)n}h_\alpha + \sum_{1\leq l\leq k\alpha} (-1)^l \sum_{i_1<\ldots<i_l} |B_{i_1}\cap\ldots\cap B_{i_l}|. \end{aligned}$$

When $k+1$ sets are intersected, at least two equations will use the same $P_{\alpha+1}^j$ variable, which is in contradiction with h_α. Thus,

$$h_{\alpha+1} = 2^{(k-1)n}h_\alpha + \sum_{1\leq l\leq k} (-1)^l \sum_{i_1<\ldots<i_l} |B_{i_1}\cap\ldots\cap B_{i_l}|. \tag{4}$$

We study the number of possible messages in function of the number of sets in the intersection.

- l **equations,** $1 \leq l \leq k-1$**:**

If we want $|B_{i_1}\cap\ldots\cap B_{i_l}| \neq 0$, every new β_i equation should bring a new variable $P_{\alpha+1}^j$. In this case, X and β_i fix $l+1$ variables, the remaining ones are free, so $|B_{i_1}\cap\ldots\cap B_{i_l}| = 2^{(k-l-1)n}h_\alpha$ and

$$\sum_{i_1<\ldots<i_l} |B_{i_1}\cap\ldots\cap B_{i_l}| = \binom{k}{l} \alpha^l 2^{(k-l-1)n}h_\alpha$$

- k **equations:**

As well as above, in order to have $|B_{i_1}\cap\ldots\cap B_{i_k}| \neq 0$, there must be an equation in every new variable:

$$P_{\alpha+1}^j = P_{i_j}^j,\ 1 \leq j \leq k.$$

So the condition $P_{\alpha+1}^1 \oplus \ldots \oplus P_{\alpha+1}^k = b_{\alpha+1}$ becomes:

$$P_{i_1}^1 \oplus \ldots \oplus P_{i_k}^k = b_{\alpha+1}.$$

Let $h'_\alpha(b_1,\ldots,b_{\alpha+1})(i_1,\ldots,i_k)$ or $h'_\alpha(i_1,\ldots,i_k)$ the number of $(P_i^j)_{1\leq i\leq\alpha, 1\leq j\leq k} \in I_n^{k\alpha}$ such that:

– the conditions h_α hold,
– $P_{i_1}^1 \oplus \ldots \oplus P_{i_k}^k = b_{\alpha+1}$.

Let $Y(i_1,\ldots,i_k)$ be this equality. Thus

$$\sum_{i_1<\ldots<i_k} |B_{i_1}\cap\ldots\cap B_{i_k}| = \sum_{1\leq i_1,\ldots,i_k\leq\alpha} h'_\alpha(i_1,\ldots,i_k).$$

From (4), we have:

$$h_{\alpha+1} = \frac{(2^n-\alpha)^k - (-1)^k\alpha^k}{2^n} h_\alpha + (-1)^k \sum_{1\le i_1,\ldots,i_k\le\alpha} h'_\alpha(i_1,\ldots,i_k). \quad (5)$$

Remark: if k is even, one has:

$$h_{\alpha+1} \ge h_\alpha \left(\frac{(2^n-\alpha)^k - \alpha^k}{2^n}\right).$$

So

$$\frac{h_{\alpha+1}}{\tilde{h}_{\alpha+1}} \ge \frac{h_\alpha}{\tilde{h}_\alpha}\left(1 - \frac{\alpha^k}{(2^n-\alpha)^k}\right).$$

As $h_1 = \tilde{h}_1 = 2^{(k-1)n}$,

$$\begin{aligned} h_q &\ge \tilde{h}_q \left(1 - \frac{q^k}{(2^n-q)^k}\right)^q \\ &\ge \tilde{h}_q \left(1 - \frac{q^{k+1}}{(2^n-q)^k}\right) \end{aligned}$$

Then, using Corollary 8,

$$\mathbf{Adv}^{\mathrm{cpa}}_{f_1\oplus\ldots\oplus f_k} \le \frac{q^{k+1}}{(2^n-q)^k}.$$

The upper bound we get in this case is in the same order of magnitude as the one from [9]. If we study more closely h'_α, we will get a better inequality.

4.2 Second Approximation

In this section, we suppose that $k \ge 3$.

Let $M = \{i,\ 1 \le i \le \alpha,\ b_i = b_{\alpha+1}\}$. If $i \in M$, we have $h'_\alpha(i,\ldots,i) = h_\alpha$ and if $i \notin M$, $h'_\alpha(i,\ldots,i) = 0$. Furthermore, in order to be compatible with h_α, if $i \in M$, for each $1 \le j \le \alpha$, $i \ne j$, $h'_\alpha(j,i,\ldots,i) = h'_\alpha(i,j,\ldots,i) = \ldots = h'_\alpha(i,\ldots,i,j) = 0$. Let I be the set of the tuples that do not satisfy these requirements. Then $|I| = \alpha^k - \alpha - k|M|(\alpha-1)$. By applying (5), one gets:

$$h_{\alpha+1} = \frac{(2^n-\alpha)^k - (-1)^k\alpha^k + (-1)^k 2^n|M|}{2^n} h_\alpha + (-1)^k \sum_{(i_1,\ldots,i_k)\in I} h'_\alpha(i_1,\ldots,i_k). \quad (6)$$

We now need a technical lemma:

Lemma 9. *If* $i = (i_1,\ldots,i_k) \in I$,

$$1 - \frac{3\alpha}{(2^n-\alpha)(1-\frac{\alpha}{2^n})} \le \frac{2^n h'_\alpha(i_1,\ldots,i_k)}{h_\alpha} \le \frac{1}{1-\frac{3\alpha}{2^n}}.$$

Proof. Without loss of generality, we can suppose that $i_1 = \alpha$ and $i_2 = \alpha - 1$ (because we can reorder the queries). Let us evaluate h'_α and h_α from $h_{\alpha-2}$. To get h_α from $h_{\alpha-2}$, we have $2k$ new variables $P^j_{\alpha-1}$ and P^j_α, $1 \le j \le k$, such that:

- $P^1_\alpha \oplus \ldots \oplus P^k_\alpha = b_\alpha$,
- $P^1_{\alpha-1} \oplus \ldots \oplus P^k_{\alpha-1} = b_{\alpha-1}$,
- $\forall j, 1 \le j \le k, \forall i, 1 \le i \le \alpha - 2, P^j_{\alpha-1} \neq P^j_i$,
- $\forall j, 1 \le j \le k, \forall i, 1 \le i \le \alpha - 1, P^j_\alpha \neq P^j_i$.

We decide that the first equation will fix $P^1_{\alpha-1}$ and the next one P^1_α. For $j \ge 3$, we have respectively $2^n - (\alpha - 2)$ and $2^n - (\alpha - 1)$ possibilities for $P^j_{\alpha-1}$ and P^j_α. When these messages have been chosen, only $P^2_{\alpha-1}$ and P^2_α remain, and they must satisfy:

- $P^2_{\alpha-1} \neq P^2_i$, $1 \le i \le \alpha - 2$,
- $P^2_{\alpha-1} \neq P^1_i \oplus b_{\alpha-1} \oplus P^3_{\alpha-1} \oplus \ldots \oplus P^k_{\alpha-1}$, $1 \le i \le \alpha - 2$,
- $P^2_\alpha \neq P^2_i$, $1 \le i \le \alpha - 1$,
- $P^2_\alpha \neq P^1_i \oplus b_\alpha \oplus P^3_\alpha \oplus \ldots \oplus P^k_\alpha$, $1 \le i \le \alpha - 1$.

There are for $P^2_{\alpha-1}$ between $2^n - 2(\alpha - 2)$ and $2^n - (\alpha - 2)$ choices and for $P^2_{\alpha-1}$ between $2^n - 2(\alpha - 1)$ and $2^n - (\alpha - 1)$. Thus

$$(2^n - (\alpha - 2))^{k-2}(2^n - (\alpha - 1))^{k-2}(2^n - 2(\alpha - 2))(2^n - 2(\alpha - 1)) \le \tfrac{h_\alpha}{h_{\alpha-2}}, \quad (7)$$

$$\tfrac{h_\alpha}{h_{\alpha-2}} \le (2^n - (\alpha - 2))^{k-1}(2^n - (\alpha - 1))^{k-1}. \quad (8)$$

In order to go from $h_{\alpha-2}$ to h'_α, we also have $2k$ new variables $P^j_{\alpha-1}$ and P^j_α, $1 \le j \le k$, such that:

- $P^1_{\alpha-1} \oplus \ldots \oplus P^k_{\alpha-1} = b_{\alpha-1}$,
- $P^1_\alpha = b_{\alpha+1} \oplus P^2_{\alpha-1} \oplus P^3_{i_3} \oplus \ldots \oplus P^k_{i_k}$,
- $P^1_\alpha \oplus \ldots \oplus P^k_\alpha = b_\alpha$,
- $\forall j, 1 \le j \le k, \forall i, 1 \le i \le \alpha - 2, P^j_{\alpha-1} \neq P^j_i$,
- $\forall j, 1 \le j \le k, \forall i, 1 \le i \le \alpha - 1, P^j_\alpha \neq P^j_i$.

We have, for $j \ge 4$, respectively $2^n - (\alpha - 2)$ and $2^n - (\alpha - 1)$ possibilities for $P^j_{\alpha-1}$ and P^j_α. From these 3 equalities, we can fix the following variables:

1. $P^1_{\alpha-1} = b_{\alpha-1} \oplus P^2_{\alpha-1} \oplus \ldots \oplus P^k_{\alpha-1}$,
2. $P^1_\alpha = b_{\alpha+1} \oplus P^2_{\alpha-1} \oplus P^3_{i_3} \oplus \ldots \oplus P^k_{i_k}$,
3. $P^2_\alpha = (b_{\alpha+1} \oplus b_\alpha) \oplus P^2_{\alpha-1} \oplus (P^3_{i_3} \oplus P^3_\alpha) \oplus \ldots \oplus (P^k_{i_k} \oplus P^k_\alpha)$.

Then

- the condition $\forall i, 1 \le i \le \alpha - 2, P^1_{\alpha-1} \neq P^1_i$ becomes:

$$\forall i, 1 \le i \le \alpha - 2, P^2_{\alpha-1} \neq P^1_i \oplus b_{\alpha-1} \oplus P^3_{\alpha-1} \oplus \ldots \oplus P^k_{\alpha-1},$$

- $\forall i, 1 \le i \le \alpha - 1, P^1_\alpha \neq P^1_i$ becomes:

$$\forall i, 1 \le i \le \alpha - 1, P^2_{\alpha-1} \neq b_{\alpha+1} \oplus P^1_i \oplus P^3_{i_3} \oplus \ldots \oplus P^k_{i_k},$$

– $\forall i, 1 \leq i \leq \alpha - 2, P^2_\alpha \neq P^1_i$ becomes:

$$\forall i, 1 \leq i \leq \alpha - 2,\ P^2_{\alpha-1} \neq (b_{\alpha+1} \oplus b_\alpha) \oplus P^2_i \oplus (P^3_{i_3} \oplus P^3_\alpha) \oplus \ldots \oplus (P^k_{i_k} \oplus P^k_\alpha)$$

For $P^2_\alpha \neq P^2_{\alpha-1}$, there are two cases. If $i_3 = \ldots = i_k = \alpha$, since $(i_1, \ldots, i_k) \in I$, we have $b_{\alpha+1} \neq b_\alpha$ and this non-equality is automatically verified. Else, this means that there is an index $3 \leq j \leq k$ such that $i_j \neq \alpha$, e.g. $j = 3$. Then $P^2_\alpha \neq P^2_{\alpha-1}$ becomes:

$$P^3_\alpha \neq (b_{\alpha+1} \oplus b_\alpha) \oplus P^3_{i_3} \oplus \ldots \oplus (P^k_{i_k} \oplus P^k_\alpha).$$

Thus, after the other messages have been chosen, there are between $2^n - \alpha$ and $2^n - (\alpha - 1)$ possibilities for P^3_α, $2^n - (\alpha - 2)$ possibilities for $P^3_{\alpha-1}$ and finally between $2^n - (4\alpha - 7)$ and $2^n - (\alpha - 2)$ possibilities for $P^2_{\alpha-1}$. Then

$$(2^n - (\alpha - 2))^{k-2}(2^n - (\alpha - 1))^{k-3}(2^n - \alpha)(2^n - (4\alpha - 7)) \leq \frac{h'_\alpha}{h_{\alpha-2}} \quad (9)$$

$$(2^n - (\alpha - 2))^{k-1}(2^n - (\alpha - 1))^{k-2} \geq \frac{h'_\alpha}{h_{\alpha-2}}. \quad (10)$$

From 7 and 9 we can deduce the following inequalities that allow us to get the result we want:

$$\frac{2^n h'_\alpha}{h_{\alpha-2}} \geq 2^n \frac{(2^n - 4\alpha + 7)(2^n - \alpha)}{(2^n - (\alpha-2))(2^n - (\alpha-1))^2},$$
$$\frac{2^n h'_\alpha}{h_{\alpha-2}} \leq 2^n \frac{2^n - (\alpha-2)}{(2^n - 2(\alpha-2))(2^n - 2(\alpha-1)).}$$

□

Remark: if we suppose $\alpha < \frac{2^n}{12}$, we get

$$0 < 1 - \frac{12\alpha}{2^n} \leq \frac{2^n h'_\alpha(i_1, \ldots, i_k)}{h_\alpha} \leq 1 + \frac{3\alpha}{2^n - 3\alpha}. \quad (11)$$

One has:

$$\frac{h_{\alpha+1}}{\tilde{h}_{\alpha+1}} = \frac{h_\alpha}{\tilde{h}_\alpha}\left(1 + \frac{(-1)^{k+1}\alpha^k}{(2^n - \alpha)^k} + (-1)^k \frac{2^n |M|}{(2^n - \alpha)^k} + (-1)^k \frac{\sum \frac{2^n h'_\alpha}{h_\alpha}}{(2^n - \alpha)^k}\right) \quad (12)$$

$$= \frac{h_\alpha}{\tilde{h}_\alpha}(1 - A_\alpha) \quad (13)$$

where

$$A_\alpha := \frac{(-1)^k \alpha^k}{(2^n - \alpha)^k} - (-1)^k \frac{2^n |M|}{(2^n - \alpha)^k} - (-1)^k \frac{\sum \frac{2^n h'_\alpha}{h_\alpha}}{(2^n - \alpha)^k}.$$

Lemma 10. *If* $q < \frac{2^n}{12}$,

$$A_\alpha \leq \frac{k.2^n \alpha}{(2^n - \alpha)^k} + 12 \frac{\alpha^{k+1}}{(2^n - 3\alpha)(2^n - \alpha)^k}.$$

Proof. We have to study A_α according to the parity of k.

- k **even:**

$$\begin{aligned} A_\alpha &\leq \frac{\alpha^k}{(2^n-\alpha)^k} - \frac{2^n|M|}{(2^n-\alpha)^k} - \frac{(\alpha^k-\alpha-k|M|(\alpha-1))(1-\frac{12\alpha}{2^n})}{(2^n-\alpha)^k} \\ &\leq -\frac{2^n|M|}{(2^n-\alpha)^k} + \frac{(\alpha+k|M|(\alpha-1))(1-\frac{12\alpha}{2^n})}{(2^n-\alpha)^k} + 12\frac{\alpha^{k+1}}{2^n(2^n-\alpha)^k} \\ &\leq \frac{k.\alpha^2}{(2^n-\alpha)^k} + 12\frac{\alpha^{k+1}}{2^n(2^n-\alpha)^k} \end{aligned}$$

- k **odd:**

$$\begin{aligned} A_\alpha &\leq -\frac{\alpha^k}{(2^n-\alpha)^k} + \frac{2^n|M|}{(2^n-\alpha)^k} + \frac{(\alpha^k-\alpha-k|M|(\alpha-1))(1+\frac{3\alpha}{2^n-3\alpha})}{(2^n-\alpha)^k} \\ &\leq \frac{2^n|M|}{(2^n-\alpha)^k} - \frac{(\alpha+k|M|(\alpha-1))(1+\frac{3\alpha}{2^n-3\alpha})}{(2^n-\alpha)^k} + \frac{3\alpha^{k+1}}{(2^n-\alpha)^k(2^n-3\alpha)} \\ &\leq \frac{2^n\alpha}{(2^n-\alpha)^k} + \frac{3\alpha^{k+1}}{(2^n-\alpha)^k(2^n-3\alpha)} \end{aligned}$$

So, in both cases,

$$A_\alpha \leq \frac{k.2^n\alpha}{(2^n-\alpha)^k} + 12\frac{\alpha^{k+1}}{(2^n-3\alpha)(2^n-\alpha)^k},$$

□

From this lemma and 12,

$$\frac{h_{\alpha+1}}{\tilde{h}_{\alpha+1}} \geq \frac{h_\alpha}{\tilde{h}_\alpha}\left(1 - \frac{k.2^n\alpha}{(2^n-\alpha)^k} - 12\frac{\alpha^{k+1}}{(2^n-3\alpha)(2^n-\alpha)^k}\right).$$

Since $h_1 = \tilde{h}_1$, we get:

$$\begin{aligned} \frac{h_q}{\tilde{h}_q} &\geq \left(1 - \frac{k2^nq}{(2^n-q)^k} - 12\frac{q^{k+1}}{(2^n-3q)(2^n-q)^k}\right)^q \\ &\geq 1 - \frac{kq^2.2^n}{(2^n-q)^k} - 12\frac{q^{k+2}}{(2^n-3q)(2^n-q)^k}. \end{aligned}$$

Thus, with Corollary 8, we have proven that, when $q < \frac{2^n}{12}$:

$$\mathbf{Adv}^{\mathrm{cpa}}_{f_1\oplus\ldots\oplus f_k} \leq \frac{kq^2.2^n}{(2^n-q)^k} + 12\frac{q^{k+2}}{(2^n-3q)(2^n-q)^k} \tag{14}$$

$$\leq \frac{kq^2}{2^{(k-1)n}(1-k\frac{q}{2^n})} + 12\frac{q^{k+2}}{2^{(k+1)n}(1-(k+3)\frac{q}{2^n})}. \tag{15}$$

Hence we get the following result:

Table 1. Comparison of the bounds on the advantage from 3 techniques

Technique	Upper bound for $\mathbf{Adv}^{\mathrm{cpa}}_{f_1 \oplus \ldots \oplus f_k}$	Order of magnitude
S. Lucks	$2^{-k(n-1)} \sum_{0 \leq i < q} i^k$	$\mathcal{O}\left(\frac{q^{k+1}}{2^{k(n-1)}}\right)$
H	$\frac{kq^2}{2^{(k-1)n}(1-k\frac{q}{2^n})} + 12\frac{q^{k+2}}{2^{(k+1)n}(1-(k+3)\frac{q}{2^n})}$	$O\left(\frac{q^{k+2}}{2^{(k+1)n}}\right)$
H_σ	$2\left(\frac{q^2}{2^{(2k-1)n}(1-\frac{q}{2^n})^{2k}} + \frac{2kq^{2k+2}}{2^{(2k+1)n}(1-\frac{6kq}{2^n})}\right)^{1/3}$	$O\left(\left(k\frac{q^{2k+2}}{2^{(2k+1)n}}\right)^{1/3}\right)$

Table 2. Upper bound plotted versus the logarithm of q

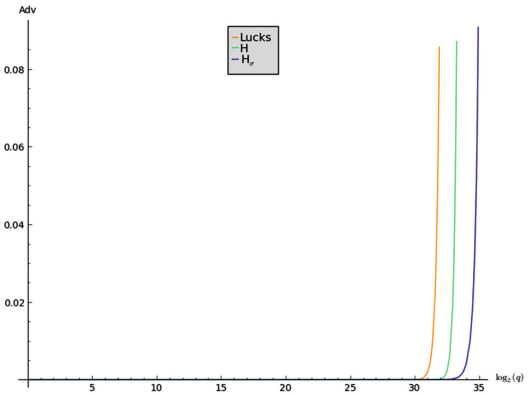
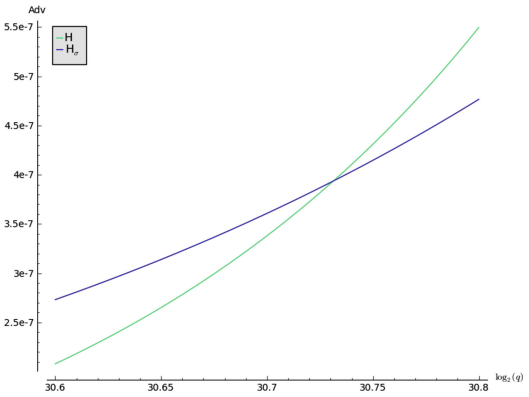


Table 3. Upper bound plotted versus the logarithm of q: comparison between H and H_σ



Theorem 3 (Upper Bound for the Advantage with the Standard H Technique). *Let $k \geq 3$ and $q < \frac{2^n}{12}$. The advantage to distinguish, with q queries, the XOR of k bijections from a function $f \in_R F_n$ satisfies:*

$$\mathbf{Adv}^{\mathrm{cpa}}_{f_1 \oplus \ldots \oplus f_k} \leq \frac{kq^2}{2^{(k-1)n}(1-k\frac{q}{2^n})} + 12\frac{q^{k+2}}{2^{(k+1)n}(1-(k+3)\frac{q}{2^n})}.$$

Since $k \geq 3$, the first term is negligible when $q \ll 2^n$. This theorem shows that the XOR of k bijections is indistinguishable when $q \ll 2^{\frac{k+1}{k+2}n}$. This upper bound on q is worse than the previous one, but if $q \ll 2^{\frac{k+2}{k+4}n}$ (i.e. for small values of q) this new upper bound on the advantage is actually better.

5 Conclusion

This table regroups our results and the previous one from S. Lucks in [9], with order of magnitudes for these bounds beyond the birthday bound (Tables 1, 2 and 3):

The upper bound we got with the coefficients H technique is smaller than the one from [9] by a factor $\frac{q}{2^n}$. The one we proved with the coefficients H_σ technique allows us to have $\mathbf{Adv}^{\mathrm{cpa}}_{f_1 \oplus \ldots \oplus f_k} \ll 1$ when $q \ll 2^{\frac{2k+1}{2k+2}n}$ instead of $q \ll 2^{\frac{k}{k+1}n}$ for [9]. For example with $k = 3$ we have proven that $\mathbf{Adv}^{\mathrm{cpa}}_{f_1 \oplus \ldots \oplus f_k} \ll 1$ when $q \ll 2^{\frac{7}{8}n}$ instead of $q \ll 2^{\frac{3}{4}n}$. However, when q is fixed and k increases, the upper bound from the H technique becomes better than the one from H_σ. This graph shows the evolution of the order of magnitude of these three upper bounds in function of the logarithm of q, with $k = 5$ and $n = 40$:

Here is a more accurate view of the region where the curves from H and H_σ intersect:

This illustrates that, depending on the value of q, our best bound can be the one from Sect. 3 or the one from Sect. 4. Moreover, the curve from [9] does not appear in this second graph because its values were much higher than ours (around $6 \cdot 10^{-4}$ whereas the bounds from this article are around $4 \cdot 10^{-7}$ in this graph). This shows why the two techniques studied in this paper are both useful.

References

1. Aiello, W., Venkatesan, R.: Foiling birthday attacks in length-doubling transformations. In: Maurer, U.M. (ed.) EUROCRYPT 1996. LNCS, vol. 1070, pp. 307–320. Springer, Heidelberg (1996)
2. Bellare, M., Impagliazzo, R.: A Tool for Obtaining Tighter Security Analyses of Pseudorandom Function Based Constructions, with Applications to PRP to PRF Conversion. ePrint Archive 1999/024: Listing for 1999 (1999)
3. Bellare, M., Krovetz, T., Rogaway, P.: Luby-rackoff backwards: increasing security by making block ciphers non-invertible. In: Nyberg, K. (ed.) EUROCRYPT 1998. LNCS, vol. 1403, pp. 266–280. Springer, Heidelberg (1998)
4. Goldreich, O., Goldwasser, S., Micali, S.: How to construct random functions. J. ACM **33**(4), 792–807 (1986)
5. Hall, C., Wagner, D., Kelsey, J., Schneier, B.: Building PRFs from PRPs. In: Krawczyk, H. (ed.) CRYPTO 1998. LNCS, vol. 1462, p. 370. Springer, Heidelberg (1998)
6. Levin, L.: One way functions and pseudorandom generators. Combinatorica **7**(4), 357–363 (1987)
7. Luby, M., Rackoff, C.: How to construct pseudorandom permutations from pseudorandom functions. SIAM J. Comput. **17**(2), 373–386 (1988)

8. Lucks, S.: Faster Luby-Rackoff ciphers. In: Gollmann, D. (ed.) FSE 1996. LNCS, vol. 1039, pp. 189–203. Springer, Heidelberg (1996)
9. Lucks, S.: The sum of PRPs is a secure PRF. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 470–484. Springer, Heidelberg (2000)
10. Mandal, A., Patarin, J., Nachef, V.: Indifferentiability beyond the birthday bound for the XOR of two public random permutations. In: Gong, G., Gupta, K.C. (eds.) INDOCRYPT 2010. LNCS, vol. 6498, pp. 69–81. Springer, Heidelberg (2010)
11. Naor, M., Reingold, O.: On the construction of pseudo-random permutations: Luby-Rackoff revisited. J. Cryptol. **12**(1), 29–66 (1999)
12. Patarin, J.: A proof of security in $O(2^n)$ for the XOR of two random permutation. In: Safavi-Naini, R. (ed.) ICITS 2008. LNCS, vol. 5155, pp. 232–248. Springer, Heidelberg (2008)
13. Patarin, J.: Generic Attacks for the XOR of k Random Permutations. Available on eprint (2008)
14. Patarin, J.: The "coefficients H" technique. In: Avanzi, R.M., Keliher, L., Sica, F. (eds.) SAC 2008. LNCS, vol. 5381, pp. 328–345. Springer, Heidelberg (2009)
15. Patarin, J.: Security in $O(2^n)$ for the XOR of Two Random Permutations - Proof with the standard H technique - Available on eprint (2013)

Impact of ANSI X9.24-1:2009 Key Check Value on ISO/IEC 9797-1:2011 MACs

Tetsu Iwata[1(✉)] and Lei Wang[2]

[1] Nagoya University, Nagoya, Japan
iwata@cse.nagoya-u.ac.jp
[2] Nanyang Technological University, Singapore, Singapore
wang.lei@ntu.edu.sg

Abstract. ANSI X9.24-1:2009 specifies the key check value, which is used to verify the integrity of the blockcipher key. This value is defined as the most significant bits of the ciphertext of the zero block, and is assumed to be publicly known data for verification. ISO/IEC 9797-1:2011 illustrates a total of ten CBC MACs, where one of these MACs, the basic CBC MAC, is widely known to be insecure. In this paper, we consider the remaining nine CBC MACs and derive the quantitative security impact of using the key check value. We first show attacks against five MACs by taking advantage of the knowledge of the key check value. We then *prove* that the analysis is tight, in a concrete security paradigm. For the remaining four MACs, we prove that *the standard birthday bound still holds* even with the presence of the key check value. As a result, we obtain a complete characterization of the impact of using ANSI X9.24-1 key check value with the ISO/IEC 9797-1 MACs.

Keywords: ANSI X9.24-1:2009 · Key check value · ISO/IEC 9797-1:2011 · CBC MAC · Proof of security

1 Introduction

Background. A Message Authentication Code, or a MAC, is a fundamental cryptographic primitive to ensure the authenticity of messages. A MAC is a keyed function that takes a message as its input and produces a fixed length string called a tag. The tag is then used to verify the integrity of the message, i.e., the message is indeed originated from the intended party who shares the key.

There are several ways of constructing MACs, and CBC MAC is a widely used MAC based on a blockcipher. While the basic CBC MAC has a proof of security when it is used for messages of one fixed length [4,6], it is known that it allows the so called length-extension attack when the message length can vary. To avoid the weakness, several variants of CBC MAC were proposed. ISO/IEC 9797-1:2011 [14] specifies six different versions of CBC MACs, where each MAC is defined by specifying the final iteration and the output transformation. The six MACs are further classified by specifying the key derivation method and the

C. Cid and C. Rechberger (Eds.): FSE 2014, LNCS 8540, pp. 303–322, 2015.
DOI: 10.1007/978-3-662-46706-0_16

padding method, and Annex C in [14] illustrates a total of ten different CBC MACs depending on the choice of these methods.

ANSI X9.24-1:2009 [2] specifies the manual and automated management of keying material used for financial services. The services include point-of-sale (POS) transactions (debit and credit), automated teller machine (ATM) transactions, messages among terminals and financial institutions, and interchange messages among acquirers, switches, and card issuers [2]. Annex C in [2] suggests the use of the *key check value* for the integrity verification of the blockcipher key. Let $E_K : \{0,1\}^n \to \{0,1\}^n$ be a blockcipher with a key K. ANSI X9.24-1 suggests to use the most significant s bits of $E_K(0^n)$, a ciphertext of the zero block, as the key check value for the key K, where E is DES or TDES [3] and s is 16 or 24. That is, the key check value, KCV, is defined as $\mathsf{KCV} = \mathsf{msb}_s(E_K(0^n))$. The value KCV is then used to verify the integrity of K, or as the ID for K. Therefore, KCV is treated as a public value and it can be transmitted, sent, or stored in clear. This implies that an adversary has chance to learn this value.

In some MACs and other blockcipher modes of operation, the value $E_K(0^n)$ is used to derive the "sub-key." For example, MAC5 in [14], also known as CMAC [11], uses this value as a sub-key that has to be kept secret from the adversaries. Other examples that use $E_K(0^n)$ as a sub-key include GCM [12,22], PMAC [7,28], and OCB [28,30]. MAC5 has the proof of security [15,16], while disclosing a part of $E_K(0^n)$ is not taken into account in the proof, and it is anticipated that there is some security loss when it is used with the key check value. Indeed, [2,11] give an explicit warning that the value $E_K(0^n)$ has to be kept secret, but there is no prior work that derives the security loss in a *quantitative* way, which is the main question solved in the present paper.

Contributions. We consider the security of ten MACs, CBC MAC and its variants, specified in [14], in the presence of the key check value. We first consider MAC2.1, which is also known as EMAC [9]. Petrank and Rackoff showed that it is a secure MAC [25]. The security bound without the use of the key check value is $O(\sigma^2/2^n)$ [25] assuming that the underlying blockcipher satisfies an appropriate security notion, where σ is the total length of queries in blocks made by the adversary, and n is the block size of the underlying blockcipher in bits. This implies that the adversary needs to make $\Omega(2^{n/2})$ queries in order to make a forgery. In its specification, the value $E_K(0^n)$ does not appear as in the case for MAC5. However, based on a similar technique to the one in [18], we present attacks with $O(2^{(n-s)/2})$ queries when it is used with the s-bit key check value. For MAC5, the security bound without the use of the key check value is $O(\sigma^2/2^n)$ [15,16]. We show that almost the same attacks against MAC2.1 can be used to attack MAC5 with $O(2^{(n-s)/2})$ queries when it is used with the s-bit key check value. We point out that similar attacks can be used against MAC2.2 (EMAC [9]), MAC3 (ANSI retail MAC [1]), and MAC6.2 (FCBC [8]).

We then formalize a security notion of a variant of a pseudorandom function that captures the key check value, and for these five MACs, we *prove* that the analysis is tight. That is, under the appropriate assumption about the underlying blockcipher, we show that the adversary actually needs to make $\Omega(2^{(n-s)/2})$

queries in order to mount the attacks. For the remaining four MACs, MAC1.2, MAC4.1 (MacDES [19]), MAC4.2 (MacDES [19]), and MAC6.1 (FCBC [8]), the situation is quite different. Even if the key check value consists of the entire n bits of $E_K(0^n)$, we show that the standard birthday bound still holds, and there is almost no security loss for these MACs. As a result, we obtain a complete characterization of the impact of using ANSI X9.24-1 key check value with the ISO/IEC 9797-1 MACs. See Table 1 for the summary of this paper. In the table, the block size of the underlying blockcpher is n bits, s is the bit length of the key check value, and σ is the total length of queries in blocks made by the adversary. All results for MAC1.1 are widely known. Results on existential forgeries for MAC1.2, MAC4.1, MAC4.2, and MAC6.1 follow from [14,27]. The attacks are possible without using the key check value. We note that there are other attacks for all these MACs, e.g., a key recovery attack. See [14,29] for more details.

We also discuss several generic ways to avoid the security loss in the presence of the key check value.

Remarks. We argue that our security bounds, both $O(\sigma^2/2^{n-s})$ and $O(\sigma^2/2^n)$, are non-trivial, in the sense that there are blockcipher modes of operation that become completely insecure if the key check value is used. For instance consider the CTR encryption mode where the initial counter value starts with 0^n. It is not hard to see that the adversary succeeds in distinguishing between a ciphertext and a random string of the same length as the ciphertext even if the key check value consists of one bit.

We remark that the presence of the key check value can be considered as a special case of leakage of the internal state. Leakage resilient MACs are proposed, e.g., in [10,21]. In contrast to designing new schemes, the purpose of this paper is to analyze the security of widely standardized and deployed MACs when used with the key check value, a common practice in financial applications.

We also remark that this paper does not show the analysis of MACs in the older version of ISO/IEC 9797-1 that was published in 1999 [13]. Specifically, MAC5 and MAC6 in the 2011 version are different from those in the 1999 version. We leave the analyses of these MACs as open questions.

2 Preliminaries

For a finite set $\mathcal{X}$, $X \xleftarrow{\$} \mathcal{X}$ means that an element X is chosen from $\mathcal{X}$ uniformly at random. Let $\{0,1\}^*$ be the set of all finite bit strings including the empty string. If $X, Y \in \{0,1\}^*$ are equal-length strings then $X \oplus Y$ is their bitwise xor. If $X, Y \in \{0,1\}^*$ are strings then $X \,\|\, Y$, or simply XY, denote their concatenation. If $X \in \{0,1\}^*$ is a string then $|X|$ denotes its length in bits. Throughout the paper we fix the *block size* n. Typical values of n are 64 and 128. We let $\{0,1\}^n$ be a set of all bit strings of n bits. For a string $X \in \{0,1\}^{n\ell}$ with $\ell \geq 1$, the *partition* $X[1] \cdots X[\ell]$ of X are defined as the unique strings satisfying the conditions $X[1] \,\|\, \cdots \,\|\, X[\ell] = X$ and $|X[1]| = \cdots = |X[\ell]| = n$. We write $X[1] \cdots X[\ell] \xleftarrow{n} X$. For an n-bit string $X = X_n \cdots X_2 X_1 \in \{0,1\}^n$, $X \ll 1 = X_{n-1} \cdots X_2 X_1 0$ is the

Table 1. Summary of the results. All results are in the presence of the key check value. The figures in the "known" column indicate the required number of known message and tag pairs, and "chosen" indicates the number of chosen message and tag pairs.

MAC	Existential forgery			Selective forgery			Security bound
	Known	Chosen	Reference	Known	Chosen	Reference	[Sect. 6]
MAC1.1	1	0	folklore	1	1	folklore	—
MAC1.2	$O(2^{n/2})$	1	[14,27]	—	—		$O(\sigma^2/2^n)$
MAC2.1	1	$O(2^{(n-s)/2})$	[Sect. 5]	0	$O(2^{(n-s)/2})$	[Sect. 5]	$O(\sigma^2/2^{n-s})$
MAC2.2	1	$O(2^{(n-s)/2})$	[Sect. 5]	0	$O(2^{(n-s)/2})$	[Sect. 5]	$O(\sigma^2/2^{n-s})$
MAC3	1	$O(2^{(n-s)/2})$	[Sect. 5]	0	$O(2^{(n-s)/2})$	[Sect. 5]	$O(\sigma^2/2^{n-s})$
MAC4.1	$O(2^{n/2})$	1	[14,27]	—	—		$O(\sigma^2/2^n)$
MAC4.2	$O(2^{n/2})$	1	[14,27]	—	—		$O(\sigma^2/2^n)$
MAC5	1	$O(2^{(n-s)/2})$	[Sect. 5]	0	$O(2^{(n-s)/2})$	[Sect. 5]	$O(\sigma^2/2^{n-s})$
MAC6.1	$O(2^{n/2})$	1	[14,27]	—	—		$O(\sigma^2/2^n)$
MAC6.2	1	$O(2^{(n-s)/2})$	[Sect. 5]	0	$O(2^{(n-s)/2})$	[Sect. 5]	$O(\sigma^2/2^{n-s})$

left shift of X by 1 bit. For a positive integer ℓ and a string X such that $\ell \leq |X|$, $\mathsf{msb}_\ell(X)$ is the leftmost ℓ bits of the string X, and $\mathsf{lsb}_\ell(X)$ is the rightmost ℓ bits of X. For non-negative integers ℓ and n such that $\ell < 2^n$, $\mathsf{bin}_n(\ell)$ is an n-bit binary representation of ℓ.

A blockcipher is a family of permutations. We write $E : \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n$ for a blockcipher, where $K \in \{0,1\}^k$ is a key and $E_K(\cdot) = E(K,\cdot)$ is the permutation over $\{0,1\}^n$ specified by K. We write $\mathrm{Perm}(n)$ for the set of all permutations over $\{0,1\}^n$, and $\mathrm{Rand}(n)$ for the set of all functions from $\{0,1\}^n$ to $\{0,1\}^n$.

A MAC is a keyed function $\mathcal{M}_K$. It takes an input message $M \in \{0,1\}^*$ and outputs a tag $T \in \{0,1\}^\tau$. The value τ is called a tag length, and we consider the case $\tau = n$. An adversary $\mathcal{A}$ against the MAC is an algorithm that has access to the MAC oracle, $\mathcal{M}_K$, and outputs a *forgery*, which is a pair of a message and a tag, (M^*, T^*). $\mathcal{A}$ can be a probabilistic algorithm or a deterministic algorithm. We say $\mathcal{A}$ *forges* if T^* was not previously returned from the MAC oracle in a response to a query M^*.

3 ISO/IEC 9797-1:2011 MACs

Let $E : \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n$ be a blockcipher. The basic CBC MAC is defined in Fig. 1. It takes a message M as input, where $|M|$ is a positive multiple of n, and returns an n-bit tag T. We write $T \leftarrow \mathrm{CBC}_K(M)$. The specification of the MACs defined in ISO/IEC 9797-1:2011 [14] is in Fig. 2, and illustrated in Fig. 3. The subroutines KD, double, $\mathsf{pad1}$, $\mathsf{pad2}$, and $\mathsf{pad3}$ are specified in Fig. 1. KD is a key derivation function and is used in MAC6.1. double is a doubling operation in $\mathrm{GF}(2^n)$ and is used in MAC5. We note that L is defined as $\mathsf{double}(E_K(0^n))$. $\mathsf{pad1}$, $\mathsf{pad2}$, and $\mathsf{pad3}$ are functions for padding. Our description may seem to be different from the ones in [14] in appearance, but they are carefully distilled for simpler presentation, and they are the same algorithms as specified in [14].

Algorithm $\mathrm{CBC}_K(M)$

1. $Y[0] \leftarrow 0^n$
2. $M[1] \cdots M[m] \xleftarrow{n} M$
3. **for** $i \leftarrow 1$ **to** m **do**
4. $\quad X[i] \leftarrow Y[i-1] \oplus M[i]$
5. $\quad Y[i] \leftarrow E_K(X[i])$
6. $T \leftarrow Y[m]$
7. **return** T

Subroutine $\mathsf{KD}(K)$

1. $\ell \leftarrow \lceil k/n \rceil$
2. **for** $i \leftarrow 1$ **to** ℓ **do**
3. $\quad K'[i] \leftarrow E_K(\mathsf{bin}_n(i))$
4. $\quad K''[i] \leftarrow E_K(\mathsf{bin}_n(\ell + i))$
5. $K' \leftarrow \mathsf{msb}_k(K'[1] \cdots K'[\ell])$
6. $K'' \leftarrow \mathsf{msb}_k(K''[1] \cdots K''[\ell])$
7. **return** (K', K'')

Subroutine $\mathsf{double}(L)$

1. **if** $\mathsf{msb}_1(L) = 0$ **then**
2. $\quad L \leftarrow L \ll 1$
3. **else**
4. $\quad L \leftarrow (L \ll 1) \oplus \mathsf{constant}_n$
5. **return** L

Subroutine $\mathsf{pad1}(M)$

1. $M \leftarrow M \,\|\, 1 \,\|\, 0^{n-1-(|M| \bmod n)}$
2. **return** M

Subroutine $\mathsf{pad2}(M)$

1. **if** $(|M| > 0) \wedge (|M| \bmod n = 0)$ **then**
2. $\quad M \leftarrow \mathsf{bin}_n(|M|) \| M$
3. **else**
4. $\quad M \leftarrow \mathsf{bin}_n(|M|) \| M \| 0^{n-(|M| \bmod n)}$
5. **return** M

Subroutine $\mathsf{pad3}(L, M)$

1. **if** $(|M| = 0) \vee (|M| \bmod n > 0)$ **then**
2. $\quad M \leftarrow \mathsf{pad1}(M)$
3. $\quad L \leftarrow \mathsf{double}(L)$
4. $M[1] \cdots M[m] \xleftarrow{n} M$
5. $M[m] \leftarrow M[m] \oplus L$
6. **return** $M[1] \cdots M[m]$

Fig. 1. Pseudocode of the basic CBC MAC, and the subroutines used in the definition of MACs in ISO/IEC 9797-1. In $\mathsf{KD}(K)$, $\ell + i$ is the arithmetic addition of ℓ and i. In $\mathsf{double}(L)$, $\mathsf{constant}_n$ is an n-bit constant that depends on n. For example, $\mathsf{constant}_{64} = \mathtt{0x0} \cdots \mathtt{01b}$ and $\mathsf{constant}_{128} = \mathtt{0x0} \cdots \mathtt{087}$. When M is the empty string, we have $\mathsf{pad2}(M) = 0^{2n}$.

A total of six MACs are specified in [14]. The ten MACs in Fig. 2 are taken from [14, Annex C, Table C.1], which are specified as the concrete choices of the final iteration, the output transformation, and the padding method. For all MACs except for MAC4.1, they take a message $M \in \{0,1\}^*$ as their input, while MAC4.1 takes a message M such that $|M| \geq n$ as input. All these ten MACs return an n-bit tag T (we only consider the case where the tag consists of a full n-bit string). The key derivation method is not specified in [14, Annex C, Table C.1]. In MAC2.1, following the examples in [14, Annex B.3], we let $K' \leftarrow K \oplus (\mathtt{0xf0f0} \cdots \mathtt{f0})$. We also have a similar key derivation in MAC4.1 and MAC4.2, and in these algorithms, $k = |K|$ is assumed to be a multiple of 8. In MAC6.1, following [14, Annex B.7, NOTE 2 in Sect. 7.7], we let $(K', K'') \leftarrow \mathsf{KD}(K)$.

4 ANSI X9.24:2009 Key Check Value

In [2], the key check value is defined as follows.

Algorithm $\mathrm{MAC1.1}_K(M)$

1. $M \leftarrow \mathsf{pad1}(M)$
2. $T \leftarrow \mathrm{CBC}_K(M)$
3. **return** T

Algorithm $\mathrm{MAC1.2}_K(M)$

1. $M \leftarrow \mathsf{pad2}(M)$
2. $T \leftarrow \mathrm{CBC}_K(M)$
3. **return** T

Algorithm $\mathrm{MAC2.1}_K(M)$

1. $K' \leftarrow K \oplus (\mathtt{0xf0f0} \cdots \mathtt{f0})$
2. $M \leftarrow \mathsf{pad1}(M)$
3. $S \leftarrow \mathrm{CBC}_K(M)$
4. $T \leftarrow E_{K'}(S)$
5. **return** T

Algorithm $\mathrm{MAC2.2}_{K,K'}(M)$

1. $M \leftarrow \mathsf{pad1}(M)$
2. $S \leftarrow \mathrm{CBC}_K(M)$
3. $T \leftarrow E_{K'}(S)$
4. **return** T

Algorithm $\mathrm{MAC3}_{K,K'}(M)$

1. $M \leftarrow \mathsf{pad1}(M)$
2. $S \leftarrow \mathrm{CBC}_K(M)$
3. $T \leftarrow E_K(E_{K'}^{-1}(S))$
4. **return** T

Algorithm $\mathrm{MAC4.1}_{K,K'}(M)$

1. $K'' \leftarrow K' \oplus (\mathtt{0xf0f0} \cdots \mathtt{f0})$
2. $M[1] \cdots M[m] \xleftarrow{n} \mathsf{pad1}(M)$
3. $M[2] \leftarrow E_{K''}(E_K(M[1])) \oplus M[2]$
4. $S \leftarrow \mathrm{CBC}_K(M[2] \cdots M[m])$
5. $T \leftarrow E_{K'}(S)$
6. **return** T

Algorithm $\mathrm{MAC4.2}_{K,K'}(M)$

1. $K'' \leftarrow K' \oplus (\mathtt{0xf0f0} \cdots \mathtt{f0})$
2. $M[1] \cdots M[m] \xleftarrow{n} \mathsf{pad2}(M)$
3. $M[2] \leftarrow E_{K''}(E_K(M[1])) \oplus M[2]$
4. $S \leftarrow \mathrm{CBC}_K(M[2] \cdots M[m])$
5. $T \leftarrow E_{K'}(S)$
6. **return** T

Algorithm $\mathrm{MAC5}_K(M)$

1. $L \leftarrow \mathsf{double}(E_K(0^n))$
2. $M \leftarrow \mathsf{pad3}(L, M)$
3. $T \leftarrow \mathrm{CBC}_K(M)$
4. **return** T

Algorithm $\mathrm{MAC6.1}_K(M)$

1. $(K', K'') \leftarrow \mathsf{KD}(K)$
2. $M[1] \cdots M[m] \xleftarrow{n} \mathsf{pad1}(M)$
3. $S \leftarrow 0^n$
4. **if** $m \geq 2$ **then**
5. $\quad S \leftarrow \mathrm{CBC}_{K'}(M[1] \cdots M[m-1])$
6. $T \leftarrow E_{K''}(S \oplus M[m])$
7. **return** T

Algorithm $\mathrm{MAC6.2}_{K,K'}(M)$

1. $M[1] \cdots M[m] \xleftarrow{n} \mathsf{pad1}(M)$
2. $S \leftarrow 0^n$
3. **if** $m \geq 2$ **then**
4. $\quad S \leftarrow \mathrm{CBC}_K(M[1] \cdots M[m-1])$
5. $T \leftarrow E_{K'}(S \oplus M[m])$
6. **return** T

Fig. 2. Pseudocode of the ISO/IEC 9797-1 MACs

"The optional check values, as mentioned in notes 2 and 3 above, are the left-most six hexadecimal digits from the ciphertext produced by using the DEA in ECB mode to encrypt to 64-bit binary zero value with the subject key or key component. The check value process may be simplified operationally, while still retaining reliability, by limiting the check value to the left-most four or six hexadecimal digits of the ciphertext. (Using the truncated check value may provide additional security in that the ciphertext which could be used for exhaustive key determination would be unavailable.)"

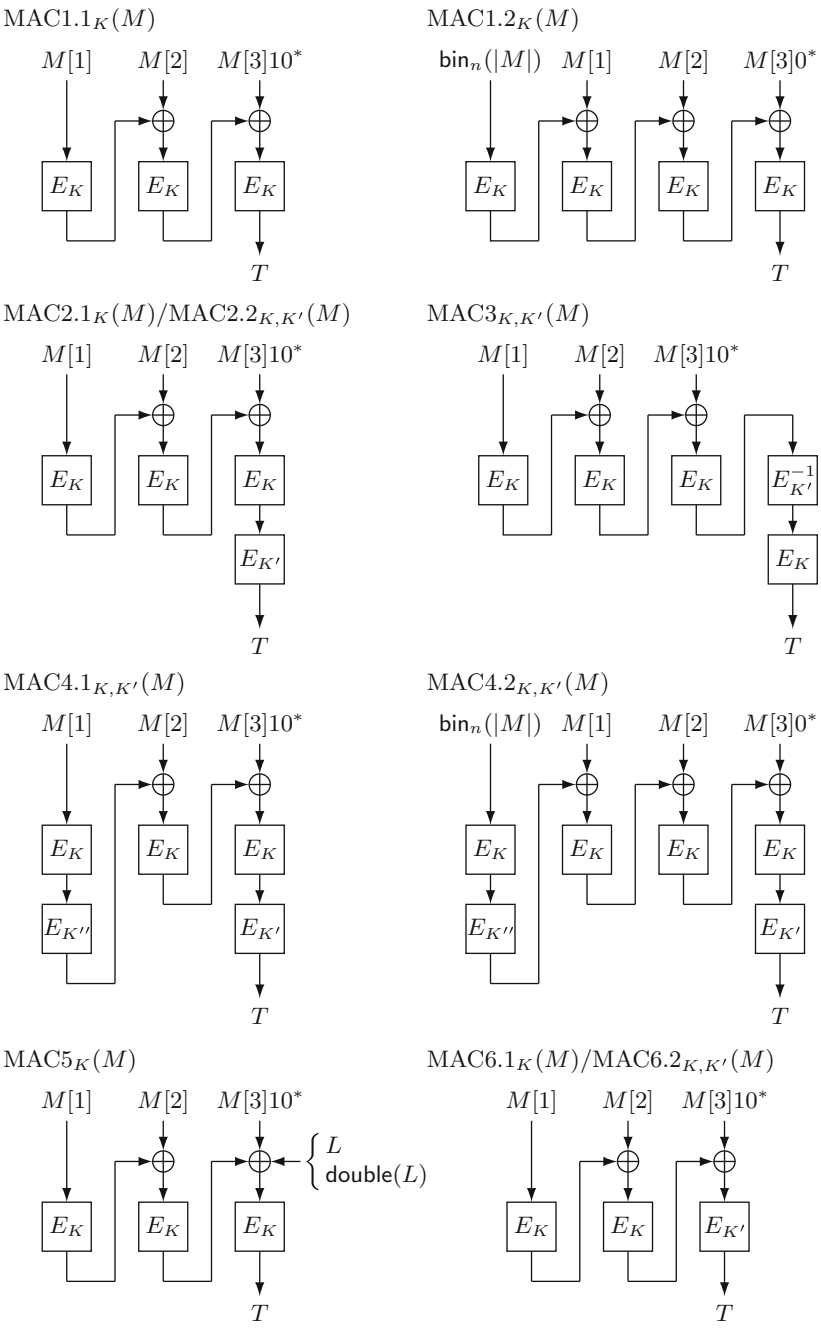


Fig. 3. Illustrations of the ISO/IEC 9797-1 MACs for $M = M[1]M[2]M[3]$, where $|M[1]| = |M[2]| = n$ and $1 \leq |M[3]| \leq n - 1$. In MAC6.1, $(E_K, E_{K'})$ is replaced with $(E_{K'}, E_{K''})$.

The key check value for the 56-bit key K is thus $\mathsf{msb}_{24}(\mathrm{DES}_K(0^{64}))$ or $\mathsf{msb}_{16}(\mathrm{DES}_K(0^{64}))$. The key check value is used to verify the integrity of K or as the ID for K in financial services including banking systems. The value is inherently a public value for verification, and it may be transmitted, sent or stored in clear, which implies that an adversary can learn this value.

In [2], the key check value is defined for DES or TDES, 64-bit blockciphers. However, other documents do not limit the block size being 64 bits. For example, the key check value of AES, a 128-bit blockcipher, is mentioned in [23]. MACs in [14] can be used with AES, and the document gives a warning about the use of the key check value. A similar warning can be found in [11,12], where AES can be used. Although it is not clear how the key check value is defined for AES in these documents, in this paper, for generality, we naturally extend the definition to any blockcipher E and allow other lengths of the key check value. For a blockcipher $E_K : \{0,1\}^n \to \{0,1\}^n$ with key $K \in \{0,1\}^k$ and an integer s such that $0 \le s \le n$, we define the key check value, KCV, as

$$\mathsf{KCV} = \mathsf{msb}_s(E_K(0^n)).$$

We leave s as a parameter that can be defined by a user of the blockcipher.

Now suppose that a MAC internally uses the blockcipher E and the key space of the MAC is $(\{0,1\}^k)^w$ for some integer $w \ge 1$. That is, a total of w blockcipher keys $K_1, \ldots, K_w \in \{0,1\}^k$ are used in the MAC, and it is built from $E_{K_1}, \ldots, E_{K_w}$. Then we define the key check value of the MAC as

$$\mathsf{KCV} = (\mathsf{msb}_s(E_{K_1}(0^n)), \ldots, \mathsf{msb}_s(E_{K_w}(0^n))).$$

Specifically, if the MAC uses single blockcipher key K, which corresponds to MAC1.1, MAC1.2, MAC2.1, MAC5, and MAC6.1, then the adversary is given $\mathsf{KCV} = \mathsf{msb}_s(E_K(0^n))$. For the remaining MACs that use two blockcipher keys K and K', i.e., for MAC2.2, MAC3, MAC4.1, MAC4.2, and MAC6.2, the adversary is given $\mathsf{KCV} = (\mathsf{msb}_s(E_K(0^n)), \mathsf{msb}_s(E_{K'}(0^n)))$.

5 Security Analysis

5.1 Security Analysis for Case $s = n$

We first consider the most extreme case $s = n$ to illustrate the impact.

Existential Forgery Against MAC2.1. Let $\mathsf{KCV} = \mathsf{msb}_s(E_K(0^n))$ be the key check value given to the adversary $\mathcal{A}$, where $s = n$. Let M be any message, and $T = \mathrm{MAC2.1}_K(M)$ be the corresponding tag. $\mathcal{A}$ is given the known message and tag pair (M, T). If $|M| \ge n$, then $\mathcal{A}$ defines M^* as

$$M^* = 0^n \,\|\, \mathsf{KCV} \,\|\, \cdots \,\|\, \mathsf{KCV} \,\|\, \mathsf{KCV} \oplus \mathsf{msb}_n(M) \,\|\, \mathsf{lsb}_{|M|-n}(M).$$

We see that all these messages, regardless of the number of the intermediate KCV blocks, share the same tag T, and therefore, (M^*, T) is a valid forgery.

We next consider the case $0 \leq |M| < n$. We assume that $\mathsf{pad1}(M) \neq \mathsf{KCV}$. Then there exists some M' such that $0 \leq |M'| < n$ and $\mathsf{pad1}(M') = \mathsf{pad1}(M) \oplus \mathsf{KCV}$. We then define M^* as

$$M^* = 0^n \,\|\, \mathsf{KCV} \,\|\, \cdots \,\|\, \mathsf{KCV} \,\|\, M', \tag{1}$$

and it can be easily verified that (M^*, T) is a valid forgery. If $\mathsf{pad1}(M) = \mathsf{KCV}$, then M' in (1), and hence M^*, cannot be defined and the attack does not work. However, there is at most one such M, and thus $\mathcal{A}$ can simply ask for the second known message and tag pair to mount the attack.

Therefore, MAC2.1 allows existential forgeries if the adversary knows any message and tag pair (M, T), where $\mathsf{pad1}(M) \neq \mathsf{KCV}$.

Selective Forgery Against MAC2.1. Almost the same idea can be used for a selective forgery against MAC2.1. Let M^* be a message that $\mathcal{A}$ tries to forge. We show that $\mathcal{A}$ is able to obtain the correct tag for M^* with one chosen message and tag pair, provided that $\mathsf{pad1}(M^*) \neq \mathsf{KCV}$. Now $\mathcal{A}$ defines M as

$$M = \begin{cases} 0^n \,\|\, \mathsf{KCV} \oplus \mathsf{msb}_n(M^*) \,\|\, \mathsf{lsb}_{|M^*|-n}(M^*) & \text{if } |M^*| \geq n, \\ 0^n \,\|\, M' & \text{else,} \end{cases}$$

where M' is a string that satisfies $0 \leq |M'| < n$ and $\mathsf{pad1}(M') = \mathsf{pad1}(M^*) \oplus \mathsf{KCV}$. Then $\mathcal{A}$ asks M to its MAC oracle and obtains $T = \mathrm{MAC2.1}_K(M)$. We see that (M^*, T) is a valid forgery.

We remark that if $\mathsf{pad1}(M^*) = \mathsf{KCV}$, then this particular M^* does not seem to allow attacks.

Existential/Selective Forgeries Against MAC5. As in the case for MAC2.1, $\mathcal{A}$ is given $\mathsf{KCV} = E_K(0^n)$. We see that existential and selective forgeries against MAC2.1 are irrelevant to the operations on the last message block, and almost the same attacks can be applied against MAC5.

Besides these attacks, there are other trivial attacks on MAC5. Recall that $\mathsf{KCV} = E_K(0^n)$ is used to compute the value of L. With KCV, $\mathcal{A}$ can compute both $L = \mathsf{double}(E_K(0^n))$ and $\mathsf{double}(\mathsf{double}(E_K(0^n)))$. This implies that, from $\mathcal{A}$'s view point, MAC5 is essentially reduced to MAC1.1, and therefore almost the same attacks against MAC1.1 in Appendix A work for MAC5.

There are subtle differences due to the padding rule of MAC5, but the modification is rather straightforward and we thus omit the details.

Existential/Selective Forgeries Against Other MACs. We point out that very similar attacks against MAC2.1 can be applied on MAC2.2, MAC3, and MAC6.2. For MAC2.2, $\mathcal{A}$ is given $\mathsf{KCV} = (E_K(0^n), E_{K'}(0^n))$, but the attacks are possible by using only $E_K(0^n)$. For MAC3, we see that the attacks against MAC2.1 are irrelevant to the operations on the last message block, and thus the same attacks can be applied. With the same reasoning these attacks can be applied on MAC6.2.

5.2 Security Analysis for Case $s < n$

We next consider the case $s < n$.

Existential/Selective Forgeries Against MAC2.1. The adversary $\mathcal{A}$ has access to the MAC2.1$_K(\cdot)$ oracle. Let $\mathsf{KCV} = \mathsf{msb}_s(E_K(0^n))$ be the key check value given to $\mathcal{A}$. We first derive the value of $E_K(0^n)$.

Let $r = 2^{(n-s)/2}$. $\mathcal{A}$ first chooses r random strings $\mathsf{rand}_1, \ldots, \mathsf{rand}_r$, where $|\mathsf{rand}_i| = n - s$ for all $1 \le i \le r$ and $\mathsf{rand}_i \ne \mathsf{rand}_j$ for all $1 \le i < j \le r$. Similarly, $\mathcal{A}$ chooses r random strings $\mathsf{rand}'_1, \ldots, \mathsf{rand}'_r$, where $|\mathsf{rand}'_i| = n - s$ for all $1 \le i \le r$ and $\mathsf{rand}'_i \ne \mathsf{rand}'_j$ for all $1 \le i < j \le r$. Let $M_i = 0^n \,\|\, (0^s \,\|\, \mathsf{rand}_i)$ and $M'_i = (\mathsf{KCV} \,\|\, \mathsf{rand}'_i)$. $\mathcal{A}$ then makes $2r$ queries, $M_1, \ldots, M_r, M'_1, \ldots, M'_r$, to its oracle and obtains $T_1, \ldots, T_r, T'_1, \ldots, T'_r$, where $T_i = \text{MAC2.1}_K(M_i)$ and $T'_i = \text{MAC2.1}_K(M'_i)$.

Then it is easy to see that we have $\mathsf{pad1}(M_i) = 0^n \,\|\, (0^s \,\|\, \mathsf{rand}_i) \,\|\, 10^{n-1}$ and $\mathsf{pad1}(M'_i) = (\mathsf{KCV} \,\|\, \mathsf{rand}'_i) \,\|\, 10^{n-1}$. Let $(X_i[1], X_i[2], X_i[3])$ be the input sequence of E_K in the computation of MAC2.1$_K(M_i)$, and $(Y_i[1], Y_i[2], Y_i[3])$ be the corresponding output sequence. Similarly, let $(X'_i[1], X'_i[2])$ and $(Y'_i[1], Y'_i[2])$ be the input and output sequences of E_K in the computation of MAC2.1$_K(M'_i)$. We have

$$\begin{cases} X_i[1] = 0^n, X_i[2] = Y_i[1] \oplus (0^s \,\|\, \mathsf{rand}_i), X_i[3] = Y_i[2] \oplus 10^{n-1}, \\ Y_i[1] = E_K(X_i[1]), Y_i[2] = E_K(X_i[2]), Y_i[3] = E_K(X_i[3]). \end{cases}$$

Similarly, we have

$$\begin{cases} X'_i[1] = (\mathsf{KCV} \,\|\, \mathsf{rand}'_i), X'_i[2] = Y'_i[1] \oplus 10^{n-1}, \\ Y'_i[1] = E_K(X'_i[1]), Y'_i[2] = E_K(X'_i[2]). \end{cases}$$

Note that $Y_i[3] = E_K(X_i[3]) = S_i$ and $Y'_i[2] = E_K(X'_i[2]) = S'_i$. We also note that $E_{K'}(S_i) = T_i$ and $E_{K'}(S'_i) = T'_i$ hold.

We claim that, with a high probability, there exists a pair of indices (j, j') such that $T_j = T'_{j'}$. To see this, we have $T_j = T'_{j'}$ if and only if $X_j[3] = X'_{j'}[2]$ since E_K and $E_{K'}$ are permutations. Now $X_j[3] = X'_{j'}[2]$ holds if and only if $Y_j[2] = Y'_{j'}[1]$ since the same value, 10^{n-1}, is xor-ed. Then $Y_j[2] = Y'_{j'}[1]$ holds if and only if $X_j[2] = X'_{j'}[1]$ from the invertibility of E_K, and this is equivalent to $E_K(0^n) \oplus (0^s \,\|\, \mathsf{rand}_j) = (\mathsf{KCV} \,\|\, \mathsf{rand}'_{j'})$. Now the last condition is equivalent to $\mathsf{lsb}_{n-s}(E_K(0^n)) \oplus \mathsf{rand}_j = \mathsf{rand}'_{j'}$, since $\mathsf{KCV} = \mathsf{msb}_s(E_K(0^n))$, and from the standard birthday paradox, we have the claim.

Let (j, j') be the pair of indices such that $T_j = T'_{j'}$. Observe that $\mathcal{A}$ can now retrieve the value of $E_K(0^n)$ since we have $E_K(0^n) = (\mathsf{KCV} \,\|\, \mathsf{rand}_j \oplus \mathsf{rand}'_{j'})$. With the knowledge of $E_K(0^n)$, $\mathcal{A}$ can produce arbitrarily number of existential forgeries with one known message and tag pair, and can produce a selective forgery with one chosen message and tag pair, as described in Sect. 5.1. Therefore, MAC2.1 allows existential forgeries with $2 \cdot 2^{(n-s)/2}$ chosen messages and one known message, and it allows selective forgery with $1 + 2 \cdot 2^{(n-s)/2}$ chosen messages.

Existential/Selective Forgeries Against MAC5. As in the case for MAC2.1, $\mathcal{A}$ is given $\mathsf{KCV} = \mathsf{msb}_s(E_K(0^n))$. We see that exactly the same procedure can be used to retrieve the value of $E_K(0^n)$, which is also used to compute a value of L. Therefore, MAC5 allows existential forgeries with $2 \cdot 2^{(n-s)/2}$ chosen messages and one known message, and it allows selective forgery with $1+2 \cdot 2^{(n-s)/2}$ chosen messages.

Besides these attacks, since L is now known to the adversary, the standard length-extension attack in Appendix A can be used to attack MAC5.

Existential/Selective Forgeries Against Other MACs. We remark that similar attacks can be used against MAC2.2, MAC3, and MAC6.2. These MACs allow existential forgeries with $2 \cdot 2^{(n-s)/2}$ chosen messages and one known message, and they allow selective forgeries with $1 + 2 \cdot 2^{(n-s)/2}$ chosen messages.

We note that the attacks presented in this section cannot be used against MAC1.2, MAC4.1, MAC4.2, and MAC6.1 even if $s = n$.

6 Provable Security Results

We present the provable security results for the nine ISO/IEC 9797-1 MACs.

Security Definition for MACs. Let $\mathcal{M}_{K_1,\ldots,K_w} : \{0,1\}^* \to \{0,1\}^n$ be a MAC with key space $(\{0,1\}^k)^w$ for some $w \geq 1$. An adversary $\mathcal{A}$ is an algorithm that outputs a bit. We consider the following game. First, $\mathcal{A}$ is given the key check value $\mathsf{KCV} = (\mathsf{msb}_s(E_{K_1}(0^n)), \ldots, \mathsf{msb}_s(E_{K_w}(0^n)))$. Then $\mathcal{A}$ is given access to an oracle, which is either the MAC oracle or the ideal random oracle. The MAC oracle $\mathcal{M}_{K_1,\ldots,K_w}$ takes a message M as the input and returns $T = \mathcal{M}_{K_1,\ldots,K_w}(M)$. The random oracle $\mathcal{R}$ takes a message M to return a random string T. We define

$$\mathbf{Adv}^{\text{prf-kcv}}_{\mathcal{M}}(\mathcal{A}) = \Pr\left[\mathcal{A} \leftarrow \mathsf{KCV}, \mathcal{A}^{\mathcal{M}_{K_1,\ldots,K_w}(\cdot)} \Rightarrow 1\right] - \Pr\left[\mathcal{A} \leftarrow \mathsf{KCV}, \mathcal{A}^{\mathcal{R}(\cdot)} \Rightarrow 1\right],$$

where the first probability is taken over the choices of $K_1, \ldots, K_w$ and $\mathcal{A}$'s coin, and the last is over the choices of $K_1, \ldots, K_w$ used for KCV, the random oracle, and $\mathcal{A}$'s coin.

Specifically, if $\mathcal{M}$ uses single blockcipher key K, i.e., if $\mathcal{M} \in \{\text{MAC1.1}, \text{MAC1.2}, \text{MAC2.1}, \text{MAC5}, \text{MAC6.1}\}$, then $\mathcal{A}$ is given $\mathsf{KCV} = \mathsf{msb}_s(E_K(0^n))$, and we consider

$$\mathbf{Adv}^{\text{prf-kcv}}_{\mathcal{M}}(\mathcal{A}) = \Pr\left[\mathcal{A} \leftarrow \mathsf{KCV}, \mathcal{A}^{\mathcal{M}_K(\cdot)} \Rightarrow 1\right] - \Pr\left[\mathcal{A} \leftarrow \mathsf{KCV}, \mathcal{A}^{\mathcal{R}(\cdot)} \Rightarrow 1\right].$$

For $\mathcal{M}$ with two blockcipher keys K and K', i.e., for $\mathcal{M} \in \{\text{MAC2.2}, \text{MAC3}, \text{MAC4.1}, \text{MAC4.2}, \text{MAC6.2}\}$, then $\mathsf{KCV} = (\mathsf{msb}_s(E_K(0^n)), \mathsf{msb}_s(E_{K'}(0^n)))$ and we consider

$$\mathbf{Adv}^{\text{prf-kcv}}_{\mathcal{M}}(\mathcal{A}) = \Pr\left[\mathcal{A} \leftarrow \mathsf{KCV}, \mathcal{A}^{\mathcal{M}_{K,K'}(\cdot)} \Rightarrow 1\right] - \Pr\left[\mathcal{A} \leftarrow \mathsf{KCV}, \mathcal{A}^{\mathcal{R}(\cdot)} \Rightarrow 1\right].$$

We write $\mathbf{Adv}^{\text{prf-kcv}}_{\mathcal{M}}(t,q,\sigma) = \max_{\mathcal{A}} \mathbf{Adv}^{\text{prf-kcv}}_{\mathcal{M}}(\mathcal{A})$, where the maximum is taken over adversaries $\mathcal{A}$ whose time complexity, number of queries, and query complexity are at most t, q, and σ, respectively. For the time complexity, we fix a model of computation and a choice of encoding, and it includes the running time and the code size. The query complexity is the total length in blocks of the *padded* queries made to the oracle. For instance if we consider an adversary $\mathcal{A}$ attacking MAC1.2 and if $\mathcal{A}$ makes queries $M_1, \ldots, M_q$, then the query complexity is $\sum_{1 \le i \le q} |\mathsf{pad2}(M_i)|/n$. Without loss of generality, we exclude the trivial queries, and we apply this convention to all adversaries in this paper.

We note that the above definitions capture the security of a MAC as a pseudorandom function, or a PRF, in the presence of KCV. It is well known that PRFs are secure MACs, see e.g. [4].

Security Definition for Blockciphers. We consider three security notions for the underlying blockcipher [5,20]. Let $E_K : \{0,1\}^n \to \{0,1\}^n$ be a blockcipher, E_K^{-1} be its inverse, $P, P' : \{0,1\}^n \to \{0,1\}^n$ be two independent random permutations, and P^{-1} be the inverse of P. An adversary $\mathcal{A}$ is an algorithm that outputs a bit. For $\mathcal{A}$, we define

$$\mathbf{Adv}^{\text{prp}}_{E}(\mathcal{A}) = \Pr\left[\mathcal{A}^{E_K(\cdot)} \Rightarrow 1\right] - \Pr\left[\mathcal{A}^{P(\cdot)} \Rightarrow 1\right],$$
$$\mathbf{Adv}^{\text{sprp}}_{E}(\mathcal{A}) = \Pr\left[\mathcal{A}^{E_K(\cdot),E_K^{-1}(\cdot)} \Rightarrow 1\right] - \Pr\left[\mathcal{A}^{P(\cdot),P^{-1}(\cdot)} \Rightarrow 1\right],$$
$$\mathbf{Adv}^{\text{prp-rka}}_{E}(\mathcal{A}) = \Pr\left[\mathcal{A}^{E_K(\cdot),E_{K'}(\cdot)} \Rightarrow 1\right] - \Pr\left[\mathcal{A}^{P(\cdot),P'(\cdot)} \Rightarrow 1\right],$$

where $K' = K \oplus (\texttt{0xf0f0}\cdots\texttt{f0})$. The last one is a particular form of related key attacks [5]. We fix a model of computation and a choice of encoding, and write $\mathbf{Adv}^{\text{prp}}_{E}(t,\sigma) = \max_{\mathcal{A}} \mathbf{Adv}^{\text{prp}}_{E}(\mathcal{A})$, $\mathbf{Adv}^{\text{sprp}}_{E}(t,\sigma) = \max_{\mathcal{A}} \mathbf{Adv}^{\text{sprp}}_{E}(\mathcal{A})$, and $\mathbf{Adv}^{\text{prp-rka}}_{E}(t,\sigma) = \max_{\mathcal{A}} \mathbf{Adv}^{\text{prp-rka}}_{E}(\mathcal{A})$, where the maximum is taken over adversaries $\mathcal{A}$ whose time complexity is at most t and whose query complexity is at most σ. The query complexity is the total number of queries made to the oracles.

Theorem Statement. Let $\mathcal{M}[E]$ be a MAC $\mathcal{M}$, where $E : \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n$ is used as the underlying blockcipher. We have the following result.

Theorem 1. *Fix t, q, and σ, where $q, \sigma \ge 1$. Then the following bounds hold.*

$$\mathbf{Adv}^{\text{prf-kcv}}_{\text{MAC1.2}[E]}(t,q,\sigma) \le \mathbf{Adv}^{\text{prp}}_{E}(t', \sigma+1) + n/2^{n/2} + 7.5\sigma^2/2^n, \tag{2}$$
$$\mathbf{Adv}^{\text{prf-kcv}}_{\text{MAC2.1}[E]}(t,q,\sigma) \le \mathbf{Adv}^{\text{prp-rka}}_{E}(t', q+\sigma+1) + 3.5\sigma^2/2^{n-s}, \tag{3}$$
$$\mathbf{Adv}^{\text{prf-kcv}}_{\text{MAC2.2}[E]}(t,q,\sigma) \le 2\mathbf{Adv}^{\text{prp}}_{E}(t', \sigma+1) + 8\sigma^2/2^{n-s}, \tag{4}$$
$$\mathbf{Adv}^{\text{prf-kcv}}_{\text{MAC3}[E]}(t,q,\sigma) \le 2\mathbf{Adv}^{\text{sprp}}_{E}(t', q+\sigma+1) + 23.5\sigma^2/2^{n-s}, \tag{5}$$
$$\mathbf{Adv}^{\text{prf-kcv}}_{\text{MAC4.1}[E]}(t,q,\sigma) \le 2\mathbf{Adv}^{\text{prp-rka}}_{E}(t', 2\sigma+1) + 11.5\sigma^2/2^n, \tag{6}$$

$$\mathbf{Adv}^{\text{prf-kcv}}_{\text{MAC4.2}[E]}(t,q,\sigma) \le 2\mathbf{Adv}^{\text{prp-rka}}_{E}(t',2\sigma+1) + 11.5\sigma^2/2^n, \tag{7}$$

$$\mathbf{Adv}^{\text{prf-kcv}}_{\text{MAC5}[E]}(t,q,\sigma) \le \mathbf{Adv}^{\text{prp}}_{E}(t',\sigma+1) + 5\sigma^2/2^{n-s}, \tag{8}$$

$$\mathbf{Adv}^{\text{prf-kcv}}_{\text{MAC6.1}[E]}(t,q,\sigma) \le 2\mathbf{Adv}^{\text{prp}}_{E}(t',\sigma+1) + \mathbf{Adv}^{\text{prp}}_{E}(t'',2\ell+1) + 8\sigma^2/2^n + 4.5\ell^2/2^n, \tag{9}$$

$$\mathbf{Adv}^{\text{prf-kcv}}_{\text{MAC6.2}[E]}(t,q,\sigma) \le 2\mathbf{Adv}^{\text{prp}}_{E}(t',\sigma+1) + 8\sigma^2/2^{n-s}, \tag{10}$$

where $t' = t + O(\sigma)$, $t'' = t + O(\ell+\sigma)$, *and* $\ell = \lceil k/n \rceil$.

A proof overview is presented in Sect. 7, and the proof is presented in [17].

Discussions. For the assumption about the underlying blockcipher, we require the security against related key attacks for MAC2.1, MAC4.1, and MAC4.2. These are the MACs that use $K' = K \oplus (\texttt{0xf0f0}\cdots\texttt{f0})$. MAC3 is the only MAC that requires the strong pseudorandomness assumption, as it uses the inverse of the blockcipher. The remaining MACs, MAC1.2, MAC2.2, MAC5, MAC6.1, and MAC6.2, need the standard pseudorandomness assumption.

For the security bound, we see that MAC1.2, MAC4.1, MAC4.2, and MAC6.1 have the standard birthday bound that does not depend on s. This implies that the security bounds for these MACs remain unchanged even if the key check value consists of the entire n bits. Other MACs, MAC2.1, MAC2.2, MAC3, MAC5, and MAC6.2, have the security loss by $s/2$ bits. With respect to the term $n/2^{n/2}$ in MAC1.2, we do not know if it can be removed, but there is an attack with the suggested success probability with two queries and the birthday query complexity. See Appendix B. We also note that the use of ℓ in MAC6.1 comes from the use of the key derivation function.

We argue that, if s stays relatively small as specified in [2], depending on applications, these MACs can still be used in practice. See Table 2 for the expected number of blocks of queries to attack these MACs.

7 Proof Overview of Theorem 1

Although nine MACs in Theorem 1 share the same basic structure of CBC MAC, the security proofs are different in details. Our proof of Theorem 1 can be divided into five cases, MAC1.2, MAC2.1, MAC3, MAC4.1, and MAC5. The proofs for MAC2.2, MAC6.1, and MAC6.2 are similar to that of MAC2.1. The proof for MAC4.2 follows from that of MAC4.1. The first step is to replace the blockcipher with a random permutation. This will introduce $\mathbf{Adv}^{\text{prp}}_{E}(t',\sigma')$, $\mathbf{Adv}^{\text{prp-rka}}_{E}(t',\sigma')$, or $\mathbf{Adv}^{\text{sprp}}_{E}(t',\sigma')$ depending on the usage of the underlying blockcipher. We then replace the random permutation with a random function. This will introduce a term $O(\sigma^2/2^n)$. The rest of the proofs are different depending on the MACs.

Table 2. Required number of blocks of queries to mount attacks against MAC2.1, MAC2.2, MAC3, MAC5, and MAC6.2

$n = 64$			$n = 128$				
$s = 0$	$s = 16$	$s = 24$	$s = 0$	$s = 16$	$s = 24$	$s = 32$	$s = 48$
2^{32}	2^{24}	2^{20}	2^{64}	2^{56}	2^{52}	2^{48}	2^{40}

Case MAC1.2. The analysis of MAC1.2 is quite involved. We define a number of oracles. Let M be an ℓ-bit message. After applying the padding, we have $M[1] \cdots M[m] \xleftarrow{n} \mathsf{pad2}(M)$, where $m = \lceil \ell/n \rceil + 1$. Then we define an oracle that is only used to encrypt the j-th block $M[j]$. That is, our oracles are parameterized by ℓ and j, and a specific oracle is used only for encrypting the j-th block of an ℓ-bit message M. By doing so, we eliminate the interaction between KCV and the MAC computation part, except for a rare case of computing a tag for the empty string. We proceed by showing that MAC1.2, instantiated with such oracles, is indistinguishable from the MAC1.2 that is based on a single random function. We then show that CBC MAC that uses independent random functions for every block is indistinguishable from a random function.

Case MAC2.1, MAC2.2, MAC6.1, and MAC6.2. For MAC2.1, we make use of the following lemma, where $\mathrm{CBC}_F(M)$ is the CBC MAC value of M where a random function $F : \{0,1\}^n \to \{0,1\}^n$ is used as the underlying blockcipher.

Lemma 1. *For any* $\mathsf{KCV} \in \{0,1\}^s$, $M \in \{0,1\}^{mn}$, *and* $M' \in \{0,1\}^{m'n}$, *where* $m, m' \geq 1$ *and* $M \neq M'$, *we have*

$$\Pr[\mathrm{CBC}_F(M) = \mathrm{CBC}_F(M') \mid \mathsf{msb}_s(F(0^n)) = \mathsf{KCV}] \leq \frac{mm' + \max\{m, m'\}}{2^{n-s}}.$$

Proof. To simplify the notation, let E_1 be the event $\mathrm{CBC}_F(M) = \mathrm{CBC}_F(M')$. Similarly, let E_2 be the event $\mathsf{msb}_s(F(0^n)) = \mathsf{KCV}$. Now [8, Lemma 3] shows that $\Pr[\mathsf{E}_1] \leq (mm' + \max\{m, m'\})/2^n$. We also have $\Pr[\mathsf{E}_2] = 1/2^s$. We have the claimed bound from the two probabilities and the Bayes' theorem as

$$\Pr[\mathsf{E}_1 \mid \mathsf{E}_2] = \frac{\Pr[\mathsf{E}_1 \wedge \mathsf{E}_2]}{\Pr[\mathsf{E}_2]} \leq \frac{\Pr[\mathsf{E}_1]}{\Pr[\mathsf{E}_2]} \leq \frac{mm' + \max\{m, m'\}}{2^{n-s}},$$

and this completes the proof. □

The proof of MAC2.1 is obtained by bounding the probability that we have a collision at the input of the last random function, which can be derived by using Lemma 1. We use the following lemma in the proof of MAC2.2.

Lemma 2. *For any* $\mathsf{KCV} \in \{0,1\}^s$, $\mathsf{constant} \in \{0,1\}^n$, *and* $M \in \{0,1\}^{mn}$, *where* $m \geq 1$, *we have*

$$\Pr[\mathrm{CBC}_F(M) = \mathsf{constant} \mid \mathsf{msb}_s(F(0^n)) = \mathsf{KCV}] \leq \frac{2(m+1)}{2^{n-s}}.$$

Proof. Let $\tilde{M} \leftarrow M \,\|\, \mathsf{constant}$ and $\tilde{M'} \leftarrow 0^n$. We see that if $\mathrm{CBC}_F(M) = \mathsf{constant}$ holds, then we have $\mathrm{CBC}_F(\tilde{M}) = \mathrm{CBC}_F(\tilde{M'})$. By applying Lemma 1 to $\tilde{M}$ and $\tilde{M'}$, the upper bound of $\Pr[\mathrm{CBC}_F(M) = \mathsf{constant} \mid \mathsf{msb}_s(F(0^n)) = \mathsf{KCV}]$ is obtained as

$$\Pr[\mathrm{CBC}_F(\tilde{M}) = \mathrm{CBC}_F(\tilde{M'}) \mid \mathsf{msb}_s(F(0^n)) = \mathsf{KCV}] \leq \frac{2(m+1)}{2^{n-s}},$$

which completes the proof. □

The proof of MAC2.2 closely follows that of MAC2.1, and we consider the additional event that we have 0^n at the input of the last random function. We use Lemma 2 to derive a bound on the probability. The proof for MAC6.2 is similar to that of MAC2.2, and the proof of MAC6.1 is obtained by using the result of MAC6.2 without the key check value.

Case MAC3. Let F be a random function that replaces E_K, and P' be a random permutation that replaces $E_{K'}$. Let $Q(X) = F(P'^{-1}(F(X)))$ and G be a random function. The core of the proof of MAC3 lies in proving that three oracles $\mathcal{Q} = (P'(\cdot), F(\cdot), Q(\cdot))$ are indistinguishable from three oracles $\mathcal{G} = (P'(\cdot), F(\cdot), G(\cdot))$. We show this when the domain of the first oracle, P', is restricted to $\{0^n\}$. We only need P' to generate the key check value, and hence it is sufficient for our purpose. We then replace the call of $Q(X)$ for the final block by $G(X)$, and the rest of the proof follows from those of MAC2.2 and MAC6.2.

Case MAC4.1 and MAC4.2. We define five oracles, $\mathcal{Q} = (Q_1(\cdot), \ldots, Q_5(\cdot))$. We use Q_1 and Q_2 to obtain the key check value. For a query M, we let $M[1] \cdots M[m] \xleftarrow{n} \mathsf{pad1}(M)$, and we use Q_3 to encrypt $M[1]$, Q_4 to encrypt blocks that correspond to $M[2], \ldots, M[m-1]$, and Q_5 to encrypt the last block that corresponds to $M[m]$. We show that these oracles can be used to simulate MAC4.1, and we also show that they are indistinguishable from five independent random functions. Therefore, this eliminates the interaction between the key check value and the MAC computation. The rest of the proof is similar to that of MAC2.1, and the proof of MAC4.2 follows from that of MAC4.1.

Case MAC5. For MAC5, we define seven oracles, $\mathcal{Q} = (Q_1(\cdot), \ldots, Q_7(\cdot))$. We use Q_1 for the key check value. Q_2 is used for the first block, Q_3 is used for the middle blocks, and we use four oracles $Q_4, \ldots, Q_7$ for the final block, depending on the length of the input. We show that these oracles can be used to simulate MAC5, and that they are indistinguishable from seven independent random functions. Then the MAC computation becomes independent from the key check value, and the proof follows.

8 Possible Fixes

There are applications where the security loss from using the key check value is not an issue. For instance the key check value may be computed on a master

key, and MACs are computed with session keys that are derived from the master key in a cryptographically strong way, and the master key may never be used to compute the MAC.

For other applications that need to fix the issue of reduction in security, one possible option is to change the specification of the scheme, or the other is to change the definition of the key check value. Since the latter seems to be impractical in view of the long history and the wide spread deployment of the standard, we discuss the former option. We present two generic solutions, meaning that they do not harm the provable security of the mode of operation, and they work for MACs, encryption modes, and authenticated encryption modes.

We can *always* use the key derivation function used in MAC6.1 even when the underlying blockcipher uses n-bit keys. Specifically, consider the case of MAC5, or CMAC, with 128-bit key AES. In this case, $\mathrm{AES}_K(0^n)$ is used as the key check value and $\mathrm{AES}_K(\cdot)$ is used in the actual computation of the tag. Instead, one can use $\mathrm{AES}_K(0^n)$ as the key check value, derive K' as $K' \leftarrow \mathrm{AES}_K(0^{n-1}1)$, and use $\mathrm{AES}_{K'}(\cdot)$ in computing the tag. Under the assumption that AES is a pseudorandom permutation, the key check value and $\mathrm{AES}_{K'}(\cdot)$ are independent, and thus the original security proof of MAC5 carries over.

The above solution introduces an additional key scheduling process, and we present another solution without it. Let K and K' be two independent keys for a blockcipher E. If we use $E_K(0^n)$ to derive the key check value and $E_{K'}(\cdot)$ for the mode of operation, then this clearly does not harm the provable security. Now consider a blockcipher E'_K defined as $E'_K(X) = E_K(X \oplus L) \oplus L$, where $L = E_K(0^{n-1}1)$. Then similarly to XEX construction [28], under the assumption that E is a strong pseudorandom permutation, the pairs of oracles $(E_K(\cdot), E_{K'}(\cdot), E_{K'}^{-1}(\cdot))$ and $(E_K(\cdot), E'_K(\cdot), E'^{-1}_K(\cdot))$ are indistinguishable if the first (leftmost) oracle takes only one value, 0^n, as the input. Therefore, we can use $E_K(0^n)$ to derive the key check value, and use $E'_K(\cdot)$ and $E'^{-1}_K(\cdot)$ for the mode of operation. If the mode of operation is provably secure with the pseudorandomness assumption, we can use E''_K defined as $E''_K(X) = E_K(X \oplus L)$, where $L = E_K(0^{n-1}1)$, instead of E'_K. In this case, similarly to the proof of XE construction [28], $(E_K(\cdot), E_{K'}(\cdot))$ and $(E_K(\cdot), E''_K(\cdot))$ are indistinguishable if the first oracle takes only 0^n as the input. Therefore, we can use $E_K(0^n)$ to derive the key check value, and use $E''_K(\cdot)$ for the mode of operation.

9 Conclusions

We have investigated the use of ANSI X9.24-1 key check value with the MACs specified in ISO/IEC 9797-1. MAC1.1 is widely known to be insecure, and we showed attacks against five MACs, out of nine MACs, by taking advantage of the knowledge of the key check value. We also showed that, for these five MACs, the analysis is tight and the attack cannot be improved. The results suggest that using the key check value *does* result in a security loss by $s/2$ bits, but it does *not* result in a total security loss. This indicates that, depending on the applications and the length of the key check value, they can still be used in practice even in the presence of the key check value, as the security impact is limited as long as s

is not large, say 16 or 24 as suggested in [2]. For the remaining four MACs, the security impact of using the key check value is small, even if the key check value consists of the entire block. We also presented possible ways to fix the issue of the security loss.

It would be interesting to see the impact of the key check value on the security of other blockcipher modes of operation e.g., MAC5 and MAC6 in the 1999 version of ISO/IEC 9797-1 [13], and it would also be interesting to see a more efficient way to fix the issue of the security loss, possibly a solution that depends on the mode of operation. Finally, some stronger security bounds than the standard birthday bound are known for several MACs [24,26], and it would be interesting to see if similar bounds can be obtained in the presence of the key check value.

Acknowledgments. The authors would like to thank Morris Dworkin for informing them of the issue of the key check value on CMAC, which motivated the work. The authors also would like to thank participants of Dagstuhl Seminar 09031 (Symmetric Cryptography), participants of ASK 2011 (The First Asian Workshop on Symmetric Key Cryptography), and the anonymous FSE 2014 reviewers for comments. The work by Tetsu Iwata was carried out in part while visiting Nanyang Technological University, Singapore, and was supported in part by MEXT KAKENHI, Grant-in-Aid for Young Scientists (A), 22680001, and in part by the Naito Science & Engineering Foundation. The work by Lei Wang was supported by the Singapore National Research Foundation Fellowship 2012 (NRF-NRFF2012-06).

References

1. ANSI: Financial Institution Retail Message Authentication (American Bankers Association). ANSI X9.19 (1986)
2. ANSI: Retail Financial Services Symmetric Key Management Part 1: Using Symmetric Techniques. ANSI X9.24-1:2009 (2009)
3. Barker, W.C., Barker, E.: Recommendation for the Triple Data Encryption Algorithm (TDEA) Block Cipher. NIST Special Publication 800-67, Revision 1 (2012)
4. Bellare, M., Kilian, J., Rogaway, P.: The Security of the Cipher Block Chaining Message Authentication Code. J. Comput. Syst. Sci. **61**(3), 362–399 (2000)
5. Bellare, M., Kohno, T.: A Theoretical Treatment of Related-Key Attacks: RKA-PRPs, RKA-PRFs, and Applications. In: Biham, E. (ed.) EUROCRYPT 2003. LNCS, vol. 2656, pp. 491–506. Springer, Heidelberg (2003)
6. Bellare, M., Pietrzak, K., Rogaway, P.: Improved Security Analyses for CBC MACs. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 527–545. Springer, Heidelberg (2005)
7. Black, J., Rogaway, P.: A Block-Cipher Mode of Operation for Parallelizable Message Authentication. In: Knudsen, L.R. (ed.) EUROCRYPT 2002. LNCS, vol. 2332, pp. 384–397. Springer, Heidelberg (2002)
8. Black, J., Rogaway, P.: CBC MACs for Arbitrary-Length Messages: the Three-key Constructions. J. Cryptol. **18**(2), 111–131 (2005)
9. Bosselaers, A., Preneel, B. (eds.): Integrity Primitives for Secure Information Systems. LNCS, vol. 1007. Springer, Heidelberg (1995)
10. Dodis, Y., Pietrzak, K.: Leakage-Resilient Pseudorandom Functions and Side-Channel Attacks on Feistel Networks. In: Rabin, T. (ed.) CRYPTO 2010. LNCS, vol. 6223, pp. 21–40. Springer, Heidelberg (2010)

11. Dworkin, M.: Recommendation for Block Cipher Modes of Operation: The CMAC Mode for Authentication, NIST Special Publication 800-38B (2005)
12. Dworkin, M.: Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC, NIST Special Publication 800-38D (2007)
13. ISO/IEC: Information Technology – Security Techniques – Message Authentication Codes (MACs) – Part 1: Mechanisms Using a Block Cipher. ISO/IEC 9797-1:1999 (1999)
14. ISO/IEC: Information Technology – Security Techniques – Message Authentication Codes (MACs) – Part 1: Mechanisms Using a Block Cipher. ISO/IEC 9797-1:2011 (2011)
15. Iwata, T., Kurosawa, K.: OMAC: One-Key CBC MAC. In: Johansson, T. (ed.) FSE 2003. LNCS, vol. 2887, pp. 129–153. Springer, Heidelberg (2003)
16. Iwata, T., Kurosawa, K.: Stronger Security Bounds for OMAC, TMAC, and XCBC. In: Johansson, T., Maitra, S. (eds.) INDOCRYPT 2003. LNCS, vol. 2904, pp. 402–415. Springer, Heidelberg (2003)
17. Iwata, T., Wang, L.: Impact of ANSI X9.24-1:2009 Key Check Value on ISO/IEC 9797-1:2011 MACs. Cryptology ePrint Archive, Report 2014/183 (2014). Full version of this paper http://eprint.iacr.org/
18. Knudsen, L.: Chosen-Text Attack on CBC-MAC. Electron. Lett. **33**(1), 48–49 (1997)
19. Knudsen, L., Preneel, B.: MacDES: MAC Algorithm Based on DES. Electron. Lett. **34**(9), 871–873 (1998)
20. Luby, M., Rackoff, C.: How to Construct Pseudorandom Permutations from Pseudorandom Functions. SIAM J. Comput. **17**(2), 373–386 (1988)
21. Martin, D., Oswald, E., Stam, M.: A Leakage Resilient MAC. Cryptology ePrint Archive, Report 2013/292 (2013). http://eprint.iacr.org/
22. McGrew, D.A., Viega, J.: The Security and Performance of the Galois/Counter Mode (GCM) of Operation. In: Canteaut, A., Viswanathan, K. (eds.) INDOCRYPT 2004. LNCS, vol. 3348, pp. 343–355. Springer, Heidelberg (2004)
23. Nachtigall, E.: ICSF Verify Key Check Value. IBM Techdocs Library, Doc: PRS4840 (2011)
24. Nandi, M.: Improved Security Analysis for OMAC as a Pseudorandom Function. J. Math. Cryptol. **3**(2), 133–148 (2009)
25. Petrank, E., Rackoff, C.: CBC MAC for Real-Time Data Sources. J. Cryptol. **13**(3), 315–338 (2000)
26. Pietrzak, K.: A Tight Bound for EMAC. In: Bugliesi, M., Preneel, B., Sassone, V., Wegener, I. (eds.) ICALP 2006. LNCS, vol. 4052, pp. 168–179. Springer, Heidelberg (2006)
27. Preneel, B., van Oorschot, P.C.: On the Security of Iterated Message Authentication Codes. IEEE Trans. Inf. Theory **45**(1), 188–199 (1999)
28. 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)
29. Rogaway, P.: Evaluation of Some Blockcipher Modes of Operation. Investigation Reports on Cryptographic Techniques in FY 2010 (2011). http://www.cryptrec.go.jp/english/
30. Rogaway, P., Bellare, M., Black, J.: OCB: A Block-cipher Mode of Operation for Efficient Authenticated Encryption. ACM Trans. Inf. Syst. Secur. **6**(3), 365–403 (2003)

A Attacks Against MAC1.1

It is widely known that MAC1.1 is not secure for variable length messages. These attacks are known as the length-extension attack. We here recall these attacks for completeness.

Existential Forgery Against MAC1.1. Let (M, T) be a known message and tag pair, where $T = \mathrm{MAC1.1}_K(M)$. We show that existential forgeries are possible provided that $\mathsf{pad1}(M) \neq T$ holds. If $|M| \geq n$ then define

$$M^* = \mathsf{pad1}(M) \,\|\, M' \,\|\, \cdots \,\|\, M' \,\|\, T \oplus \mathsf{msb}_n(M) \,\|\, \mathsf{lsb}_{|M|-n}(M),$$

where $M' = T \oplus \mathsf{msb}_n(\mathsf{pad1}(M)) \,\|\, \mathsf{lsb}_{|\mathsf{pad1}(M)|-n}(\mathsf{pad1}(M))$. If $0 \leq |M| < n$ then define

$$M^* = \mathsf{pad1}(M) \,\|\, T \oplus \mathsf{pad1}(M) \,\|\, \cdots \,\|\, T \oplus \mathsf{pad1}(M) \,\|\, M'',$$

where M'' is a string that satisfies $0 \leq |M''| < n$ and $\mathsf{pad1}(M'') = T \oplus \mathsf{pad1}(M)$. We see that T is the correct tag for all M^* defined above.

Selective Forgery Against MAC1.1. Let M^* be the message that A tries to forge. The following selective forgery attack uses one known message and tag pair and one chosen message and tag pair. Let (M_1, T_1) be the known message and tag pair, where $T_1 = \mathrm{MAC1.1}_K(M_1)$. We assume that $M_1 \neq M^*$ and $T_1 \neq \mathsf{pad1}(M^*)$. If $M_1 = M^*$ holds, then the attack fails, and if $T_1 = \mathsf{pad1}(M^*)$, then A can ask for a different known message and tag pair. Let M_2 be

$$M_2 = \begin{cases} \mathsf{pad1}(M_1) \,\|\, T_1 \oplus \mathsf{msb}_n(M^*) \,\|\, \mathsf{lsb}_{|M^*|-n}(M^*) & \text{if } |M^*| \geq n \\ \mathsf{pad1}(M_1) \,\|\, M' & \text{else} \end{cases}$$

where M' is a string that satisfies $0 \leq |M'| < n$ and $\mathsf{pad1}(M') = T_1 \oplus \mathsf{pad1}(M^*)$. Next, A asks M_2 to its MAC oracle and obtains $T_2 = \mathrm{MAC1.1}_K(M_2)$. We see that T_2 is a valid tag for M^*.

B Existential Forgery Against MAC1.2

Let $s = n$. We show an existential forgery against MAC1.2 with a success probability of about $n/2^{n/2}$, and the query complexity of about $2^{n/2}$. Our adversary makes one query to the MAC oracle and one verification query. Now $\mathcal{A}$ is given $\mathsf{KCV} = E_K(0^n)$. Denote the integer representation of KCV as $\mathsf{int}(\mathsf{KCV})$, i.e., $\mathsf{int}(\mathsf{KCV})$ is an integer Z such that $\mathsf{bin}_n(Z) = \mathsf{KCV}$. Let ε be the empty string, and $V = \mathrm{MAC1.2}_K(\varepsilon)$ be the corresponding tag. Note that $\mathsf{pad2}(\varepsilon) = 0^n \| 0^n$ and hence $V = E_K(0^n \oplus E_K(0^n)) = E_K(\mathsf{KCV})$. $\mathcal{A}$ makes a query ε to the MAC1.2 oracle and receives the value of V. Next, $\mathcal{A}$ defines M as

$$M = V \,\|\, \mathsf{KCV} \,\|\, \cdots \,\|\, \mathsf{KCV} \,\|\, 0^m,$$

where $0 < m \leq n$ and $\mathsf{int}(\mathsf{KCV}) = |M|$. Then, it holds that

$$\mathsf{pad2}(M) = \mathsf{KCV} \,\|\, V \,\|\, \mathsf{KCV} \,\|\, \cdots \,\|\, \mathsf{KCV} \,\|\, 0^n.$$

Recall that $V = E_K(\mathsf{KCV})$ and $\mathsf{KCV} = E_K(0^n)$. We see that V is the correct tag for M defined as above.

Now we evaluate the complexity of the above attack. It is dominated by the verification of (M, V), and hence the query complexity is about $\mathsf{int}(\mathsf{KCV})/n$. Since we are interested in attacks with complexity below $2^{n/2}$, it is necessary that we have $\mathsf{int}(\mathsf{KCV}) < n2^{n/2}$, which holds with a probability of $n/2^{n/2}$ as the value of KCV is uniformly random.

Overall this existential forgery attack can be applied to MAC1.2 with a probability of $n/2^{n/2}$.

Stream Ciphers

Plaintext Recovery Attacks Against WPA/TKIP

Kenneth G. Paterson, Bertram Poettering, and Jacob C.N. Schuldt(✉)

Information Security Group, Royal Holloway, University of London, Surrey, UK
jacob.schuldt@rhul.ac.uk

Abstract. We conduct an analysis of the RC4 algorithm as it is used in the IEEE WPA/TKIP wireless standard. In that standard, RC4 keys are computed on a per-frame basis, with specific key bytes being set to known values that depend on 2 bytes of the WPA frame counter (called the TSC). We observe very large, TSC-dependent biases in the RC4 keystream when the algorithm is keyed according to the WPA specification. These biases permit us to mount an effective statistical, plaintext-recovering attack in the situation where the same plaintext is encrypted in many different frames (the so-called "broadcast attack" setting). We assess the practical impact of these attacks on WPA/TKIP.

1 Introduction

The cryptographic mechanisms that aim at protecting transmitted data in modern wireless computer networks have seen an ongoing evolution. Most prominent are the results of the IEEE 802.11 standardization effort; amongst others, IEEE introduced *Wired Equivalent Privacy* (WEP) in 1999, *Wi-Fi Protected Access* (WPA) in 2003, and WPA2 in 2004.

In a nutshell, the WEP protocol [1] works as follows: when a message m is to be transmitted, a CRC32 checksum is appended to it and the resulting string encrypted using RC4; the corresponding packet-specific RC4 key consists of the concatenation of a monotonically increasing sequence number and a shared secret. Practical attacks against integrity, authenticity, and secrecy of transmitted data were reported soon after the publication of WEP, exploiting a wide range of shortcomings of the protocol (including short sequence numbers, lack of randomization, linearity of both RC4 encryption and CRC32, and others) [4]. Refined versions of these attacks [5,19,21] rely on advanced cryptanalysis of RC4 and are based on the fact that the (public) packet sequence number is part of the encryption key. Today, WEP is considered fully broken, and its usage is discouraged even in the IEEE 802.11 standard itself.

To counter these attacks, IEEE decided to redesign the cryptographic components of their wireless standards from scratch. Indeed, the WPA2 standard mandates support for encryption based on the AES block cipher running in CCM mode instead of on RC4 [2]. However, as most wireless devices implement their

This is the proceedings version. The full version can be found at https://eprint.iacr.org/2013/748.

C. Cid and C. Rechberger (Eds.): FSE 2014, LNCS 8540, pp. 325–349, 2015.
DOI: 10.1007/978-3-662-46706-0_17

core cryptographic routines directly in silicon, switching from WEP to WPA2 also requires the replacement of hardware in all involved network computers and access points. In order to mitigate these costs, IEEE additionally proposed the *Temporal Key Integrity Protocol* (TKIP) as part of the WPA standard as an intermediate solution[1]. The design of WPA/TKIP is by intention quite close to that of the original WEP, so that all required modifications can be implemented using only firmware updates. Indeed, WPA/TKIP also encrypts packets using RC4, but with (supposedly) better per-packet keys.

The intention of IEEE was that WPA/TKIP should only be a temporary standard during the transition to WPA2. Indeed, it was recently announced that IEEE will deprecate WPA/TKIP in 2014. Yet use of WPA/TKIP is still widespread. For example, the 2013 paper [23] reported that 71 % of 6803 different IEEE 802.11 networks surveyed still permit WPA/TKIP, with 19 % of those networks using encryption only allowing WPA/TKIP. Given its widespread use, and the spurred on by the WEP fiasco, WPA/TKIP has received a good deal of attention from researchers, including [7,13,14,19,20,22,23].

WPA/TKIP requires the 16-byte RC4 key $\mathtt{K} = (\mathtt{K}_0, \ldots, \mathtt{K}_{15})$ used to encrypt a frame to be generated in a very specific way from the *temporal encryption key* `TK` (128 bits), the TKIP sequence counter `TSC` (48 bits, incremented for each frame that is transmitted), and the transmitter address `TA` (48 bits). Specifically, `K` is computed via a so-called 'key mixing' procedure, which we write as $\mathtt{K} \leftarrow \mathtt{KM}(\mathtt{TA}, \mathtt{TK}, \mathtt{TSC})$. Internally, `KM` implements key derivation by mixing together its inputs using a custom 8-round Feistel cipher whose round function relies on the AES S-boxes. Key `K` is derived from the output of this routine, with some structure added to "preclude the use of known RC4 weak keys" [2]. More precisely, writing $\mathtt{TSC} = (\mathtt{TSC}_0, \mathtt{TSC}_1, \ldots, \mathtt{TSC}_5)$, i.e., the least-significant byte on the left, we have

$$\mathtt{K}_0 = \mathtt{TSC}_1 \qquad \mathtt{K}_1 = (\mathtt{TSC}_1 \mid \mathtt{0x20}) \,\&\, \mathtt{0x7f} \qquad \mathtt{K}_2 = \mathtt{TSC}_0 \tag{1}$$

and $\mathtt{K}_3, \ldots, \mathtt{K}_{15}$ being assigned from the output of the Feistel cipher. Notably here, bytes $\mathtt{K}_0, \mathtt{K}_1, \mathtt{K}_2$ depend only on bytes $\mathtt{TSC}_0$ and $\mathtt{TSC}_1$ of `TSC`. Moreover, the bits of $\mathtt{TSC}_1$ are used twice. So the bytes of `K` have more structure than they would if they were chosen with uniform distribution (as is the case when RC4 is used in TLS, for example); in addition, as `TSC` values are public information, partial information about `K` might be known to attackers.

Recently, AlFardan *et al.* [3] showed that RC4 with uniformly distributed keys (as in TLS) is biased in its initial keystream bytes, and that these biases can be exploited in passive attacks to recover plaintext. More specifically, they worked in the *broadcast* or *multi-session* setting, wherein the same plaintext is repeatedly encrypted under different, independent keys. This setting is readily realised for TLS protecting web traffic by using JavaScript code running in the client's browser to automatically generate the required encryptions, with the target plaintext being a secure cookie belonging to the client. AlFardan *et al.* [3] presented two

[1] Note that the WPA2 standard has optional support for TKIP for backward compatibility.

attacks on RC4 in TLS, one based on single-byte biases in the initial keystream bytes, the other based on the long-term Fluhrer-McGrew double-byte biases [6]. The first attack proceeds on a simple statistical basis: given enough ciphertext samples encrypting the same plaintext, one may proceed position by position, by simply trying each possibility for plaintext byte P_r (in position r), recovering the corresponding keystream bytes K_r, and then selecting as the output plaintext byte P_r the one for which the induced distribution on the keystream bytes has the highest likelihood when compared to an empirical estimate of the distribution for that position. This attack, therefore, requires the attacker to first build a good estimate of the keystream distribution in each output position r; this was done in [3] using 2^{44} random RC4 keys to get very accurate estimates of the keystream byte distributions.

It is natural to ask whether the same techniques might be deployed against other applications of the RC4 algorithm. In this paper, we address that question for WPA/TKIP.

1.1 Overview of Results

We begin by simply applying the single-byte plaintext recovery attack of AlFardan *et al.* [3] to RC4 with keys generated according to the TKIP specification, as described above. More exactly, we build a keystream estimate using 2^{41} RC4 keys obtained by considering 2^{19} random choices for `TK` and `TA`, and then 2^{22} `TSC` values (implemented as a counter with a random starting value). We then simulate the single-byte bias attack of [3], again generating the RC4 keys and ciphertexts according to the TKIP specification. This not only produces encouraging results in terms of plaintext recovery, but also reveals intriguing behaviour in the biases and in the plaintext recovery rates. The keystream biases that we observe exhibit more complex behaviour than for the random 16-byte RC4 keys used in TLS and that were considered in [3] (see in particular, Figs. 2, 3 and 4 in Sect. 3 below). This makes plaintext recovery easier in some positions, but harder in others when compared to the case of TLS.

In this first approach, the keystream estimates are calculated *averaging over the values of the pair* $(\texttt{TSC}_0, \texttt{TSC}_1)$, whereas it might be expected that there would also be keystream biases that depend on the specific values of $\texttt{TSC}_0$ and $\texttt{TSC}_1$ because of the way the TKIP key `K` in turn depends on these bytes. Thus it seems reasonable that, even though we have already observed significant and exploitable biases in the course of developing our first attack on TKIP, quite different and/or bigger biases might be found by sampling over keystreams for keys having specific values for the `TSC` pair $(\texttt{TSC}_0, \texttt{TSC}_1)$. In fact, we discover that the keystream distributions are not just different for different $(\texttt{TSC}_0, \texttt{TSC}_1)$ pairs – they are *radically* different. Moreover, very large biases in the RC4 keystream distributions appear, much larger than were observed in our first analysis. In a sense, these larger and different biases disappear to leave a different set of much smaller biases behind when one averages over the `TSC` pair $(\texttt{TSC}_0, \texttt{TSC}_1)$ as in our first attack.

These $(\mathtt{TSC}_0, \mathtt{TSC}_1)$-dependent biases can be exploited to build a second, more powerful attack, which, at a high level, works as follows:

1. Bin the available ciphertexts into 2^{16} bins according to the value of the TSC pair $(\mathtt{TSC}_0, \mathtt{TSC}_1)$. This binning is possible in TKIP because the TSC field is sent in the clear in each frame's header.
2. Perform a likelihood analysis of plaintext candidates for each of the bins.
3. Combine the resulting plaintext likelihood estimates for the different bins in a statistically sound procedure to get an estimate of the overall likelihood for each plaintext.

Essentially, while our first analysis effectively averages out any $(\mathtt{TSC}_0, \mathtt{TSC}_1)$-specific behaviour, our second, more delicate analysis exploits it to the full. We refer to this attack as a $(\mathtt{TSC}_0, \mathtt{TSC}_1)$ *binning attack*.

This second approach once again requires the computation of keystream biases, but now we need a good estimate of the distribution of keystream bytes in each output position r $(1 \leq r \leq 256)$[2] for each of the 2^{16} $(\mathtt{TSC}_0, \mathtt{TSC}_1)$ pairs, a dataset containing 2^{32} items. To gain a similar level of accuracy for each $(\mathtt{TSC}_0, \mathtt{TSC}_1)$ pair as we obtained for our first attack, we would need to compute statistics for around 2^{56} RC4 keystreams. This is currently well beyond our computational reach: generating 2^{40} RC4 keystreams currently takes about 4 days on our 16-core machine, and gives estimates for keystream biases based on only 2^{24} RC4 keystreams for each of the 2^{16} $(\mathtt{TSC}_0, \mathtt{TSC}_1)$ pairs, while the desired computation would be 2^{16} times larger (estimated at 2^{22} core days). The estimates for the biases that we get from 2^{24} RC4 keystreams per $(\mathtt{TSC}_0, \mathtt{TSC}_1)$ pair are somewhat noisy, in the sense that they do not accurately reflect the true keystream distributions except when there are large biases present. And, unfortunately, our experience is that using a noisy set of keystream estimates introduces inaccuracies into our plaintext recovery attacks which substantially reduces their success rates.

We present and investigate two methods to compensate for the problem of not having very accurate estimates of the keystream biases for each $(\mathtt{TSC}_0, \mathtt{TSC}_1)$ pair:

1. $\mathtt{TSC}_1$ is used to set two of the TKIP RC4 key bytes, so one may suspect that the keystream biases would be particularly dependent on this single byte of TSC. We therefore carry out the approach suggested above, but computing 2^8 keystream estimates, one for each value of $\mathtt{TSC}_1$, instead of the original 2^{16} estimates. In our experiments, we use 2^{32} keystreams to obtain each estimate, bringing the total computation up to that of 2^{40} RC4 keystreams. In the attack, we then bin the available ciphertexts into 2^8 bins according to the known value of $\mathtt{TSC}_1$, perform our likelihood analysis for each bin, and then combine the results. We refer to this attack as a $\mathtt{TSC}_1$ *binning attack*.

[2] We focus on the keystream distributions in the first 256 bytes because we did not observe significant biases beyond these bytes in our experiments, with the exception of byte 257.

2. Based on a computation with 2^{24} RC4 keystreams for each of the 2^{16} ($\mathtt{TSC}_0$, $\mathtt{TSC}_1$) pairs, and confirmed by larger computations for specific $(\mathtt{TSC}_0, \mathtt{TSC}_1)$ pairs, we have observed that there are many very large biases present in the $(\mathtt{TSC}_0, \mathtt{TSC}_1)$-specific keystreams. For example, while the typical bias observed in our first (TSC-averaged) analysis is on the order of $\pm 2^{-16}$, we find that there are many thousands of $(\mathtt{TSC}_0, \mathtt{TSC}_1)$-specific biases on the order of $\pm 2^{-12}$ and larger. (See Fig. 7(a) for a pictorial representation of the numbers of such large biases across all $(\mathtt{TSC}_0, \mathtt{TSC}_1)$ pairs in each position.) Heuristically, these biases might be expected to dominate the plaintext recovery procedure. We therefore "de-noise" our keystream distribution estimates by applying a cut-off procedure to them, setting all probability estimates that fall within a threshold around 2^{-8} to an average value, and leaving those that lie outside that threshold untouched. We then execute our binning attack using these idealised keystream estimates.

Of course, these two methods can be combined, and we examine the effect of doing so on the success rate of our plaintext recovery attacks.

As we shall see in Sect. 5, our attacks are effective. For example, using just the first method above with 2^{26} ciphertexts, we obtain an average success rate of 65 % in recovering each of the first 256 bytes of plaintext. The rate rises to higher than 90 % in even positions, this improvement being due to the presence of particularly large and $\mathtt{TSC}_1$-specific RC4 keystream biases in the even positions when TKIP keys are used.

1.2 Related Work

In independent and concurrent work, Sen Gupta *et al.* [16] have identified biases in WPA that are TSC-dependent, and speculated that there may be correlations between keystream bytes and linear combinations of the known key bytes $\mathtt{K}_0, \mathtt{K}_1, \mathtt{K}_2$ (which are computed exclusively from the TSC). This is similar to our approach. However, they did not perform a systematic search for biases, and did not apply them to plaintext recovery except in positions 1, 2, 3, 256 and 257. Their approach to plaintext recovery uses an *ad hoc* approach, with each linear combination being used to compute a keystream estimate, which then suggests a plaintext byte. Our approach is likelihood-based, and takes the reverse approach: for every possibility for the plaintext byte, and each $(\mathtt{TSC}_0, \mathtt{TSC}_1)$ pair (or $\mathtt{TSC}_1$ value), we compute the likelihood of the resulting induced keystream estimate, and combine these estimates to select the plaintext having the highest likelihood. Note that this recovery algorithm is optimal.

1.3 Paper Organisation

Section 2 provides further background on the RC4 stream cipher and its use in WPA. Section 3 reports biases in RC4 keystreams when RC4 is keyed according to the WPA specification, comparing biases for random-TSC WPA keys with biases for keys generated according to specific values of $\mathtt{TSC}_1$ and $(\mathtt{TSC}_0, \mathtt{TSC}_1)$.

Algorithm 1. Key schedule (KSA)

input : key K of l bytes
output: initial internal state st_0
begin
  for $i = 0$ **to** 255 **do**
    $\mathcal{S}[i] \leftarrow i$
  $j \leftarrow 0$
  for $i = 0$ **to** 255 **do**
    $j \leftarrow j + \mathcal{S}[i] + K[i \bmod l]$
    swap$(\mathcal{S}[i], \mathcal{S}[j])$
  $i, j \leftarrow 0$
  $st_0 \leftarrow (i, j, \mathcal{S})$
  return st_0

Algorithm 2. Keystream generator (PRGA)

input : internal state st_r
output: keystream byte Z_{r+1}
updated internal state st_{r+1}
begin
  parse $(i, j, \mathcal{S}) \leftarrow st_r$
  $i \leftarrow i + 1$
  $j \leftarrow j + \mathcal{S}[i]$
  swap$(\mathcal{S}[i], \mathcal{S}[j])$
  $Z_{r+1} \leftarrow \mathcal{S}[\mathcal{S}[i] + \mathcal{S}[j]]$
  $st_{r+1} \leftarrow (i, j, \mathcal{S})$
  return (Z_{r+1}, st_{r+1})

Fig. 1. Algorithms implementing the RC4 stream cipher. All additions are performed modulo 256.

Section 4 describes our plaintext recovery attacks on WPA that exploit these biases. We evaluate the attacks in Sect. 5 via simulation. Finally, Sect. 6 discusses the impact of and countermeasures to our attacks.

2 Further Background

2.1 The RC4 Stream Cipher

The stream cipher RC4, originally designed by Ron Rivest, became public in 1994 and found application in a wide variety of cryptosystems; well-known examples include SSL/TLS, WEP [1], WPA [2], and some Kerberos-related encryption modes [8]. RC4 has a remarkably short description and is extremely fast when implemented in software. However, these advantages come at the price of lowered security: several weaknesses have been identified in RC4 [5,6,9–11,17–19,24].

Technically, RC4 consists of two algorithms: a *key scheduling algorithm* (KSA) and a *pseudo-random generation algorithm* (PRGA), which are specified in Algorithms 1 and 2 (Fig. 1). The KSA takes as input a key K, typically a byte-array of length between 5 and 32 (i.e., 40–256 bits), and produces the initial internal state $st_0 = (i, j, \mathcal{S})$, where $\mathcal{S}$ is the canonical representation of a permutation on the set $[0, 255]$ as an array of bytes, and i, j are indices into this array. The PRGA will, given an internal state st_r, output 'the next' keystream byte Z_{r+1}, together with the updated internal state st_{r+1}.

2.2 WPA

We describe the cryptographic operation of WPA when RC4 is selected as the encryption method (referred to as TKIP). Our description is not complete, but provides sufficient detail to enable our subsequent attacks to be understood. We refer the reader to our introduction for an explanation of how TKIP generates its per-frame key `K` as a function `K` $\leftarrow$ `KM(TA, TK, TSC)` of the temporal encryption

key TK (128 bits), the TKIP sequence counter TSC (48 bits), and the transmitter address TA (48 bits). The per-frame key K is then used to produce an RC4 keystream, following the above description. The initialisation of RC4 in WPA is the standard one for this algorithm. Notably, none of the initial keystream bytes is discarded when RC4 is used in WPA, despite these bytes having known weaknesses.

The TKIP plaintext (consisting of the frame payload, a 64-bit MAC value MIC , and a 32-bit Integrity Check Vector ICV) is then XORed in a byte-by-byte fashion with the RC4 keystream, i.e., the ciphertext bytes are computed as

$$C_r = P_r \oplus Z_r \qquad \text{for } r = 1, 2, 3, \ldots ,$$

where P_r are the individual bytes of P, and Z_r are the RC4 keystream bytes. The data transmitted over the air then has the form

$$\mathtt{HDR} || C,$$

where C is the concatenation of the bytes C_r and HDR is the unencrypted frame header.

3 Biases in the RC4 Keystream for WPA Keys

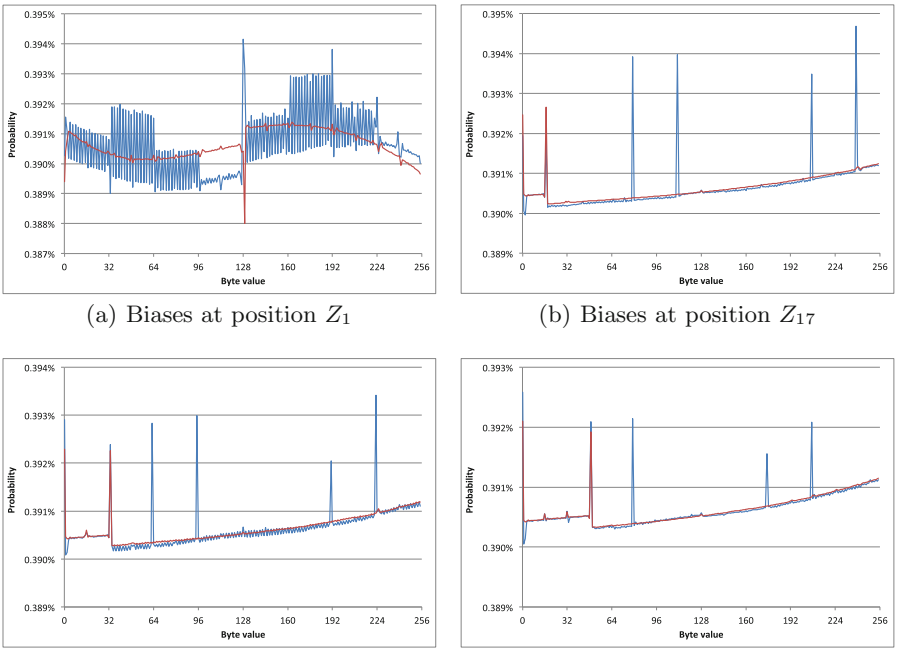
In the context of our analysis, we need to assess the strength of biases in the RC4 output streams for the keys K output by KM. If strong biases exist, then an attack on WPA is likely to be feasible using the ideas sketched in the introduction. That is, in a setting where the same plaintext message is repeatedly transmitted in a WPA-protected wireless network, one can expect that this plaintext is (at least partially) recoverable.

3.1 Fully Aggregated Biases for TKIP

We first experimentally determined single-byte keystream biases in WPA, without regard to TSC values (as would be consumed in a direct application of the attack from [3]). We refer to the biases obtained as being *fully aggregated biases*, since they are computed by using random TSC values and hence can be considered as being generated by aggregating over all $(\mathtt{TSC}_0, \mathtt{TSC}_1)$ pairs (in contrast to the $(\mathtt{TSC}_0, \mathtt{TSC}_1)$-pair-specific biases that we consider below). More precisely, we implemented the KM key derivation function, verified it against the test vectors from [2, Annex M.1.2], and computed keys

$$\mathtt{KM}(\mathtt{TA}, \mathtt{TK}, \mathtt{TSC}), \mathtt{KM}(\mathtt{TA}, \mathtt{TK}, \mathtt{TSC} + 1), \ldots, \mathtt{KM}(\mathtt{TA}, \mathtt{TK}, \mathtt{TSC} + (2^{22} - 1))$$

for 2^{19} random assignments of variables TA, TK, TSC, aiming at modelling a realistic application of TKIP. Considering all resulting 2^{41} RC4 keys, we measured the distribution of keystream bytes at positions 1–256. Independently, from the set of all keys K consistent with Eq. (1), we generated 2^{41} random keys (i.e., with random TSC, but also setting $\mathtt{K}_3, \ldots, \mathtt{K}_{15}$ randomly instead of using the Feistel cipher).



(a) Biases at position Z_1

(b) Biases at position Z_{17}

(c) Biases at position Z_{33}

(d) Biases at position Z_{49}

Fig. 2. Measured distribution of the TKIP keystream at positions Z_1, Z_{17}, Z_{33}, and Z_{49} (blue). These estimates were obtained by considering more than 2^{41} KM-generated keys. For reference, we overlay the biases of the RC4 keystream with random 128-bit keys (red) (Color figure online).

We identified the corresponding keystream distributions at the same positions. We observed that the difference between these two sets of distributions is small, allowing us to make the assumption that the action of the Feistel cipher does not affect the output distribution of RC4. We hence base all of our further observations on statistics obtained from random keys conforming with (1). Here, and throughout, we use AES with a fixed key in counter mode to generate any random values needed (so that they are in fact pseudorandom, and we are relying on AES being a good block cipher to ensure our keys are well distributed).

In contrast to the internal Feistel cipher of KM, the structure on RC4 keys implied by Eq. (1) has a significant influence on the aggregated biases in TKIP. For instance, at positions 17, 33, 49, 65, 81, and 97 (i.e., $16k + 1$ for small k), new peaks in the distribution show up that do not appear in RC4 with random 128-bit keys. For the distributions at positions 17, 33, and 49 see Fig. 2(b), (c), and (d). Even more extreme is the difference at position 1, shown in Fig. 2(a). It is also interesting to observe how the bias towards 0x00 behaves in the TKIP case: the probability $\Pr(Z_r = \texttt{0x00})$ is persistently smaller than in the case of random 128-bit keys at positions 2–32 and 128–160, whereas for the other positions its value alternates from byte to byte between being significantly larger and being significantly smaller than the corresponding probabilitiy for uniform keys.

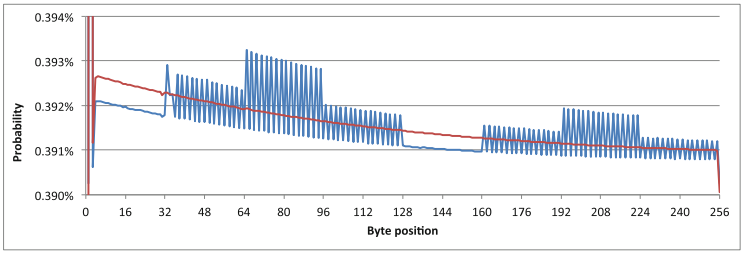
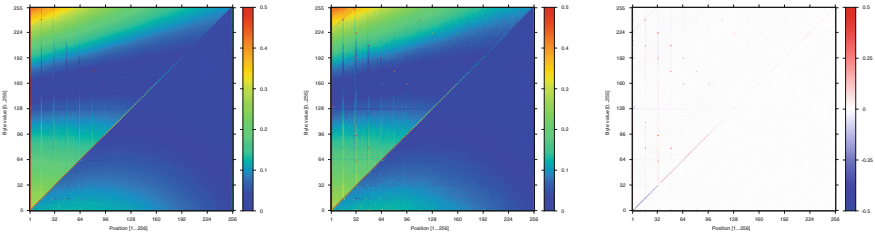


Fig. 3. Strength of the bias towards `0x00` of TKIP keystream bytes at positions 1–256 (blue). The red line corresponds to the biases of RC4 with random 128-bit keys (Color figure online).



(a) Biases in RC4 keystream (random 128-bit keys)

(b) Biases in RC4 keystream (TKIP keys)

(c) Map indicating the difference between Figures 4(a) and 4(b).

Fig. 4. Pictorial representation of biases in RC4 keystreams for random 128-bit keys and for TKIP keys, for different positions (x-axis) and byte values (y-axis). For each position we encode the bias in the keystream for the (position, value) combination as a colour; in figures (a) and (b) the colouring scheme encodes the absolute biases, i.e., the absolute difference between the occurring probabilities and the (expected) probability 1/256, scaled up by a factor of 2^{16}, capped to a maximum of 0.5. In figure (c), the colour encodes the difference between the absolute biases arising for random 128-bit keys and for TKIP keys, scaled up by a factor of 2^{16} and capped to the range $[-0.5, 0.5]$ (Color figure online).

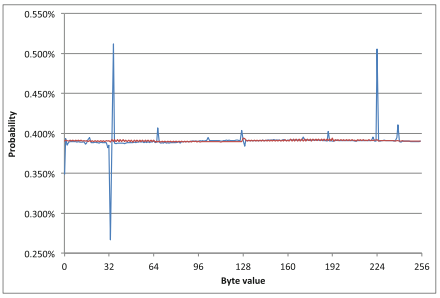
This is illustrated in Fig. 3, which compares the strength of the bias towards `0x00` for TKIP and for random 128-bit RC4 keys. Finally, to make it easier to compare the TKIP-specific biases with the biases for random 128-bit keys in [3], we present Fig. 4, in which we depict side-by-side the full set of biases for both cases. Figure 4(c) shows the differences between the two sets of biases; blue and red pixels show the places where the biases differ significantly.

It is an interesting theoretical problem to explain the differences between RC4 biases for random 128-bit keys and TKIP keys (though we stress that having such an explanation does not affect the performance of our attacks to follow).

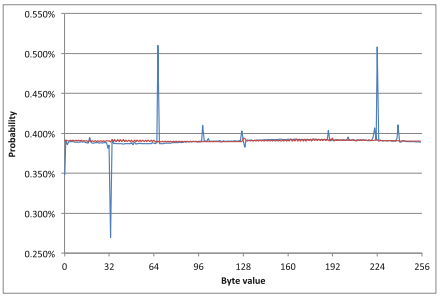
3.2 $(\mathtt{TSC}_0, \mathtt{TSC}_1)$-pair-specific Biases for TKIP

As explained in the introduction, we wish to examine how the biases in RC4 keystreams for TKIP keys depend on the $(\mathtt{TSC}_0, \mathtt{TSC}_1)$ byte pair used in defining the keys. For each $(\mathtt{TSC}_0, \mathtt{TSC}_1)$ pair, we computed 2^{24} RC4 keystreams by assigning the bytes $\mathtt{K}_0$, $\mathtt{K}_1$, $\mathtt{K}_2$ according to the $(\mathtt{TSC}_0, \mathtt{TSC}_1)$ pair (as per the specification, see Eq. (1)) and assigning the remaining 13 key bytes at random.

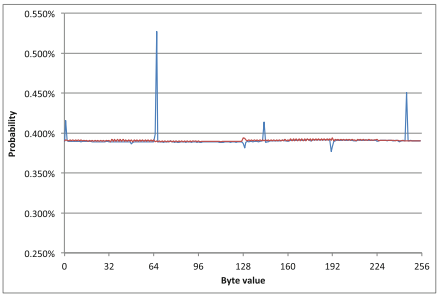
Using each set of 2^{24} RC4 keys, we then computed the distribution of keystream bytes at positions 1–256, giving a dataset containing 256 keystream byte distributions for each of the 2^{16} $(\mathtt{TSC}_0, \mathtt{TSC}_1)$ pairs. Of course, it is not possible to represent such a large dataset in full here, but we provide two small samples in Figs. 5 and 6. The former shows how the distribution of keystream byte Z_1 depends heavily on the value of the $(\mathtt{TSC}_0, \mathtt{TSC}_1)$ pair. The latter shows how different is the distribution of keystream bytes in a variety of positions for a specific $(\mathtt{TSC}_0, \mathtt{TSC}_1)$ pair, $(\mathtt{0x00}, \mathtt{0x00})$, when compared to the fully aggregated



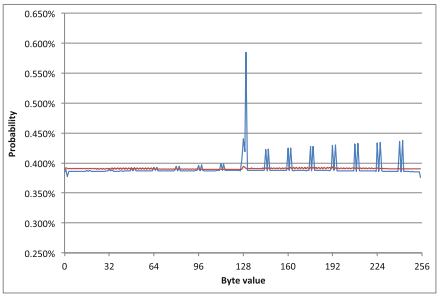
(a) Biases for $(\mathtt{TSC}_0, \mathtt{TSC}_1) = (\mathtt{0x00}, \mathtt{0x00})$



(b) Biases for $(\mathtt{TSC}_0, \mathtt{TSC}_1) = (\mathtt{0x00}, \mathtt{0x20})$

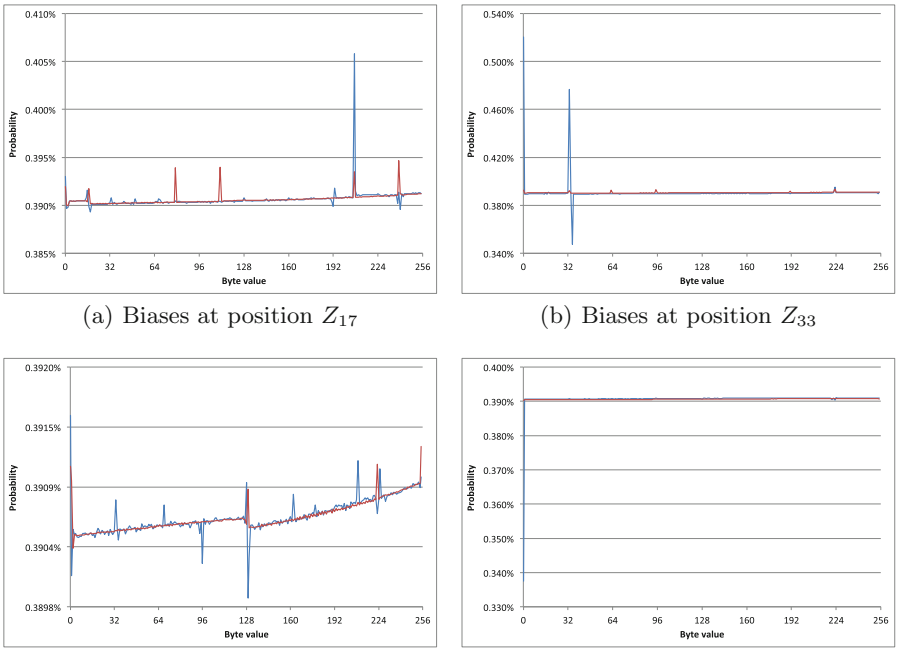


(c) Biases for $(\mathtt{TSC}_0, \mathtt{TSC}_1) = (\mathtt{0x8F}, \mathtt{0x34})$



(d) Biases for $(\mathtt{TSC}_0, \mathtt{TSC}_1) = (\mathtt{0xFF}, \mathtt{0xFF})$

Fig. 5. Measured distribution of the TKIP RC4 keystream at position Z_1 for $(\mathtt{TSC}_0, \mathtt{TSC}_1)$ pairs $(\mathtt{0x00}, \mathtt{0x00})$, $(\mathtt{0x00}, \mathtt{0x20})$, $(\mathtt{0x8F}, \mathtt{0x34})$, $(\mathtt{0xFF}, \mathtt{0xFF})$ (blue). These estimates were obtained by considering more than 2^{36} keys per $(\mathtt{TSC}_0, \mathtt{TSC}_1)$ pair. For reference, we overlay the corresponding fully aggregated TKIP keystream biases (red) (Color figure online).



(a) Biases at position Z_{17}

(b) Biases at position Z_{33}

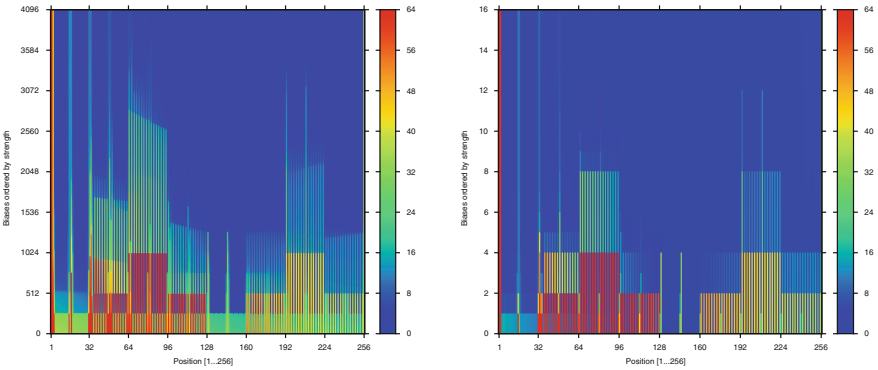
(c) Biases at position Z_{129}

(d) Biases at position Z_{256}

Fig. 6. Measured distribution of the TKIP keystream at positions Z_{17}, Z_{33}, Z_{129}, Z_{256} for $(\texttt{TSC}_0, \texttt{TSC}_1)$ pair $(\texttt{0x00}, \texttt{0x00})$ (blue; see Fig. 5(a) for distribution of Z_1). These estimates were obtained by considering more than 2^{36} keys per TSC pair. For reference, we overlay the corresponding fully aggregated TKIP keystream biases (red) (Color figure online).

results reported above. This is indicative that plaintext recovery attacks that focus on exploiting biases arising for individual $(\texttt{TSC}_0, \texttt{TSC}_1)$ pairs may perform better than those working with fully aggregated biases.

Our plaintext recovery attacks to be presented in Sect. 4 proceed on a position-by-position basis, and can be expected to work well in a given position r if there are large biases in that position over the different $(\texttt{TSC}_0, \texttt{TSC}_1)$ pairs. Figure 7(a) shows that large biases are indeed plentiful and well-spread over the keystream positions. For instance, it reveals that the 256 strongest biases at positions 1–128 have a value of about 2^{-11}, with a couple of exceptions where strengths of more than 2^{-10} can be reported. Even more impressive are the many thousands of biases of strength $> 2^{-10}$ at positions 1–3. Also for position 256 many thousands of relatively strong biases do exist. The interesting structure in intervals 32–128 and 160–256, where positions with strong biases alternate with positions having no strong biases, will considerably affect the recovery rate of our attacks, as we will see. To conclude, the numbers of large biases seen is significantly larger than one would expect if the keystream bytes were uniformly random. For example, with 2^{24} keystreams per $(\texttt{TSC}_0, \texttt{TSC}_1)$ pair, we would expect the count for each byte value in each position to follow a (roughly) Normal distribution with mean 2^{16} and standard deviation σ



(a) Largest 4096 biases of the RC4 keystream, taken over all 2^{16} $(\mathtt{TSC}_0, \mathtt{TSC}_1)$ combinations

(b) Largest 16 biases of the RC4 keystream, taken over all 2^8 aggregated counts for $\mathtt{TSC}_1$

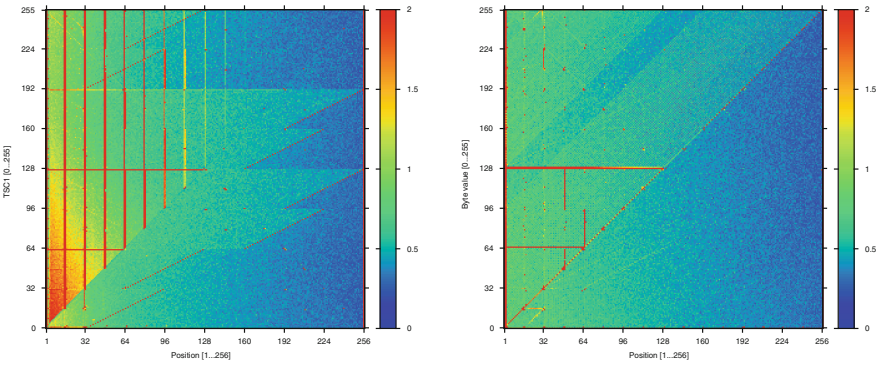
Fig. 7. Pictorial representation of the number of large biases in TKIP keystream distributions across different positions in the keystream. For each position we show the strengths of the largest biases, sorted in descending order (largest bias on the bottom line). The colouring scheme encodes the absolute difference between the occurring probabilities and the (expected) probability 1/256, scaled up by a factor of 2^{16}, capped to a maximum of 64 (Color figure online).

approximately 2^8. We would then expect the number of counts outside the range $\left[2^{16} - 2^{10}, 2^{16} + 2^{10}\right]$ (i.e., outside the 4σ range) to be roughly 0.2 %, whereas the actual rate of such counts is about 1.5–2 % for at least half of the positions, and for some positions even larger.

We recall from the introduction our hunch that the TKIP keystream biases would be particularly dependent on the single byte $\mathtt{TSC}_1$. To test this, we used our bias data for all $(\mathtt{TSC}_0, \mathtt{TSC}_1)$ pairs to compute $\mathtt{TSC}_0$-*aggregated biases*, that is, we aggregated our previous data over $\mathtt{TSC}_0$ values to obtain 2^8 different keystream distributions for positions 1–256, one distribution for each value of $\mathtt{TSC}_1$. Effectively, this gives us distribution estimates based on 2^{32} keystreams for each value of $\mathtt{TSC}_1$. We then compared the original distributions to the $\mathtt{TSC}_0$-aggregated data.

An indication towards the correctness of our hunch is provided by Fig. 7(b) that reports strengths of biases similarly to Fig. 7(a) – however aggregating over $\mathtt{TSC}_0$ values as described. Indeed, both the obvious similarity of the graphs and the applied scaling factor of 256 along the y-axis are exactly as expected when assuming that each strong bias in the aggregated counts appears 256 times in the plain (unaggregated) counts. This suggests that our hunch concerning the relative importance of $\mathtt{TSC}_1$ is correct, and gives weight to the idea of considering $\mathtt{TSC}_0$-aggregated biases in our plaintext recovery attack (the first of our two methods designed to compensate for the problem of not having very accurate keystream bias estimates as mentioned in the introduction).

Given a fixed position in the keystream, our attack on TKIP works best if there are strong biases for many $(\mathtt{TSC}_0, \mathtt{TSC}_1)$ combinations (or $\mathtt{TSC}_1$ values) for that position. In practice, only TKIP frames with such $\mathtt{TSC}$ values will contribute



(a) Strength of largest available bias for each combination of position and $\texttt{TSC}_1$ value

(b) Strength of largest available bias for each combination of position and byte value

Fig. 8. Pictorial representation of the correlation between position in TKIP keystream, $\texttt{TSC}_1$ value (respectively, byte value), and the corresponding strength of the largest occurring bias. The colouring scheme encodes the absolute difference between the occurring probabilities and the (expected) probability 1/256, scaled up by a factor of 2^{16}, capped to a maximum of 2 (Color figure online).

noticeably to plaintext recovery. While Fig. 7(a) and (b) tell us that quite large biases do exist almost everywhere in the first 256 positions of the RC4 keystream *for some* TSC values, it is yet unclear for which TSC values they occur. We provide Fig. 8(a) and (b) to shed more light on the distribution of 'bias-friendly' TSC values. We see that for positions 1–3, 16–17, 32–33, 48–49, 64–65, 80–81, and 96–97 strong biases exist for more than 50 % of all $\texttt{TSC}_1$ values. Orthogonally to that, a $\texttt{TSC}_1$ value of 127 guarantees strong biases in the first 128 keystream positions. Further, it is quite interesting to trace the origin of the alternating behaviour in Fig. 7(a) and (b) at position ranges 32–128 and 160–256. Finally, note the strong tendency at positions 1–128 of the TKIP cipher to produce byte values 128 and 129, and also value 65 at positions 1–64.

Again, we note that finding the underlying reasons for the observed bias behaviours in TKIP keystreams is an interesting theoretical problem.

4 Plaintext Recovery Attacks

4.1 The Attack of AlFardan *et al.*

The idea behind the single-byte bias attack of AlFardan *et al.* [3] is to first obtain a detailed picture of the distributions of RC4 keystream bytes Z_r, for all positions r of interest, by gathering statistics from keystreams generated using a large number of independent keys. That is, for all r, we (empirically) estimate

$$p_{r,k} := \Pr(Z_r = k), \quad k = \texttt{0x00}, \ldots, \texttt{0xFF},$$

where the probability is taken over a random choice of the RC4 encryption key. In [3], these keys were taken to be random 128-bit values, reflecting how session keys are set in TLS; for TKIP, these keys should be generated according to the procedure described in Sect. 3.1.

The second step in the approach of [3] is to use the $p_{r,k}$ estimates to recover plaintext using a maximum-likelihood approach, as follows. Suppose we have S ciphertexts $C_1, \ldots, C_S$ available for our attack (for the r-th byte of ciphertext C_j we write $C_{j,r}$). For any fixed position r and any candidate plaintext byte μ for that position, vector $(N_{\texttt{0x00}}^{(\mu)}, \ldots, N_{\texttt{0xFF}}^{(\mu)})$ with

$$N_k^{(\mu)} = |\{j \mid C_{j,r} = k \oplus \mu\}_{1 \le j \le S}| \qquad (\texttt{0x00} \le k \le \texttt{0xFF})$$

represents the distribution on Z_r required to obtain the observed ciphertext bytes $\{C_{j,r}\}_{1 \le j \le S}$ by encrypting μ. We compare these *induced* distributions (one for each possible μ) with the accurate distribution $p_{r,\texttt{0x00}}, \ldots, p_{r,\texttt{0xFF}}$ and interpret a close match as an indication for the corresponding plaintext candidate μ being the correct one, i.e., $P_r = \mu$. More formally, we observe that the probability λ_μ that plaintext byte μ is encrypted to ciphertext bytes $\{C_{j,r}\}_{1 \le j \le S}$ follows a multinomial distribution:

$$\lambda_\mu = \frac{S!}{N_{\texttt{0x00}}^{(\mu)}! \cdots N_{\texttt{0xFF}}^{(\mu)}!} \prod_{k \in \{\texttt{0x00}, \ldots, \texttt{0xFF}\}} p_{r,k}^{N_k^{(\mu)}} \quad . \tag{2}$$

The approach of [3] then determines the (optimal) maximum-likelihood plaintext byte value μ by computing λ_μ for all $\texttt{0x00} \le \mu \le \texttt{0xFF}$ and identifying μ such that λ_μ is largest. Algorithm 3 more formally specifies the described attack, incorporating some optimizations discussed in [3] (in particular, as the fraction in Eq. (2) is independent of μ, we compute the λ_μ values only up to that constant; in fact, we actually compute and compare $\log \lambda_\mu$, rather than λ_μ).

4.2 Attack Based on $(\texttt{TSC}_0, \texttt{TSC}_1)$ Pair Binning

We next discuss our extension of the attack in Algorithm 3 that uses the single-byte RC4 biases, along with their strengths, on a per $(\texttt{TSC}_0, \texttt{TSC}_1)$ pair basis. For ease of notation, we let $\overline{\texttt{TSC}}$ denote the pair $(\texttt{TSC}_0, \texttt{TSC}_1)$ in mathematical expressions.

The idea is to first obtain a detailed picture of the distributions of RC4 keystream bytes Z_r, for all positions r in some range, on a per $(\texttt{TSC}_0, \texttt{TSC}_1)$ pair basis, by gathering statistics from keystreams generated using a large number of keys (2^{24} per $(\texttt{TSC}_0, \texttt{TSC}_1)$ pair in our case). That is, for all r in our selected range, we now estimate

$$p_{\overline{\texttt{TSC}},r,k} := \Pr(Z_r = k),\ \overline{\texttt{TSC}} = (\texttt{0x00}, \texttt{0x00}), \ldots, (\texttt{0xFF}, \texttt{0xFF}),\ k = \texttt{0x00}, \ldots, \texttt{0xFF}$$

where the probability is taken over the random choice of the RC4 encryption key $\texttt{K}$, subject to the structure on $\texttt{K}_0$, $\texttt{K}_1$, $\texttt{K}_2$ induced by $\overline{\texttt{TSC}} = (\texttt{TSC}_0, \texttt{TSC}_1)$.

Using these biases $p_{\overline{\texttt{TSC}},r,k}$, in a second step, plaintext can be recovered using a variation of the preceding maximum-likelihood approach, as follows.

Algorithm 3. Single-byte bias attack from [3]

input : $\{C_j\}_{1 \le j \le S}$ – S independent encryptions of fixed plaintext P
r – byte position
$(p_{r,k})_{\texttt{0x00} \le k \le \texttt{0xFF}}$ – keystream distribution at position r
output: P_r^* – estimate for plaintext byte P_r
begin
  $N_{\texttt{0x00}} \leftarrow 0, \ldots, N_{\texttt{0xFF}} \leftarrow 0$
  for $j = 1$ **to** S **do**
    $N_{C_{j,r}} \leftarrow N_{C_{j,r}} + 1$
  for $\mu = \texttt{0x00}$ **to** $\texttt{0xFF}$ **do**
    for $k = \texttt{0x00}$ **to** $\texttt{0xFF}$ **do**
      $N_k^{(\mu)} \leftarrow N_{k \oplus \mu}$
    $\lambda_\mu \leftarrow \sum_{k=\texttt{0x00}}^{\texttt{0xFF}} N_k^{(\mu)} \log p_{r,k}$
  $P_r^* \leftarrow \arg\max_{\mu \in \{\texttt{0x00},\ldots,\texttt{0xFF}\}} \lambda_\mu$
  return P_r^*

Suppose we have S ciphertexts $C_1, \ldots, C_S$ available for our attack. We partition these into 2^{16} groups according to the value of the $(\texttt{TSC}_0, \texttt{TSC}_1)$ pair; for convenience, we assume the resulting bins of ciphertexts are all of equal size $T = S/2^{16}$, but this need not be the case. Let the bin of ciphertexts associated with a particular $\overline{\texttt{TSC}} = (\texttt{TSC}_0, \texttt{TSC}_1)$ pair be denoted $\mathcal{S}_{\overline{\texttt{TSC}}}$ and have members $C_{\overline{\texttt{TSC}},j}$ for $j = 1, \ldots, T$; we denote the byte at position r of $C_{\overline{\texttt{TSC}},j}$ by $C_{\overline{\texttt{TSC}},j,r}$. For any fixed position r and any candidate plaintext byte μ for that position, vector $(N^{(\mu)}_{\overline{\texttt{TSC}},\texttt{0x00}}, \ldots, N^{(\mu)}_{\overline{\texttt{TSC}},\texttt{0xFF}})$ with

$$N^{(\mu)}_{\overline{\texttt{TSC}},k} = |\{j \mid C_{\overline{\texttt{TSC}},j,r} = k \oplus \mu\}_{1 \le j \le T}| \qquad (\texttt{0x00} \le k \le \texttt{0xFF})$$

represents the distribution on Z_r required to obtain the observed ciphertext bytes $\{C_{\overline{\texttt{TSC}},j,r}\}_{1 \le j \le T}$ for bin $\mathcal{S}_{\overline{\texttt{TSC}}}$ by encrypting μ. We compare these *induced* distributions (one for each possible μ and for each possible $(\texttt{TSC}_0, \texttt{TSC}_1)$ pair) with the accurate distribution $p_{\overline{\texttt{TSC}},r,\texttt{0x00}}, \ldots, p_{\overline{\texttt{TSC}},r,\texttt{0xFF}}$ and interpret a close match as being an indication for the corresponding plaintext candidate μ being the correct one, i.e., $P_r = \mu$, in bin $\mathcal{S}_{\overline{\texttt{TSC}}}$. The probability $\lambda_{\overline{\texttt{TSC}},\mu}$ that plaintext byte μ is encrypted to ciphertext bytes $\{C_{\overline{\texttt{TSC}},j,r}\}_{1 \le j \le T}$ in bin $\mathcal{S}_{\overline{\texttt{TSC}}}$ now follows a multinomial distribution:

$$\lambda_{\overline{\texttt{TSC}},\mu} = \frac{T!}{N^{(\mu)}_{\overline{\texttt{TSC}},\texttt{0x00}}! \cdots N^{(\mu)}_{\overline{\texttt{TSC}},\texttt{0xFF}}!} \prod_{k \in \{\texttt{0x00},\ldots,\texttt{0xFF}\}} p_{\overline{\texttt{TSC}},r,k}^{N^{(\mu)}_{\overline{\texttt{TSC}},k}} . \tag{3}$$

The probability that plaintext byte μ is encrypted to ciphertext bytes $\{C_{\overline{\texttt{TSC}},j,r}\}_{1 \le j \le T}$ across all bins $\mathcal{S}_{\overline{\texttt{TSC}}}$ can then be precisely calculated as

$$\lambda_\mu = \prod_{(\texttt{0x00},\texttt{0x00}) \le \overline{\texttt{TSC}} \le (\texttt{0xFF},\texttt{0xFF})} \lambda_{\overline{\texttt{TSC}},\mu} .$$

By computing λ_μ for all $\texttt{0x00} \leq \mu \leq \texttt{0xFF}$, and identifying μ such that λ_μ is largest, we determine the (optimal) maximum-likelihood plaintext byte value. This informal description, together with some optimisations that we describe next, is specified in algorithmic form in Algorithm 4.

Observe that, for each fixed position r and set of ciphertexts $\{C_{\overline{\texttt{TSC}},j,r}\}_{1\leq j\leq T}$, values $N^{(\mu)}_{\overline{\texttt{TSC}},k}$ can be computed from values $N^{(\mu')}_{\overline{\texttt{TSC}},k}$ by equation $N^{(\mu)}_{\overline{\texttt{TSC}},k} = N^{(\mu')}_{\overline{\texttt{TSC}},k\oplus\mu'\oplus\mu}$, for all k. In other words, for a fixed $(\texttt{TSC}_0, \texttt{TSC}_1)$ pair, vectors $(N^{(\mu)}_{\overline{\texttt{TSC}},\texttt{0x00}}, \ldots, N^{(\mu)}_{\overline{\texttt{TSC}},\texttt{0xFF}})$ and $(N^{(\mu')}_{\overline{\texttt{TSC}},\texttt{0x00}}, \ldots, N^{(\mu')}_{\overline{\texttt{TSC}},\texttt{0xFF}})$ are permutations of each other; by consequence, the term $T!/(N^{(\mu)}_{\overline{\texttt{TSC}},\texttt{0x00}}! \cdots N^{(\mu)}_{\overline{\texttt{TSC}},\texttt{0xFF}}!)$ in Eq. (3) is a constant for each choice of μ (but not necessarily constant across different values for the $(\texttt{TSC}_0, \texttt{TSC}_1)$ pair). If T is fixed (as we assume it to be), then the $T!$ terms can all be omitted from all calculations. Furthermore, computing and comparing $\log(\lambda_{\overline{\texttt{TSC}},\mu})$ and $\log(\lambda_\mu)$ instead of $\lambda_{\overline{\texttt{TSC}},\mu}$ and λ_μ makes the computation more efficient and accuracy easier to maintain.

Comparing with Algorithm 3, we see that our new Algorithm 4, at its core, runs Algorithm 3 once for each $(\texttt{TSC}_0, \texttt{TSC}_1)$ pair, and then combines the resulting likelihood estimates $\lambda_{\overline{\texttt{TSC}},\mu}$ to obtain the final estimate λ_μ for plaintext candidate μ. Some care is needed, however, to use the correct scaling factors ($T!$ and $N^{(\mu)}_{\overline{\texttt{TSC}},\texttt{0x00}}! \cdots N^{(\mu)}_{\overline{\texttt{TSC}},\texttt{0xFF}}!$) for each $(\texttt{TSC}_0, \texttt{TSC}_1)$ pair.

4.3 Attack Based on Aggregation Over $\texttt{TSC}_0$ Values

As mentioned in the introduction, one method of coping with noisy estimates for the probabilities $p_{\overline{\texttt{TSC}},r,k}$ is to consider aggregation of biases over $\texttt{TSC}_0$. This is supported by the experiments reported in Sect. 3.2, where we saw that there is broad agreement between the $\texttt{TSC}_0$-aggregated data and the data for individual $(\texttt{TSC}_0, \texttt{TSC}_1)$ pairs.

It is not difficult to see how to modify Algorithm 4 to work with 2^8 bins, one for each value of $\texttt{TSC}_1$, instead of 2^{16} bins. The execution of the modified algorithm becomes in practice faster, since each estimate for a plaintext byte μ now only involves calculation of $\lambda_{\overline{\texttt{TSC}},\mu}$ over 2^8 $\texttt{TSC}_1$ values instead of 2^{16} $(\texttt{TSC}_0, \texttt{TSC}_1)$ pair values.

4.4 Further Optimizations

In specific settings where the attacker has *a priori* information about the encrypted plaintext the performance of Algorithms 3 and 4 can be further improved. Here, the considerations are similar to those in [3] and so we omit further discussion.

5 Experimental Results

In this section, we report on the results obtained by simulating the plaintext recovery attacks described in Sect. 4. To be exact, we did not mount the attacks

Algorithm 4. Plaintext recovery attack using $(\mathtt{TSC}_0, \mathtt{TSC}_1)$ binning

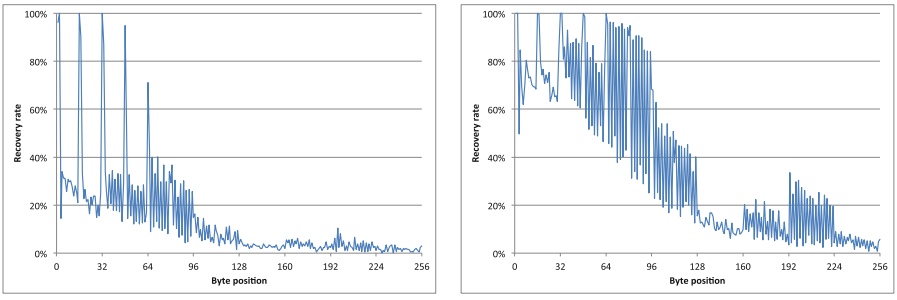
input : $\{C_{\overline{\mathtt{TSC}},j}\}_{(\mathtt{0x00},\mathtt{0x00})\leq\overline{\mathtt{TSC}}\leq(\mathtt{0xFF},\mathtt{0xFF}),1\leq j\leq T}$ – $S = 2^{16}\cdot T$ independent encryptions of fixed plaintext P
r – byte position
$(p_{\overline{\mathtt{TSC}},r,k})_{(\mathtt{0x00},\mathtt{0x00})\leq\overline{\mathtt{TSC}}\leq(\mathtt{0xFF},\mathtt{0xFF}),\mathtt{0x00}\leq k\leq\mathtt{0xFF}}$ – keystream distributions for all $(\mathtt{TSC}_0, \mathtt{TSC}_1)$ pairs at position r
output: P_r^* – estimate for plaintext byte P_r
begin
  $N_{(\mathtt{0x00},\mathtt{0x00}),\mathtt{0x00}} \leftarrow 0, \ldots, N_{(\mathtt{0xFF},\mathtt{0xFF}),\mathtt{0xFF}} \leftarrow 0$
  for $\overline{\mathtt{TSC}} = (\mathtt{0x00},\mathtt{0x00})$ **to** $(\mathtt{0xFF},\mathtt{0xFF})$ **do**
    for $j = 1$ **to** T **do**
      $k \leftarrow C_{\overline{\mathtt{TSC}},j,r}$
      $N_{\overline{\mathtt{TSC}},k} \leftarrow N_{\overline{\mathtt{TSC}},k} + 1$
  for $\overline{\mathtt{TSC}} = (\mathtt{0x00},\mathtt{0x00})$ **to** $(\mathtt{0xFF},\mathtt{0xFF})$ **do**
    $F_{\overline{\mathtt{TSC}}} \leftarrow \sum_{\mathtt{0x00}\leq j\leq\mathtt{0xFF}} \log((N_{\overline{\mathtt{TSC}},j})!)$
    for $\mu = \mathtt{0x00}$ **to** $\mathtt{0xFF}$ **do**
      for $k = \mathtt{0x00}$ **to** $\mathtt{0xFF}$ **do**
        $N^{(\mu)}_{\overline{\mathtt{TSC}},k} \leftarrow N_{\overline{\mathtt{TSC}},k\oplus\mu}$
      $\lambda_{\overline{\mathtt{TSC}},\mu} \leftarrow -F_{\overline{\mathtt{TSC}}} + \sum_{k=\mathtt{0x00}}^{\mathtt{0xFF}} N^{(\mu)}_{\overline{\mathtt{TSC}},k} \log(p_{\overline{\mathtt{TSC}},r,k})$
  for $\mu = \mathtt{0x00}$ **to** $\mathtt{0xFF}$ **do**
    $\lambda_\mu \leftarrow \sum_{(\mathtt{0x00},\mathtt{0x00})\leq\overline{\mathtt{TSC}}\leq(\mathtt{0xFF},\mathtt{0xFF})} \lambda_{\overline{\mathtt{TSC}},\mu}$
  $P_r^* \leftarrow \arg\max_{\mu\in\{\mathtt{0x00},\ldots,\mathtt{0xFF}\}} \lambda_\mu$
  return P_r^*

against real TKIP traffic, but instead generated TKIP ciphertexts corresponding to a plaintext consisting of 0x00 bytes and then tested our attacks' abilities to recover this plaintext.

5.1 Attack Using Fully Aggregated Biases

We first ran 256 times the attack in Algorithm 3 for each of $S = 2^{24}, 2^{26}, 2^{28}, 2^{30}$ simulated frames to estimate the attack's success rate. We used the fully aggregated biases described in Sect. 3.1 in the attack. The results are shown in Fig. 9(a)–(d), which display the success rate of recovering the correct plaintext byte versus the byte position r in the keystream. Some notable features of these figures are:

- With $S = 2^{26}$ frames, the first 55 plaintext bytes are recovered with rate at least 50 % per byte. Comparing Fig. 9(b) with the corresponding Fig. 5(a) from [3] created for random 128-bit keys reveals that many plaintext bytes are recovered with a significantly higher rate in the TKIP case, leading to a higher average recovery rate.
- With $S = 2^{30}$ frames, the first 130 plaintext bytes are recovered with rate close to 100 %; the first 211 bytes are recovered with rate at least 50 %. Note



(a) 2^{24} frames (b) 2^{26} frames

(c) 2^{28} frames (d) 2^{30} frames

Fig. 9. Success rate for 256 runs of attack based on fully aggregated TKIP biases using 2^{24}, 2^{26}, 2^{28} and 2^{30} simulated frames (4, 16, 64, and 256 keys generating 2^{22} frames each).

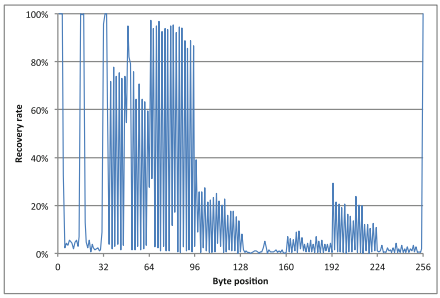
again that, while according to Fig. 5(c) in [3] for random 128-bit keys the first 251 bytes are recovered with at least 50 % probability, in the TKIP case many plaintext bytes are recovered with significantly higher probability.

– Independently of the number S of considered frames, the recovery rate is highly correlated with the strength of the bias towards `0x00` at the same position: to see this, compare Fig. 9(a)–(d) with Fig. 3.

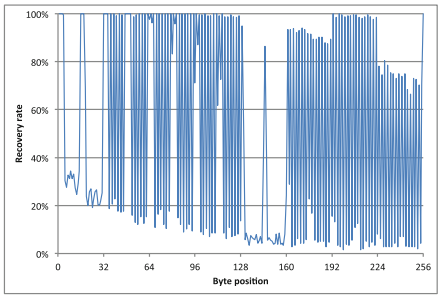
By comparing Fig. 9 with the corresponding figures in [3], we observe that the structure on keys enforced by (1), which was aiming to 'preclude the use of known RC4 weak keys' [2] (to prevent WEP key recovery), effectively allows easier recovery of plaintext bytes than with uniform keys, at least in some positions.

5.2 Attacks Using TSC Binning

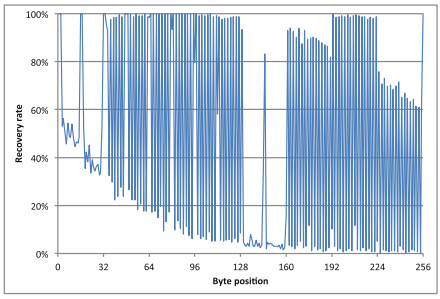
Secondly, we simulated the attack based on $(\mathtt{TSC}_0, \mathtt{TSC}_1)$ pair binning described in Algorithm 4, as well as the variant of the attack described in Sect. 4.3 which aggregates the biases over all $\mathtt{TSC}_0$ values. For both attacks, we used per-output-byte probabilities $\{p_{\overline{\mathtt{TSC}},r,k}\}_{1\le r\le 256,\mathtt{0x00}\le k\le \mathtt{0xFF}}$ derived from the keystream distribution estimate described in Sect. 3.2. Recall that this estimate was generated based on 2^{24} RC4 keystreams for each of the 2^{16} $(\mathtt{TSC}_0, \mathtt{TSC}_1)$ pairs. To judge



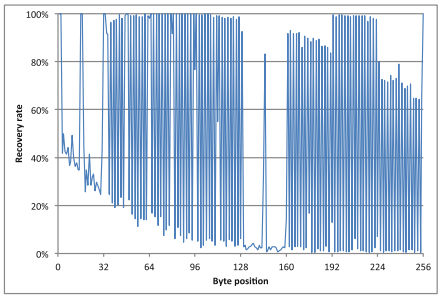
(a) Attack based on $(\texttt{TSC}_0, \texttt{TSC}_1)$ pair binning using non-idealised keystream distribution estimate.



(b) Attack based on $(\texttt{TSC}_0, \texttt{TSC}_1)$ pair binning using idealised keystream distribution estimate.



(c) Attack based on aggregation of $\texttt{TSC}_0$ values using non-idealised keystream distribution estimate.



(d) Attack based on aggregation of $\texttt{TSC}_0$ values using idealised keystream distribution estimate.

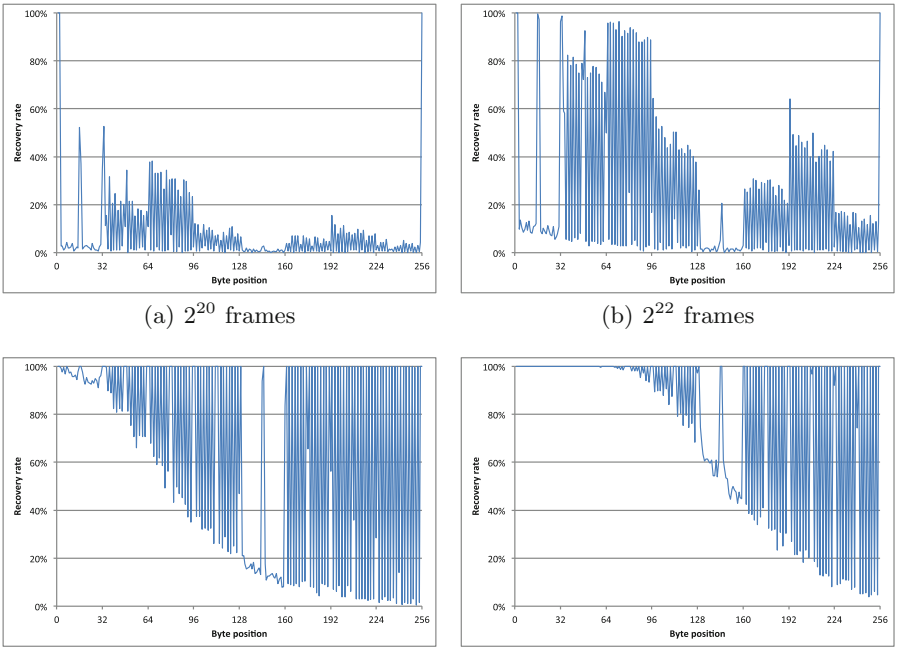
Fig. 10. Success rate for attacks on TKIP based on $(\texttt{TSC}_0, \texttt{TSC}_1)$ pair binning and aggregation of $\texttt{TSC}_0$ values, for both idealised and non-idealised keystream distribution estimates. All success rates are based on 256 runs of each attack using 2^{24} simulated frames.

the effect of noise in the keystream distribution estimate, we furthermore simulated the attacks using an idealised estimate. Specifically, we used a modified set of per-output-byte probabilities $\{p'_{\overline{\texttt{TSC}},r,k}\}_{1 \le r \le 256, \texttt{0x00} \le k \le \texttt{0xFF}}$ for which all probabilities corresponding to a bias below a threshold of four times the standard deviation for a normal distribution were replaced by the average value of these probabilities. All of the simulations were done for 2^{24} frames and each attack was run 256 times. The resulting recovery rates are shown in Fig. 10(a)–(d). We make the following observations:

– The recovery rate for all byte positions improves significantly for the attack based on full $(\texttt{TSC}_0, \texttt{TSC}_1)$ binning when using an idealised keystream distribution estimate. This indicates that the level of noise in our keystream distribution estimate based on 2^{24} RC4 keystreams for each $(\texttt{TSC}_0, \texttt{TSC}_1)$ pairs has an adverse effect on the recovery rate and is too significant for the binning attack to work optimally.

– The recovery rate for the attack based on aggregation over $\texttt{TSC}_0$ values is very similar when using idealised and non-idealised keystream distribution estimates. The recovery rate in the latter case is in fact slightly higher than in the former. This indicates that the idealisation, using a threshold of four times the standard deviation of the normal distribution, removes structure from the keystream distribution estimate that would otherwise improve the recovery rate, and that the level of noise does not have a significant effect on the recovery rate. Note that when aggregating over all $\texttt{TSC}_0$ values, the keystream distribution estimate for each $\texttt{TSC}_1$ value is based on 2^{32} RC4 keystreams.
– The recovery rate for the attack based on aggregation over all $\texttt{TSC}_0$ values is noticeably higher than the recovery rate for the attack based on full $(\texttt{TSC}_0, \texttt{TSC}_1)$ binning, even if using an idealised keystream distribution estimate in the latter case.

Based on the above observations, we decided to study in more detail the attack based on aggregation over all $\texttt{TSC}_0$ values using a non-idealised keystream estimate. Specifically, we ran the simulation of this attack 256 times for $S = 2^{20}$, 2^{22}, 2^{24}, 2^{26}, 2^{28} simulated frames. The resulting recovery rates can be seen in Figs. 10(c) and 11(a)–(d). We observe the following:



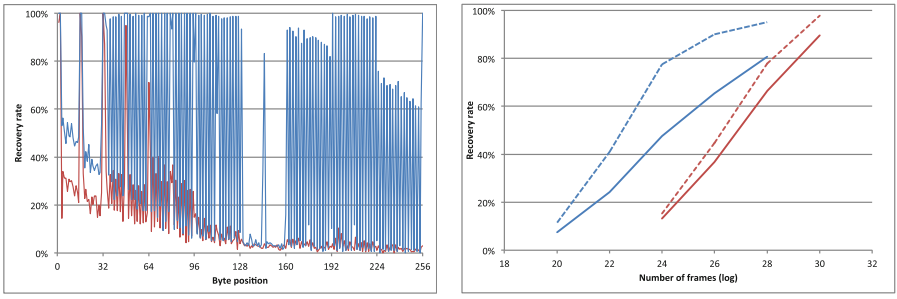
(a) 2^{20} frames (b) 2^{22} frames

(c) 2^{26} frames (d) 2^{28} frames

Fig. 11. Success rate for attack on TKIP based on aggregation of $\texttt{TSC}_0$ values for 256 simulations, each using 2^{20}, 2^{22}, 2^{26}, and 2^{28} frames. See Fig. 10(c) for the success rate for 2^{24} frames.

- Even with as few as $S = 2^{20}$ frames, a few positions of the plaintext are correctly recovered with high probability. In particular, byte positions $1, 2$ and 256 are recovered with a rate of 100 %, whereas positions that are low multiples of 16 are recovered with a rate higher than 50 %.
- With $S = 2^{22}$ frames, 26 positions are recovered with a rate higher than 80 %, and the average recovery rate is 24 %. In comparison, for 2^{24} frames, the attack using fully aggregated biases recovers only 7 positions with a rate higher than 80 % and has an average recovery rate of 13 % (cf. Fig. 9(a)). Furthermore, the recovery rate for even positions is comparable to that of the attack using fully aggregated biases for 2^{26} frames (cf. Fig. 9(b)).
- With $S = 2^{26}$ frames, 146 positions are recovered with a rate higher than 80 %, and the average recovery rate has increased to 65 %. This is similar to the corresponding numbers for the attack using fully aggregated biases, but with 2^{28} frames. The recovery rate for the even positions has furthermore increased to 90 %.
- For up to $S = 2^{26}$ frames, the number of positions recovered with a rate above 80 % and the average recovery rate seems to exceed or match the corresponding numbers for the attack using fully aggregated biases with four times the number of frames. For even positions, the number of frames required in the attack using fully aggregated biases to get comparable recovery rates seems to be a factor of 16 larger.

To enable easy comparison of the different attacks, Fig. 12(a) shows the recovery rate for 2^{24} frames for both the attack using fully aggregated biases and the attack based on aggregation of $\texttt{TSC}_0$ values, and Fig. 12(b) shows the average recovery rates of the two attacks as the number of frames increases. As can be seen from these figures and the comparisons made above, the attack based on aggregation of $\texttt{TSC}_0$ values is noticeably more successful in correctly recovering the correct plaintext bytes, in particular at even plaintext positions.



(a) Recovery rate for 2^{24} frames. (b) Average recovery rate.

Fig. 12. Comparison of attack using fully aggregated biases (red) and attack based on aggregation over all $\texttt{TSC}_0$ values (blue). Figure (a) shows the recovery rates of the two attacks for 256 runs each with 2^{24} simulated frames. Figure (b) shows the average recovery rates of the two attacks. The dashed lines correspond to the average recovery rates for even positions (Color figure online).

6 Practical Impact, Countermeasures and Open Problems

We have shown that plaintext recovery for RC4 in WPA is possible for the first 256 bytes of a frame, provided sufficiently many independent encryptions of the same plaintext are available. Certainly, the security level provided by RC4 is *far* below the strength implied by the 128-bit key in WPA. We are confident that the attacks could be improved further by using more accurate estimates of the TKIP keystream distributions in our full $(\mathtt{TSC}_0, \mathtt{TSC}_1)$ binning attack; obtaining these estimates is simply a matter of computation.

6.1 Practical Impact

Concerning the practical impact of our attacks, we note that WPA frames are quite likely to contain fixed, but *known* bytes, as well as fixed but *unknown* bytes. Examples of the former were already used in keystream recovery attacks against TKIP, as a prelude to MIC key recovery and frame injection attacks—see, for example, [20,23]. The latter would be suitable targets for our attacks and would include fields in IP, UDP and TCP headers, such as source and destination IP addresses, the IP header protocol field, and UDP and TCP port numbers. Another target of potential interest would be passwords or cookies in HTTP traffic (that are not already protected with TLS), with Javascript running in a browser as in [3] providing one possible mechanism to generate the traffic needed in the attacks.

We do not claim that our work enables practical attacks against WPA, in the same way that the work of [20,23] does, for example. Rather our work unveils some fundamental weaknesses in the way in which RC4 is employed in WPA which make it easier to attack in the broadcast setting than should be the case. In this sense, our paper places limits on the security that can be achieved by WPA.

Our work does bear further comparison with previous attacks on WPA, however. In particular, we stress that our attacks are passive, ciphertext-only attacks, with modest ciphertext requirements. This contrasts with the active attacks of [20,23] and the known-plaintext attack of [19]. The active attacks are rate-limited, in that they cannot recover more than 1 byte of plaintext per minute (this is because of peculiarities of the way in which WPA reacts to MAC errors). The attack of [19] requires 2^{38} known plaintexts and has complexity 2^{96}. On the other hand, our attacks have a repeated (but unknown) plaintext requirement and can only target the initial bytes in a frame; furthermore, we only recover plaintext, in contrast to the key-recovery attack of [19].

6.2 Countermeasures

There are some countermeasures to the attacks. These include:

– Discarding the initial keystream bytes output by RC4, as recommended in [12] (but then double-byte-bias attacks like those developed in [3] may still be applicable).

– Changing the manner in which WPA's RC4 keys are computed (but then the analysis of RC4 in TLS from [3] might apply, so security might be increased, but not all known attacks eliminated).
– Abandoning TKIP and switching to CCMP, a confidentiality mode that is based on the CCM authenticated encryption scheme.

Of these, the third is the only one that can be recommended, since the first two would still leave vulnerabilities and would require further changes to the WPA/TKIP specification.

6.3 Open Problems

Open problems suggested by this paper include:

– Carrying out a larger-scale computation to obtain more accurate estimates of the per $(\texttt{TSC}_0, \texttt{TSC}_1)$ pair keystream distributions, and investigating the effect of using these better estimates in our $(\texttt{TSC}_0, \texttt{TSC}_1)$ binning attack.
– Explaining the genesis of the RC4 keystream biases when TKIP keys are used. This has already been completed to some extent for the case of random 128-bit keys (as used in TLS) in [15], and it seems plausible that similar techniques could be deployed for the TKIP case. Indeed, recent progress on this problem has been made in [16]. A full theoretical description of the TKIP biases would obviate the need for extensive computations to establish the keystream distributions.
– Extending the 2-byte plaintext recovery attack of [3] to TKIP. This would, potentially, enable plaintext recovery attacks beyond the first 256 positions in each frame. It is also possible that there are strong dependencies between adjacent pairs of keystream bytes in the initial positions. A large computation would be needed to test this.
– Exploring whether it is possible to combine our attack methods with those of [19] to avoid the onerous known plaintext requirements of those previous attacks, and investigating whether it is possible to improve TKIP $\texttt{TK}$ key recovery attacks further by using TSC-specific biases.
– Studying other applications of RC4 in which the RC4 key is changed frequently.

Acknowledgements. We thank Jon Hart of the ISG at RHUL for his assistance with computing infrastructure. The research of the authors was supported by an EPSRC Leadership Fellowship, EP/H005455/1.

References

1. Wireless LAN medium access control (MAC) and physical layer (PHY) specification (1997)
2. Wireless LAN medium access control (MAC) and physical layer (PHY) specification: Amendment 6: Medium access control (MAC) security enhancements (2004)

3. AlFardan, N.J., Bernstein, D.J., Paterson, K.G., Poettering, B., Schuldt, J.C.N.: On the security of RC4 in TLS. In: USENIX Security (2013). https://www.usenix.org/conference/usenixsecurity13/security-rc4-tls
4. Borisov, N., Goldberg, I., Wagner, D.: Intercepting mobile communications: the insecurity of 802.11. In: Rose, C. (ed.) MOBICOM, pp. 180–189. ACM, New York (2001)
5. Fluhrer, S.R., Mantin, I., Shamir, A.: Weaknesses in the key scheduling algorithm of RC4. In: Vaudenay, S., Youssef, A.M. (eds.) SAC 2001. LNCS, vol. 2259, pp. 1–24. Springer, Heidelberg (2001)
6. Fluhrer, S.R., McGrew, D.: Statistical analysis of the alleged RC4 keystream generator. In: Schneier, B. (ed.) FSE 2000. LNCS, vol. 1978, pp. 19–30. Springer, Heidelberg (2001)
7. Halvorsen, F.M., Haugen, O., Eian, M., Mjølsnes, S.F.: An improved attack on TKIP. In: Jøsang, A., Maseng, T., Knapskog, S.J. (eds.) NordSec 2009. LNCS, vol. 5838, pp. 120–132. Springer, Heidelberg (2009)
8. Jaganathan, K., Zhu, L., Brezak, J.: The RC4-HMAC Kerberos Encryption Types Used by Microsoft Windows. RFC 4757 (Informational), December 2006. http://www.ietf.org/rfc/rfc4757.txt
9. Maitra, S., Paul, G., Sen Gupta, S.: Attack on broadcast RC4 revisited. In: Joux, A. (ed.) FSE 2011. LNCS, vol. 6733, pp. 199–217. Springer, Heidelberg (2011)
10. Mantin, I.: Predicting and distinguishing attacks on RC4 keystream generator. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 491–506. Springer, Heidelberg (2005)
11. Mantin, I., Shamir, A.: A practical attack on broadcast RC4. In: Matsui, M. (ed.) FSE 2001. LNCS, vol. 2355, pp. 152–164. Springer, Heidelberg (2002)
12. Mironov, I.: (Not so) random shuffles of RC4. In: Yung, M. (ed.) CRYPTO 2002. LNCS, vol. 2442, pp. 304–319. Springer, Heidelberg (2002)
13. Moen, V., Raddum, H., Hole, K.J.: Weaknesses in the temporal key hash of WPA. Mob. Comput. Commun. Rev. **8**(2), 76–83 (2004)
14. Morii, M., Todo, Y.: Cryptanalysis for RC4 and breaking WEP/WPA-TKIP. IEICE Trans. **94–D**(11), 2087–2094 (2011)
15. Sarkar, S., Sen Gupta, S., Paul, G., Maitra, S.: Proving TLS-attack related open biases of RC4. Cryptology ePrint Archive, Report 2013/502 (2013). http://eprint.iacr.org/
16. Sen Gupta, S., Maitra, S., Meier, W., Paul, G., Sarkar, S.: Some results on RC4 in WPA. Cryptology ePrint Archive, Report 2013/476 (2013). http://eprint.iacr.org/
17. Sen Gupta, S., Maitra, S., Paul, G., Sarkar, S.: (Non-) random sequences from (non-) random permutations - analysis of RC4 stream cipher. J. Cryptol. **27**(1), 67–108 (2014)
18. Sepehrdad, P., Vaudenay, S., Vuagnoux, M.: Discovery and exploitation of new biases in RC4. In: Biryukov, A., Gong, G., Stinson, D.R. (eds.) SAC 2010. LNCS, vol. 6544, pp. 74–91. Springer, Heidelberg (2011)
19. Sepehrdad, P., Vaudenay, S., Vuagnoux, M.: Statistical attack on RC4. In: Paterson, K.G. (ed.) EUROCRYPT 2011. LNCS, vol. 6632, pp. 343–363. Springer, Heidelberg (2011)
20. Tews, E., Beck, M.: Practical attacks against WEP and WPA. In: Basin, D.A., Capkun, S., Lee, W. (eds.) WISEC, pp. 79–86. ACM (2009)
21. Tews, E., Weinmann, R.-P., Pyshkin, A.: Breaking 104 bit WEP in less than 60 seconds. In: Kim, S., Yung, M., Lee, H.-W. (eds.) WISA 2007. LNCS, vol. 4867, pp. 188–202. Springer, Heidelberg (2008)

22. Todo, Y., Ozawa, Y., Ohigashi, T., Morii, M.: Falsification attacks against WPA-TKIP in a realistic environment. IEICE Trans. **95–D**(2), 588–595 (2012)
23. Vanhoef, M., Piessens, F.: Practical verification of WPA-TKIP vulnerabilities. In: Chen, K., Xie, Q., Qiu, W., Li, N., Tzeng, W.G. (eds.) ASIACCS, pp. 427–436. ACM (2013)
24. Vaudenay, S., Vuagnoux, M.: Passive–only key recovery attacks on RC4. In: Adams, C., Miri, A., Wiener, M. (eds.) SAC 2007. LNCS, vol. 4876, pp. 344–359. Springer, Heidelberg (2007)

Dependence in IV-Related Bytes of RC4 Key Enhances Vulnerabilities in WPA

Sourav Sen Gupta[1(✉)], Subhamoy Maitra[1], Willi Meier[2], Goutam Paul[1], and Santanu Sarkar[3]

[1] Indian Statistical Institute, Kolkata, India
sg.sourav@gmail.com,{subho,goutam.paul}@isical.ac.in
[2] FHNW, Windisch, Switzerland
willi.meier@fhnw.ch
[3] Chennai Mathematical Institute, Chennai, India
sarkar.santanu.bir@gmail.com

Abstract. The first three bytes of the RC4 key in WPA are public as they are derived from the public parameter IV, and this derivation leads to a strong mutual dependence between the first two bytes of the RC4 key. In this paper, we provide a disciplined study of RC4 biases resulting specifically in such a scenario. Motivated by the work of AlFardan et al. (2013), we first prove the interesting sawtooth distribution of the first byte in WPA and the similar nature for the biases in the initial keystream bytes towards zero. As we note, this sawtooth characteristics of these biases surface due to the dependence of the first two bytes of the RC4 key in WPA, both derived from the same byte of the IV. Our result on the nature of the first keystream byte provides a significantly improved distinguisher for RC4 used in WPA than what had been presented by Sepehrdad et al. (2011–2012). Further, we revisit the correlation of initial keystream bytes in WPA to the first three bytes of the RC4 key. As these bytes are known from the IV, one can obtain new as well as significantly improved biases in WPA than the absolute biases exploited earlier by AlFardan et al. or Isobe et al. We notice that the correlations of the keystream bytes with publicly known IV values of WPA potentially strengthen the practical plaintext recovery attack on the protocol.

Keywords: RC4 · WPA · Bias · Key correlation · Plaintext recovery

1 Introduction

The RC4 stream cipher and several modifications thereof (incorporated in various security protocols) have undergone rigorous analysis in cryptographic literature. The importance and timeliness of this topic is evident from the rich history

S. Sen Gupta—Supported by DRDO sponsored project Centre of Excellence in Cryptology (CoEC), under MOC ERIP/ER/1009002/M/01/1319/788/D(R&D) of ER&IPR, DRDO.

C. Cid and C. Rechberger (Eds.): FSE 2014, LNCS 8540, pp. 350–369, 2015.
DOI: 10.1007/978-3-662-46706-0_18

of research in RC4 over the last two decades. Among the several directions of cryptanalytic research in this area, the two most important aspects have been

1. correlation between the keystream bytes with absolute values, and
2. correlation between the keystream bytes with the Key and/or IV.

The results under item 1 have been extensively used in the broadcast attack model, and some important results in this area can be found in [1,6,8,9,11,19]. These biases directly work on the generic RC4 cipher [6] as well when RC4 is used in protocols like WPA and TLS [1]. In particular, the work of [1] received a lot of attention due to its impact on commercial protocols.

The results under item 2 explains how the RC4 keystream bytes may leak information regarding the secret key bytes. While there exist extensive research results in this area [8,13–15,19], no convincing key-recovery attack is yet available on RC4 using these biases. However, these biases work quite well in attacking protocols where some part of the RC4 key is derived from the public IV, as in the case of WEP [3,4,7,21,22,24].

To resist such attacks against WEP, the WPA [5] protocol had been proposed, where an incremental change in the IV results in a convoluted transformation of the remaining portion of the RC4 key. The two most recent and prominent attacks against WPA have been proposed by Sepehrdad et al. [20] and AlFardan et al. [1]. While the attack of [1] is based on the broadcast model for plaintext recovery, the work of [20] exploits certain weaknesses in the WPA key schedule to mount a key recovery attack with complexity 2^{96}, less than the exhaustive key search effort of 2^{104}. Before we proceed further to explain our contributions in this paper, let us describe RC4 and its usage in the WPA protocol.

We omit the mention of TKIP and refer to the WPA/TKIP protocol simply as WPA in this paper. In addition, we abuse the notation to refer to the instantiation of RC4 in this protocol as WPA, in contrast to standalone RC4.

1.1 Description of RC4

The RC4 cipher consists of a Key Scheduling Algorithm (KSA) and a Pseudo-random Generation Algorithm (PRGA). The internal state of RC4 is obtained as a permutation of $N = 256$ bytes, and the KSA produces the initial pseudorandom permutation of RC4 by scrambling an identity permutation using the secret key k. The secret key k of RC4 is of length typically between 5 to 32 bytes, which generates the expanded key K of length $N = 256$ bytes by simple repetition. If the length of the secret key $k = k_0, \ldots, k_{l-1}$ is l bytes (typically $5 \leq l \leq 32$), then the expanded key K is constructed as $K[i] = k_{i \bmod l}$ for $0 \leq i \leq N-1$. The initial permutation produced by the KSA acts as an input to the next procedure PRGA that generates the keystream, as depicted in Fig. 1.

For round $r = 1, 2, \ldots$ of RC4 PRGA, we denote the indices by i_r, j_r, the keystream output byte by Z_r, and the permutations before and after the swap by S_{r-1} and S_r respectively. All additions (subtractions) in context of RC4 are to be considered as 'addition (subtraction) modulo N', and all equalities in context of RC4 are to be considered as 'congruent modulo N'.

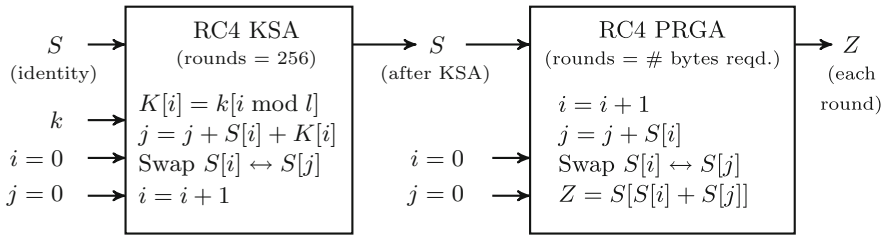


Fig. 1. Description of RC4 stream cipher.

1.2 Description of WPA

IEEE 802.11 standard protocol for WiFi security used to be Wired Equivalent Privacy (WEP), which was replaced by Wi-Fi Protected Access (WPA) in 2004. Both WEP and WPA use RC4 as their core cipher, and the WPA protocol can be thought of as a wrapper on top of WEP to provide good key mixing features. WPA introduces a key hashing module in the original WEP design to defend against the Fluhrer, Mantin and Shamir attack [4]. It also includes a message integrity feature and a key management scheme to avoid key reuse.

TKIP Key Schedule. WPA uses a 16-byte secret key for RC4 PRNG, the core encryption module of the system. This RC4 secret key RC4KEY is generated through a key schedule procedure known as TKIP [5], which takes as input a 128-bit *temporal key* TK (shared between the parties), transmitter's 48-bit MAC address TA and a 48-bit *initialization vector* IV, and passes those through two phases to obtain the final RC4 secret key.

In Phase 1, a 80-bit key P1K is generated from TK, TA and IV32, the upper 32 bits of the IV, using an unbalanced Feistel cipher with 80-bit block and 128-bit key structure. In Phase 2, the 128-bit RC4KEY is generated from TK, P1K (from Phase 1) and IV16, the lower 16 bits of the IV. In this phase, TK and P1K are mixed (using a temporary key PPK) to construct the last 104 bits (13 bytes) of the RC4KEY, and the first 24 bits (3 bytes) of the RC4KEY are constructed directly from the IV16, as follows [5, Annex H.1].

```
RC4KEY[0] = Hi8(IV16);                    /* top byte of IV16 */
RC4KEY[1] = (Hi8(IV16) | 0x20) & 0x7F;    /* avoid FMS attack */
RC4KEY[2] = Lo8(IV16);                    /* low byte of IV16 */
```

In the above expression, Hi8(IV16) and Lo8(IV16) indicate the top and lower bytes of IV16, respectively. RC4KEY[0] and RC4KEY[2] are simply two parts of the counter IV16, while RC4KEY[1] is purposefully constructed to avoid the known WEP attack by Fluhrer, Mantin and Shamir [4]. Once the 128-byte (16-byte) RC4KEY is prepared, it is directly used for encryption in the RC4 PRNG core of the protocol.

1.3 Contributions of This Paper

There is a growing concern regarding how far we should study the combinatorial nature of RC4 and protocols based on it. However, we can not help but notice the glaring implications of such studies in mounting practical attacks on commercial protocols that still handle a bulk of everyday network traffic. In this backdrop, we present the motivation and contribution of our paper as follows.

Motivation. To the best of our knowledge, the dependence of the first two bytes of the RC4 key, constructed from the public parameter IV during WPA key schedule, has not been studied thoroughly from a combinatorial viewpoint. We draw our motivation from two important questions in this direction.

1. How do the biases of keystream bytes towards absolute values differ for RC4 in WPA compared to those in case of generic RC4?
2. Are there any exploitable correlations between the keystream bytes and the first three key bytes of RC4 derived from the IV in WPA?

Contribution. Our results provide the first disciplined study of keystream non-randomness in RC4 when used in WPA. The study contains explanation of existing biases as well as discovery of new ones. The results have diverse applications in different cryptanalytic results, ranging from the best WPA distinguisher to improved broadcast attack against WPA.

Specific Outcomes of our First Motivation. We provide theoretical justification of some experimental observations on WPA, made by AlFardan et al. [1], to obtain further insight into such observations.

- In Sect. 2.3, we derive the complete sawtooth distribution of the first keystream byte Z_1 when RC4 is executed with IV's as in WPA.
- The biases in Z_1 gives a method to distinguish the keystream of WPA from that of generic RC4, with a packet complexity of approximately 2^{19}. Note that WPA may be considered as a 'mode of operation' for RC4, and our observation shows that this mode statistically deviates from the core cipher, where the deviation is visible with a considerably less number of packets. The previously known distinguisher of [20], first presented in Eurocrypt 2011, achieves a 0.5 probability of success in distinguishing WPA from generic RC4 with time complexity 2^{43} and packet complexity 2^{40}. Later in [18], the distinguisher was improved to achieve 0.5 probability of success in distinguishing WPA with time complexity 2^{42} and packet complexity 2^{42}.
- In Sect. 2.4, we show how the initial keystream bytes $Z_3, \ldots, Z_{255}$ of WPA are biased towards zero following a similar sawtooth pattern, and in Sect. 2.5, we provide a theoretical estimate for $(Z_r = r)$ better than [6].

Specific Outcomes of our Second Motivation. All biases of the keystream bytes in WPA presented in [1] are correlated to absolute values in $[0, 255]$, and the experimental study discovers that they are mostly of the order of $1/N^2$ over the probability of random association $1/N$. Indeed these are not of the same level

as the bias in the event $(Z_2 = 0)$, which is of the order $1/N$ over the probability of random association $1/N$, as proved in [11]. However, it is well known that there are quite a few significant biases of the keystream bytes with the initial key bytes of RC4 [7,8,13–15,17,19]. The first three bytes of the RC4 key in WPA are derived from the public parameter IV, and thus the correlation of keystream bytes with any combination of the first three RC4 key bytes can be successfully exploited in broadcast attack against WPA. Our investigations in this direction reveal the following results.

- There exist high biases in the keystream bytes Z_1, Z_2, Z_3, Z_{256}, Z_{257} towards the first three 'public' (IV-derived) bytes of the RC4 key in WPA. For the first time in the literature, we discover such hugely significant biases, matching the order of the $(Z_2 = 0)$ bias [11], even in the case of WPA.
- In a broadcast setting, we could recover the aforesaid bytes of the plaintext with probability close to 1 using only 2^{21} samples, in contrast with the existing works [1,6] that require 2^{30} samples for the same bytes.
- We explore some new biases in this line and present a detailed study on the correlations of the keystream bytes with different IV combinations in WPA.
- We also discover a new absolute bias at the keystream byte Z_{259}, the farthest known so far among the initial keystream bytes to have a significant bias.

An independent work [12] in a similar direction is to appear in FSE 2014.

2 Biases in WPA Resulting from TKIP Key Schedule

The first three bytes of the RC4 key in WPA is derived as in Eq. (1).

$$\begin{aligned} K[0] &= (\mathsf{IV16} >> 8) \,\&\, \mathsf{0xFF} & K[2] &= \mathsf{IV16} \,\&\, \mathsf{0xFF} \\ K[1] &= ((\mathsf{IV16} >> 8) \mid \mathsf{0x20}) \,\&\, \mathsf{0x7F} \end{aligned} \tag{1}$$

Note that a 2-byte IV16 is expanded to the initial 3 bytes of the key (Fig. 2), and the first two key bytes $K[0]$ and $K[1]$ have 6 bits in common, apart from the two fixed bits in $K[1]$. The third key byte $K[2]$ is independent of the first two bytes of the key. Thus, TKIP can generate only 2^{16}, and not 2^{24}, distinct values for the first 3 bytes of the RC4 secret key – a loss in entropy that we believe may result into some non-random behavior in the initial phases of the cipher.

2.1 Bias in $K[0] + K[1]$ for WPA

As $K[0]$ and $K[1]$ share 6 bits from the common source Hi8(IV16), we first take a look at their sum, $K[0] + K[1]$, for potential non-randomness. We notice that

1. $K[0] + K[1]$ must always be *even*, as $K[0]$ and $K[1]$ have the same LSB.
2. $K[1]$ can never exceed 127 as its MSB is 0. It can not even attain all possible values below 127, as its 6-th bit (from LSB side) is fixed at 1.
3. Values of $K[1]$ and $K[0] + K[1]$ strictly depend on the value/range of $K[0]$.

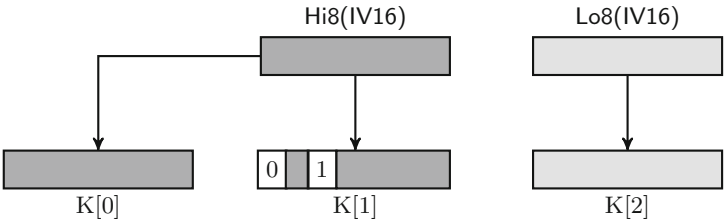


Fig. 2. Expansion of WPA IV16 into the first three bytes of the RC4 key.

These restrictions result in corresponding conditions on the range of $K[1]$ and $K[0]+K[1]$, depending on the range of $K[0]$. The complete set of conditions on the respective ranges is shown in Table 1, which results in a consolidated probability distribution of $K[0] + K[1]$ as described in Theorem 1.

Theorem 1. *The probability distribution of the sum of first two bytes of the RC4 key generated by TKIP key schedule in WPA, i.e., the distribution of* $\Pr(K[0] + K[1] = v)$ *for* $v = 0, 1, \ldots, 255$*, is as in Table 1:*

$$\begin{aligned}
&\Pr(K[0] + K[1] = v) = 0 && \text{if } v \text{ is odd;}\\
&\Pr(K[0] + K[1] = v) = 0 && \text{if } v \text{ is even and } v \in [0, 31] \cup [128, 159];\\
&\Pr(K[0] + K[1] = v) = 2/256 && \text{if } v \text{ is even and}\\
& && v \in [32, 63] \cup [96, 127] \cup [160, 191] \cup [224, 255];\\
&\Pr(K[0] + K[1] = v) = 4/256 && \text{if } v \text{ is even and } v \in [64, 95] \cup [192, 223].
\end{aligned}$$

Proof. The value of $K[0] + K[1]$ is always even, as discussed earlier. The value and range of $K[1]$, and hence that of $K[0]+K[1]$, depends on the range of $K[0]$;

Table 1. Probability distribution of $K[0] + K[1]$ resulting due to TKIP key schedule.

$K[0]$	$K[1]$ (depends on $K[0]$)		$K[0] + K[1]$ (only even)	
Range	Value	Range	Value	Range
0 – 31	$K[0] + 32$	32 – 63	$2K[0] + 32$	32 – 95
32 – 63	$K[0]$	32 – 63	$2K[0]$	64 – 127
64 – 95	$K[0] + 32$	96 – 127	$2K[0] + 32$	160 – 223
96 – 127	$K[0]$	96 – 127	$2K[0]$	192 – 255
128 – 159	$K[0] - 96$	32 – 63	$2K[0] - 96$	160 – 233
160 – 191	$K[0] - 128$	32 – 63	$2K[0] - 128$	192 – 255
192 – 223	$K[0] - 96$	96 – 127	$2K[0] - 96$	32 – 95
224 – 255	$K[0] - 128$	96 – 127	$2K[0] - 128$	64 – 127

$K[0] + K[1]$ (only even)	Prob. (0 if odd)
0 – 31	0
32 – 63	2/256
64 – 95	4/256
96 – 127	2/256
128 – 159	0
160 – 191	2/256
192 – 223	4/256
224 – 255	2/256

shown in Table 1. The probability distribution of $K[0]+K[1]$ may be calculated directly from this dependence pattern; also shown in Table 1. One may check

$$\underbrace{(128\times 0)}_{\text{odd values}}+\Bigg(16\times 0+16\times\frac{2}{256}+16\times\frac{4}{256} \\ +16\times\frac{2}{256}+16\times 0+16\times\frac{2}{256}+16\times\frac{4}{256}+16\times\frac{2}{256}\Bigg)=1,$$

to validate the consistency of the probability distribution of $K[0]+K[1]$. □

2.2 Bias in RC4 PRGA Initial Permutation S_0 for WPA

In 2007, Paul and Maitra [13] proved the famous Roos' biases [15], which state that the initial bytes of the permutation S_0 are biased towards the secret key bytes. $S_0[0]$ is biased towards $K[0]$, which is uniformly distributed, identical to the lower half of the counter IV16. For $S_0[1]$ however, we get the following result.

Theorem 2. *In case of WPA, the probability distribution of* $(S_0[1]=v)$ *for* $v=0,1,\ldots,N-1$, *after the completion of KSA, is given as*

$$\Pr(S_0[1]=v)=\alpha\cdot\Pr(K[0]+K[1]=v-1) \\ +(1-\alpha)\cdot(1-\Pr(K[0]+K[1]=v-1))\cdot\Pr(S_0[1]=v)_{RC4} \\ +\frac{(1-\alpha)}{N-1}\cdot\sum_{x\neq v}\Pr(K[0]+K[1]=x-1)\cdot\Pr(S_0[1]=x)_{RC4},$$

where $\alpha=1/N+(1-1/N)^{N+2}$, *and the probability terms* $\Pr(S_0[1]=v)_{RC4}$ *and* $\Pr(S_0[1]=x)_{RC4}$ *refer to the corresponding values in generic RC4.*

Proof. From the proof of Roos' biases in [13], we know that the initial permutation byte $S_0[1]$ is biased towards $K[0]+K[1]+1$ with a probability $\Pr(S_0[1]=K[0]+K[1]+1)\approx 1/N+(1-1/N)^{N+2}=\alpha$, say. Thus we write the probability distribution of $S_0[1]=v$ in case of WPA as follows.

$$\Pr(S_0[1]=v)=\Pr(S_0[1]=v\ \wedge\ K[0]+K[1]+1=v) \\ +\sum_{x\neq v}\Pr(S_0[1]=v\ \wedge\ K[0]+K[1]+1=x)$$

The first event $(S_0[1]=v\ \wedge\ K[0]+K[1]+1=v)$ occurs if and only if the independent events $(S_0[1]=K[0]+K[1]+1)$ and $(K[0]+K[1]=v-1)$ occur simultaneously. This happens with probability $\alpha\cdot\Pr(K[0]+K[1]=v-1)$ where α is due to Roos' bias, and the second term is obtained from Theorem 1.

On the other hand, the event $(S_0[1]=v\ \wedge\ K[0]+K[1]+1=x)$ for $x\neq v$ may be further decomposed as follows

$$\Pr(S_0[1]=v\ \wedge\ S_0[1]=K[0]+K[1]+1\ \wedge\ K[0]+K[1]+1=x) \\ +\Pr(S_0[1]=v\ \wedge\ S_0[1]\neq K[0]+K[1]+1\ \wedge\ K[0]+K[1]+1=x).$$

The first term denotes an impossible condition (probability 0), and the second term can be computed as $\Pr(K[0]+K[1]=x-1)\cdot\Pr(S_0[1]\neq K[0]+K[1]+1)\cdot\Pr(S_0[1]=v \mid S_0[1]\neq x)$, that is, as

$$(1-\alpha)\cdot\Pr(K[0]+K[1]=x-1)\cdot(\Pr(S_0[1]=v)_{RC4}+\Pr(S_0[1]=x)_{RC4}/(N-1)),$$

where we assume that $(S_0[1]=v)$ and $(S_0[1]=x)$ occur exactly as in generic RC4 when $S_0[1]\neq K[0]+K[1]+1$, with appropriate probability normalization. We get the result after due simplification of the summation over $x\neq v$. □

For $N=256$, as in WPA and RC4, we get $\alpha\approx 0.368$ in Theorem 2. The probabilities $\Pr(K[0]+K[1]=v-1)$ and $\Pr(K[0]+K[1]=x-1)$ are taken from Theorem 1, and the probabilities $\Pr(S_0[1]=v)_{RC4}$ and $\Pr(S_0[1]=x)_{RC4}$ are taken from Proposition 1, derived in [10, Theorem 6.2.1].

Proposition 1 (from [10]). *After RC4 KSA, for* $0\leq u\leq N-1$, $0\leq v\leq N-1$,

$$\Pr(S_0[u]=v)=\begin{cases}\frac{1}{N}\left(\left(\frac{N-1}{N}\right)^v+\left(1-\left(\frac{N-1}{N}\right)^v\right)\left(\frac{N-1}{N}\right)^{N-u-1}\right), & \text{if } v\leq u;\\ \frac{1}{N}\left(\left(\frac{N-1}{N}\right)^{N-u-1}+\left(\frac{N-1}{N}\right)^v\right), & \text{if } v>u.\end{cases}$$

The theoretical distribution of $S_0[1]$ in WPA, thus produced from Theorem 2, is shown in Fig. 3. This distribution closely matches our experimental data, and differs significantly from the one for generic RC4 (as derived in [10]).

2.3 Bias in the First Keystream Byte Z_1 of WPA

Recall that in the first round of RC4 PRGA, the initial permutation entry $S_0[1]$ serves as $j_1=S_0[i_1]=S_0[1]$, and plays an important role in determining the

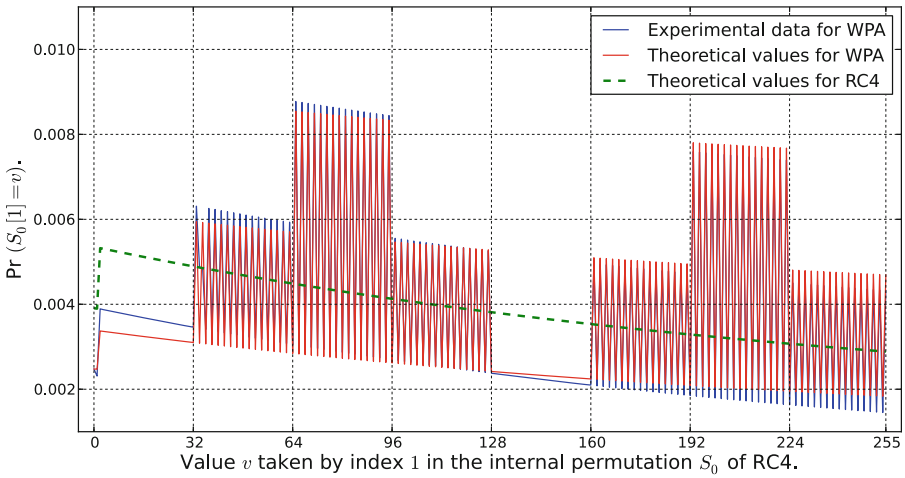


Fig. 3. Theoretical plot for $\Pr(S_0[1]=v)$ for RC4 and WPA, where $v=0,\ldots,255$.

first keystream byte $Z_1 = S_1[S_1[i_1] + S_1[j_1]] = S_1[S_0[S_0[1]] + S_0[1]]$. In fact, we know that $S_0[1]$ is prominent in the distribution of Z_1 proved by Sen Gupta et al. in [17, Theorem 13]. We reproduce the distribution as follows.

Proposition 2 (from [17]). *The probability distribution of the first output byte of RC4 keystream is as follows, where* $v \in \{0, \ldots, N-1\}$, $\mathcal{L}_v = \{0, 1, \ldots, N-1\} \setminus \{1, v\}$ *and* $\mathcal{T}_{v,X} = \{0, 1, \ldots, N-1\} \setminus \{0, X, 1-X, v\}$.

$$\Pr(Z_1 = v) = Q_v + \sum_{X \in \mathcal{L}_v} \sum_{Y \in \mathcal{T}_{v,X}} \Pr(S_0[1] = X \wedge S_0[X] = Y \wedge S_0[X+Y] = v);$$

$$Q_v = \begin{cases} \Pr(S_0[1] = 1 \ \wedge \ S_0[2] = 0), & \text{if } v = 0; \\ \Pr(S_0[1] = 0 \ \wedge \ S_0[0] = 1), & \text{if } v = 1; \\ \Pr(S_0[1] = 1 \ \wedge \ S_0[2] = v) + \Pr(S_0[1] = v \ \wedge \ S_0[v] = 0) & \\ \quad + \Pr(S_0[1] = 1 - v \ \wedge \ S_0[1-v] = v), & \text{otherwise.} \end{cases}$$

We consider two cases while computing the numeric values of $\Pr(Z_1 = v)$. If the initial permutation S_0 of RC4 PRGA is constructed from the regular KSA with random key, the probabilities $\Pr(S_0[u] = v)$ closely follow the distribution proved by Mantin in [10, Theorem 6.2.1]. However, if the initial permutation S_0 originates from RC4 KSA using TKIP-generated keys, as in the case with WPA, then $\Pr(S_0[1] = v)$ must be computed using Theorem 2, including its idiosyncratic biases for WPA shown in Fig. 3.

We compute the exact probabilities $\Pr(Z_1 = v)$ for RC4 and WPA using the estimation strategy of joint probabilities proposed in [17]; particularly estimating the joint probabilities $\Pr(S_0[X] = A \ \wedge \ S_0[Y] = B)$ as $\Pr(S_0[X] = A) \cdot (\Pr(S_0[Y] = B) + \Pr(S_0[Y] = A)/(N-1))$. The distribution of $S_0[1] = v$ is considered independently in each case. This results in two different distributions of Z_1; one for generic RC4 (same as [17]) and the other for RC4 in WPA.

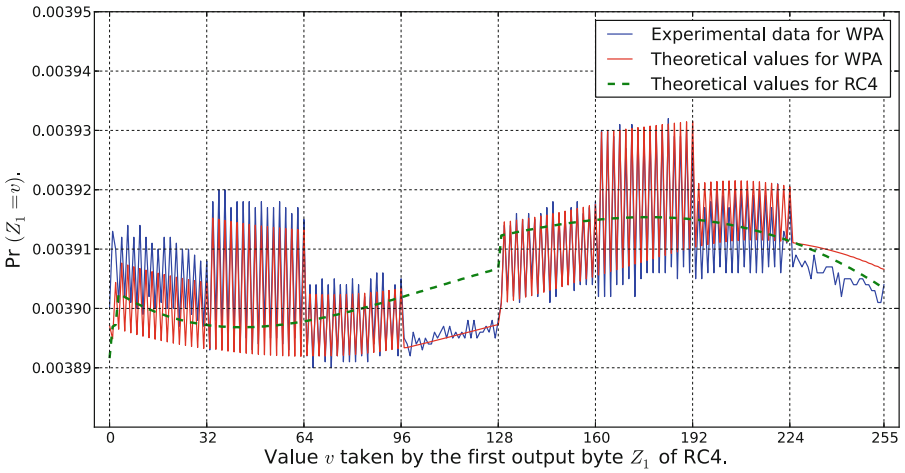


Fig. 4. Theoretical plot for $\Pr(Z_1 = v)$ for RC4 and WPA, where $v = 0, \ldots, 255$.

Figure 4 displays the two distributions, clearly pointing out the bias resulting in the PRGA as a result of TKIP key schedule, and shows that the theoretical distribution for WPA closely matches our experimental data.

Note that the patterns of these two theoretical distributions closely match the recent experimental observations of AlFardan et al. [1] (Fig. 10(a) in the full online version of the paper). The only difference is that there exist keylength dependent spikes at $Z_1 = 129$ for the observations in [1], as the experiments were done using 16-byte keys; whereas in our theoretical analysis, we disregard the keylength dependence altogether, and prove a general distribution of Z_1.

In fact, if WPA had employed RC4 with full-length 256-byte secret keys, where the first three bytes of the key $K[0], K[1], K[2]$ were constructed from the IV using TKIP key schedule principle (as in Eq. (1)), the pattern of the bias in Z_1 for WPA would have been the same. We have independently verified our theoretical results through experiments involving secret keys of various lengths.

Distinguishing WPA. We attempt to combine the values of Z_1 in suitable subsets of the support interval $\{0, 1, \ldots, 255\}$ to construct a distinguisher between WPA and generic RC4. The structure of the event considered for distinguishing WPA from RC4 in this case is '$e_S : (Z_1 \in S)$ where $S \subseteq \{0, 1, \ldots, 255\}$'. The subset S may be quite large, and thus the base probability $p = \Pr(e_S)$ in either distribution is not essentially small. In such a case, the distinguisher complexity may be estimated as $O(\frac{1-p}{pq^2})$.

Now we may define a suitable set S for the target distinguishing event. As most of higher biases are for *even* values of the first byte, we assume that the distributions of WPA and RC4 differ the most in cases when Z_1 takes an even value. Based on this intuition, we pick the set S as the set of all even values $\{0, 2, 4, \ldots, 254\}$ within the range; thus defining the distinguishing event as

$$e_S \ : \ (Z_1 = 2k \text{ for } k = 0, 1, \ldots, 127).$$

From our theoretical results on the distribution of Z_1 in WPA and RC4, as proved in Sect. 2.3, we estimate the following probabilities:

$$\left.\begin{aligned} p = \Pr(e_S) \text{ in RC4 } \& \approx 0.4999946 \\ p(1+q) = \Pr(e_S) \text{ in WPA } \& \approx 0.5007041 \end{aligned}\right\} \Rightarrow \quad q \approx 0.001419 \approx 0.363/N.$$

For $N = 256$, we require an estimated $8N^2 = 2^{19}$ keystream packets to distinguish WPA from generic RC4 with more than 70 % probability of success. This is the best distinguisher of WPA to date, improving the previous distinguishers of packet complexity more than 2^{40}, identified by Sepehrdad et al. [18,20].

It may be noted that some distinguishers of RC4 (compared to uniform random generators) remain equally effective in its WPA 'mode of operation', like the distinguisher based on $(Z_2 = 0)$. However, the sawtooth pattern of Z_1 is unique to WPA, and is not present in original RC4.

2.4 Bias Towards Zero in Bytes $Z_3, \ldots, Z_{255}$ of WPA

We extend the effect of the bias in S_0 of WPA to the biases in the initial keystream bytes towards zero. Maitra et al. [9] proved the biases of the initial keystream bytes $Z_3, \ldots, Z_{255}$ towards zero, and we reproduce their result from [17, Theorem14] in Proposition 3, as follows.

Proposition 3 (from [17]). *For RC4 PRGA rounds $3 \leq r \leq N-1$, the probability that $Z_r = 0$ is given by:*

$$\Pr(Z_r = 0) \approx \frac{1}{N} + \frac{c_r}{N^2}, \quad \textit{where}$$

$$c_r = \begin{cases} \frac{N}{N-1}\left(N \cdot \Pr(S_{r-1}[r] = r) - 1\right) - \frac{N-2}{N-1}, & \textit{for } r = 3; \\ \frac{N}{N-1}\left(N \cdot \Pr(S_{r-1}[r] = r) - 1\right), & \textit{otherwise.} \end{cases}$$

In [17], the computation of $\Pr(Z_r = 0)$ depended on the computation of $\Pr(S_{r-1}[r] = r)$, which in turn required the distributions of initial permutations S_0 and S_1 of RC4 PRGA (details in [17, Corollary 2] and [17, Lemma 1]).

We consider two cases – one in which the initial permutation S_0 is generated by generic RC4 KSA using random keys, and the other where S_0 is biased (Fig. 3) for using RC4 with keys originating from TKIP. These two cases produce two different distributions of $\Pr(Z_r = 0)$ for $r = 3, \ldots, 255$. The patterns closely match the experimental observations of AlFardan et al. [1] (Fig. 11 in the full online version of the paper) as well as our experimental data, as shown in Fig. 5.

2.5 Bias in $(Z_r = R)$ for WPA

Significant biases in the event $(Z_r = r)$ for $r = 3, \ldots, 255$ have recently surfaced in the context of plaintext recovery attack on RC4 [1,6], and these biases are

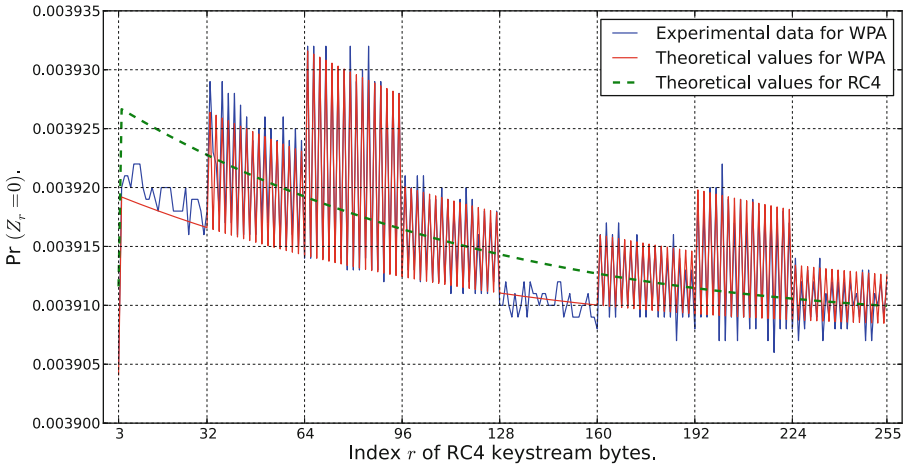


Fig. 5. Theoretical plot for $\Pr(Z_r = 0)$ for RC4 and WPA, where $r = 3, \ldots, 255$.

found to be more prominent than the previous ones for certain values of r. Isobe et al. identified these biases and attempted a proof in [6, Theorem 8], but the estimates did not 'exactly coincide with the experimental values'. Considering the significance of these biases in cryptanalysis of RC4, we explore an alternative avenue to estimate them, as detailed in Theorem 3.

Theorem 3. *For RC4 PRGA rounds $3 \le r \le N-1$, the probability that $Z_r = r$ is approximately*

$$\Pr(Z_r = r) = \frac{1}{N} + \Pr(S_0[1] = r) \cdot \frac{1}{N}\left(1 - \frac{1}{N}\right)\left(1 - \frac{r-2}{N}\right)\left(1 - \frac{2}{N}\right)^{r-3}.$$

Proof. The major path leading to the target event is as follows.

- Suppose that $S_0[1] = r$, i.e., $j_1 = r$ and $j_2 = r + S_1[2]$. This ensures that $S_1[r] = r$ after the first round of PRGA, and $S_2[j_2] = S_1[2]$ after the second.
- Suppose that $j_2 \neq 3, \ldots, r$, which occurs with probability $\left(1 - \frac{r-2}{N}\right)$. This ensures that i_r does not touch either of the locations r or j_2 till round $r-1$.
- Suppose that none of the indices $j_3, \ldots, j_{r-1}$ touches either of the locations r or j_2. This happens with probability $\left(1 - \frac{2}{N}\right)^{r-3}$ as $j_2 \neq r$, and ensures that after round $r-1$, we have $S_{r-1}[r] = r$ and $S_{r-1}[j_2] = S_1[2]$.
- Finally, suppose that $j_r = j_2$, which holds with probability $1/N$. This ensures that after round r, we have $S_r[r] = S_1[2]$ and $S_r[j_2] = r$.
- The final state results in $Z_r = S_r[r + S_1[2]] = S_r[j_2] = r$ with probability 1.

Considering the above events to be independent, the probability that the main path holds is given by $\alpha = \Pr(S_0[1] = r) \cdot \frac{1}{N}\left(1 - \frac{r-2}{N}\right)\left(1 - \frac{2}{N}\right)^{r-3}$. If the above path does not occur, then we assume that the event $(Z_r = r)$ happens due to random association, with probability $1/N$. Thus we can compute the target probability as $\Pr(Z_r = r) \approx \alpha + (1-\alpha)\frac{1}{N}$, and get the result. □

Figure 6 (upper plot) displays our theoretical result in comparison with that of Isobe et al. [6], where the experimental data for RC4 has been obtained from the authors of [1], and the values of $\Pr(S_0[1] = r)$ are obtained from Mantin's distribution [10] for S_0. It is evident that our theoretical values match the experimental data better than that of [6]. Note that the experimental values are for RC4 with 16-byte secret keys, and hence the data is non-smooth (with small spikes) at certain points. In contrast, the theoretical values of our result is for general RC4 with full-length secret keys, thus making the curve smooth.

It is interesting to note that RC4 in WPA exhibits enhanced non-random behavior in the events $(Z_r = r)$, as shown in Fig. 6 (lower plot) for 2^{32} runs of WPA. However, substituting the distribution of S_0 for WPA in our theoretical result (or that of [6]) does not match the experimental observations, and we believe that further investigation in this direction is necessary to settle the issue.

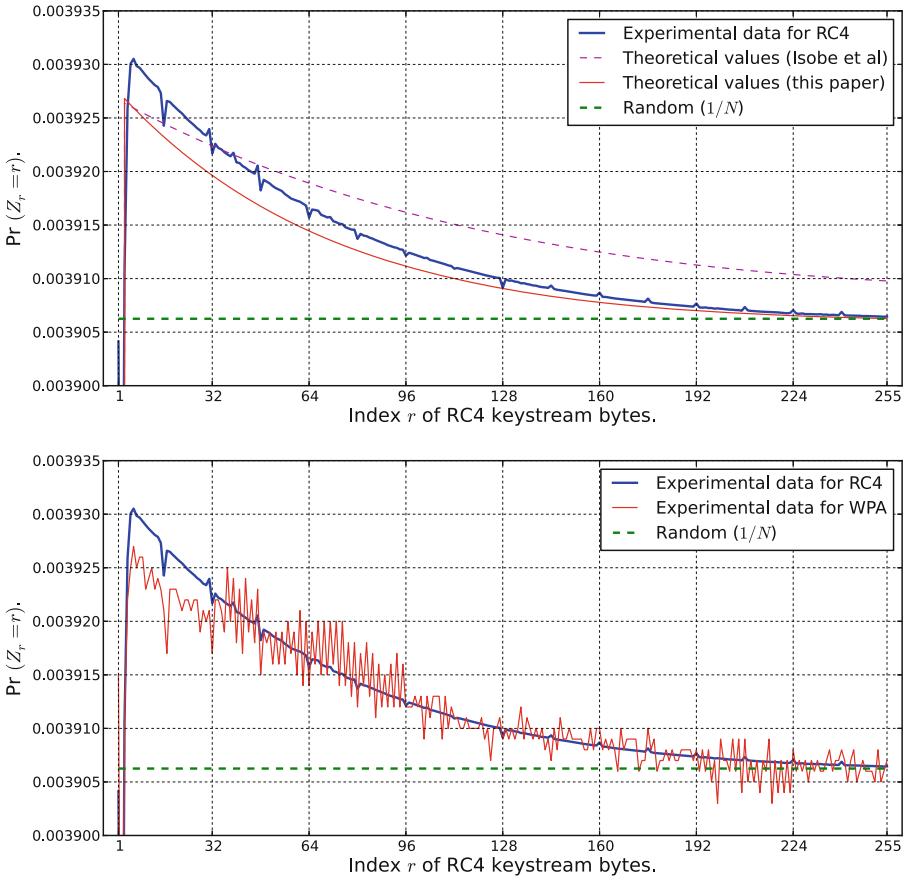


Fig. 6. Plot of $\Pr(Z_r = r)$ for RC4 and WPA, where $r = 3, \ldots, 255$. The experimental data for RC4 in these plots are obtained from the authors of [1].

3 Correlation of the Keystream Bytes with IV in WPA

The weaknesses in the WPA key schedule have recently been exploited twice in the literature of RC4 cryptanalysis – first by Sepehrdad et al. [18,20] and then by AlFardan et al. [1]. While Sepehrdad et al. [18,20] attacked the inner workings of the WPA key schedule to devise a key recovery attack with complexity 2^{96}, the recent work of AlFardan et al. [1] mounted a plaintext recovery attack on WPA by exploiting the biases of the keystream bytes towards absolute values. It was shown that the WPA key schedule, designed to prevent key recovery attacks, unintentionally made the plaintext recovery attack on RC4 even simpler. In this section, we target a third direction of attack – exploiting correlations of the keystream bytes towards the IV to perform plaintext recovery of WPA.

In WPA, the first three bytes of the RC4 key, $K[0], K[1], K[2]$, are derived from the IV. For the u-th recipient in a broadcast setting, let us denote these

bytes by $K_u[i]$ where $i = 0, 1, 2$ and $1 \leq u \leq n$. Note that the values of $K_u[i]$ are publicly known, and hence these could be exploited towards plaintext recovery attacks in case they have prominent correlations with the keystream bytes.

In this section, we will investigate for significant correlations between the keystream output bytes Z_r of WPA and certain linear combinations of the bytes $\{K[0], K[1], K[2]\}$. Let us assume that the number of such correlations with probability significantly different from $1/N = 1/256$ for the keystream byte Z_r is bounded above by Q_r. In this setting, we shall denote the corresponding linear combinations as $L_{r,q}(K[0], K[1], K[2])$, where $q = 1, 2, \ldots, Q_r$.

In Table 2, we list the most significant correlations of this kind for RC4 keystream bytes. Some of these are already known in the literature, and the ones identified by us will be pointed out clearly in the course of this discussion. The references for most of these biases can be found in [18, Figure 4.9]. We know that the bytes $K[0]$ and $K[1]$ are dependent in WPA, and hence the biases observed in RC4 may vary in case of WPA. Thus we first present detailed experimental data in Table 2 to explain the scenario. 'WPA (part)' denotes WPA with the first 3 key bytes constructed from the IV and next 13 key bytes chosen randomly, and 'WPA (full)' denotes WPA with first 3 key bytes constructed from the IV and next 13 key bytes generated by TKIP. We notice that 'WPA (part)' models 'WPA (full)' quite well, also observed earlier by [1].

In Table 2, note that there are certain cases where the biases in RC4 and WPA are not the same. In fact, there are some cases where the biases are in the opposite direction. There are also a few situations where there exist prominent biases in some cases but none in the others. Let us explain a couple of cases.

Table 2. Linear correlations observed between the keystream bytes and key of RC4 and WPA with 2^{32} samples. Probability of random association is $1/256 \approx 0.003906$.

Byte	Linear combinations	[18, Fig. 4.9]	Our Experiments		
			RC4	WPA (part)	WPA (full)
Z_1	$L_{1,1} = -K[0] - K[1]$	0.005304	0.005264	0.005334	0.005338
	$L_{1,2} = K[0]$	0.004367	0.004325	0.004179	0.004179
	$L_{1,3} = K[0] + K[1] + K[2] + 3$	0.005214	0.005220	0.004684	0.004633
	$L_{1,4} = K[0] + K[1] + 1$	0.004072	0.004025	0.003761	0.003760
	$L_{1,5} = K[0] - K[1] - 1$	0.004100	0.004083	0.003905	0.003905
	$L_{1,6} = K[2] + 3$	0.004461	0.004428	0.003904	0.003902
	$L_{1,7} = -K[0] - K[1] + K[2] + 3$	0.004458	0.004424	0.003903	0.003903
Z_2	$L_{2,1} = -1 - K[0] - K[1] - K[2]$	0.005316	0.005298	0.005304	0.005303
	$L_{2,2} = -K[1] - K[2] - 3$	0.005348	0.005303	0.005313	0.005314
	$L_{2,3} = K[1] + K[2] + 3$	0.005341	0.005304	0.005315	0.005315
	$L_{2,4} = K[0] + K[1] + K[2] + 3$	0.002512	0.002507	0.002505	0.002503
Z_3	$L_{3,1} = K[0] + K[1] + K[2] + 3$	0.004436	0.004401	0.004406	0.004405
Z_{256}	$L_{256,1} = -K[0]$	0.004450	0.004427	0.004430	0.004429
	$L_{256,2} = -K[1]$	–	0.003907	0.004037	0.004036
Z_{257}	$L_{257,1} = -K[0] - K[1]$	0.004115	0.004096	0.004095	0.004094

Correlation of $L_{1,4}$: Let us first point out the contrast when the correlation of Z_1 is studied with $K[0]+K[1]+1$. This is positive for RC4, but negative for WPA. In [19], it has been observed experimentally that Z_1 has a positive bias towards $K[0]+K[1]+1$. This bias has been explained in [16] by considering two paths. The first path considers the scenario when Z_1 is always equal to $K[0]+K[1]+1$, which requires the condition $K[1] = N-1$ to be satisfied. However in WPA, the most significant bit of $K[1]$ is zero, and thus $K[1]$ cannot be equal to $N-1$. So this path does not contribute to the event $Z_1 = K[0]+K[1]+1$ in WPA. For the other path, it is assumed in [16] that $\Pr(Z_1 = K[0]+K[1]+1) = 1/N$ when $K[1] \neq N-1$. The experimental results in [16] show that this value is actually $(1/N - 4/N^2)$ for RC4. However in WPA, this value is even lower, close to $(1/N - 9.5/N^2)$. Hence the contrast in the biases between RC4 and WPA.

Correlation of $L_{256,2}$: Consider the case where Z_{256} has no bias towards $(-K[1])$ in RC4, but it is biased in WPA. The reason is that $\Pr(K[1] = K[0]) = 1/4$ in WPA, and thus we can write $P(Z_{256} = -K[1]) \approx 0.25 \times P(Z_{256} = -K[0]) + 0.75 \times 1/N = 0.25 \times 0.004029 + 0.75/256 \approx 0.004036$, which matches with the experiment. This does not occur in RC4 as $\Pr(K[1] = K[0]) = 1/N$ is insignificant in that case due to independent values of $K[0]$ and $K[1]$.

3.1 Improvement in Broadcast Attack for WPA

We present a significantly improved broadcast attack against WPA over the existing works. In [6], only RC4 was studied and thus the idea of using the IV of WPA did not arise. In [1], broadcast attack on WPA has been mounted similar to that on TLS (which is almost equivalent to traditional RC4) and the correlations of the keystream bytes with the IV of WPA have not been explored at all. Exploiting the IV correlations significantly improves the recovery of the plaintext bytes $\{1, 3, 256, 257\}$ in broadcast attack on WPA.

Existing Attacks. In [6], the first byte was obtained using the conditional probability $P(Z_1 = 0|Z_2 = 0)$; thus the order of samples required would be $\Omega(N^2)$. The biases in bytes $\{3, 256, 257\}$ are of the order of $1/N^2$ over the random association probability of $1/N$; thus the order of samples required would be $\Omega(N^3)$ if one carries out the broadcast attack on RC4 as in [6]. The broadcast attack on WPA presented in [1] also considers biases to absolute values and those biases are again of the order of $1/N^2$ over the random association probability $1/N$. In this case as well, $\Omega(N^3)$ samples will be required to mount the attack. For actual broadcast attack, the constant involved in the order notation is quite high (around 2^6) in order to attain a success probability close to 1.

Our Attack. Contrary to the existing approaches, our correlations between the keystream bytes and $K[0], K[1], K[2]$ are of the order of $1/N$ over the random association probability $1/N$, and thus we require only $\Omega(N)$ samples to mount the broadcast attack in theory. It has been pointed out in [1] that the WPA

IV structure actually allows more efficient recovery of plaintext bytes in some positions than with uniform keys in RC4. However, they could only obtain practical results with 2^{24} samples or more to achieve a certain probability of success. In fact, the requirement was around 2^{30} samples for a success probability close to 1. We show that the WPA IV structure actually provides significantly better results (much lower number of samples) for certain plaintext bytes.

In the broadcast scenario, we obtain ciphertext bytes $C_r^{(u)}$ corresponding to various keystream bytes $Z_r^{(u)}$ and fixed message bytes M_r. For each user u, we substitute the $K_u[i]$ values in our list of $L_{r,q}$ to obtain Q_r many votes for $Z_r^{(u)}$. With absolute biases in RC4, the idea of [6] was to use the maximum (or a few top votes) for Z_i to obtain the target plaintext, but these votes could not be accumulated. In [1], the idea of multinomial distribution allowed the biases of Z_i to all possible absolute values to be utilized cumulatively. However, this idea does not work in our case as there is no immediate way to represent the linear combinations of the IV in the form of a probability distribution.

We do not use the votes for all Q_r relations of $Z_r^{(u)}$; we choose the votes corresponding to a few relations out of $L_{r,q}$, and merge those votes for all users. These votes in turn provide us with votes for the target plaintext byte M_r. For Z_1, we get the best result using two relations, while in other cases, we obtain the finest results using only the best biases in $L_{r,q}$. After merging the chosen votes, we consider the byte with the maximum votes as the probable plaintext byte $\hat{M}_r$. Table 3 presents our experimental results for broadcast attack. The success probability in each case is close to 1, and we have attained success in every practical experiment we performed with the claimed packet complexities.

We show the $(Z_2 = 0)$ case to illustrate that while the theoretical complexity of obtaining the byte in broadcast attack is only $\Omega(N)$, it requires 2^{14} samples to reach a success probability close to 1. Our results show significant improvements for recovering the four plaintext bytes $\{1, 3, 256, 257\}$, where the existing works require around 2^{30} samples to achieve the same success probability. It remains an open problem to utilize all biases in $L_{r,q}$ simultaneously in this attack.

Table 3. Experimental results for our plaintext recovery attack.

Byte	Event	Complexity
Z_1	$Z_1 = -K[0] - K[1]$, $Z_1 = K[0] + K[1] + K[2] + 3$	$5 \cdot 2^{13} \approx 2^{15.322}$
Z_2	$Z_2 = 0$	2^{14}
Z_3	$Z_3 = K[0] + K[1] + K[2] + 3$	2^{19}
Z_{256}	$Z_{256} = -K[0]$	2^{19}
Z_{257}	$Z_{257} = -K[0] - K[1]$	2^{21}

3.2 New Key Correlations in WPA

To strengthen the set of biases applicable towards a plaintext recovery attack against WPA, we investigated for correlations of the keystream bytes in WPA to the IV in the general linear form for the events $(Z_r = a{\cdot}K[0]+b{\cdot}K[1]+c{\cdot}K[2]+d)$. In particular, we tried with $a, b, c \in \{-1, 0, 1\}$ and $d \in \{-3, -2, -1, 0, 1, 2, 3\}$. As the first three bytes of the key, $K[0], K[1], K[2]$, are known parameters, these biases may be added to the set of known biases for Z_r, and this may potentially result in a stronger plaintext recovery attack on WPA.

In line of discussion in Sect. 2.1, one may easily note that the distribution of $K[0] \pm K[1] \pm 1$ is not uniform at all. We specifically identify three cases, as presented in Fig. 7, after an experimentation with 2^{35} samples, where we identify many biases of the order of μ/N^2 over random association ($\mu > 0.3$). Towards sharpening the broadcast attack against WPA, these biases need to be explored in more details and it would be an interesting open question how to use these biases in conjunction with the absolute biases as explained in [1].

3.3 Absolute Bias in Z_{259}

In [1,6], several new biases were identified in the first 257 bytes of RC4, and exploited in broadcast attack. In [6,23], the long term biases of RC4 were exploited to mount broadcast attack on later bytes. However, it may be interesting to find absolute biases little farther than byte 257, if they are better than using the long term biases, or if they could be used in conjunction with the long term biases. In this regard, we present a new bias at round $N+3 = 259$, described in Theorem 4. To the best of our knowledge, this is the farthest absolute bias in the initial keystream bytes of RC4 that is of the order of $O(1/N^2)$ over $1/N$.

Theorem 4. *The probability that the* $(N+3)$*-th keystream byte of RC4 is* 3 *is approximately* $\Pr(Z_{N+3} = 3) = 1/N + 0.18/N^2$.

Proof. The main path leading to the target event is as follows.

- Start with $S_0[1] = 3$ and $S_0[2] = 0$ to obtain $S_3[2] = 3$ and $S_3[3] = 0$ after the third round, with probability 1.
- Suppose that none of $j_4, \ldots, j_{N+1}$ touches the locations $\{2, 3\}$ and $j_{N+2} \neq 3$. This happens with probability $\left(1 - \frac{2}{N}\right)^{N-2}\left(1 - \frac{1}{N}\right)$, and eventually leads to $Z_{N+3} = 3$ with probability 1.

Considering the above events to be independent, the probability that the main path holds is given by $\alpha = \Pr\left(S_0[1] = 3 \wedge S_0[2] = 0\right)\left(1 - \frac{2}{N}\right)^{N-2}\left(1 - \frac{1}{N}\right)$. If it does not occur, we assume that $Z_{N+3} = 3$ holds due to random association, with probability $1/N$. Using [10, Theorem 6.2.1] for $\Pr\left(S_0[1] = 3 \wedge S_0[2] = 0\right)$, we compute $\Pr(Z_{N+3} = 3) \approx \alpha + (1 - \alpha) \cdot (1/N) \approx 1/N + 0.18/N^2$. □

Experiments with 2^{33} random keys show that $\Pr(Z_{N+3} = 3) = 0.003909$, both in the case of RC4 and WPA; thus conforming to the theoretical value.

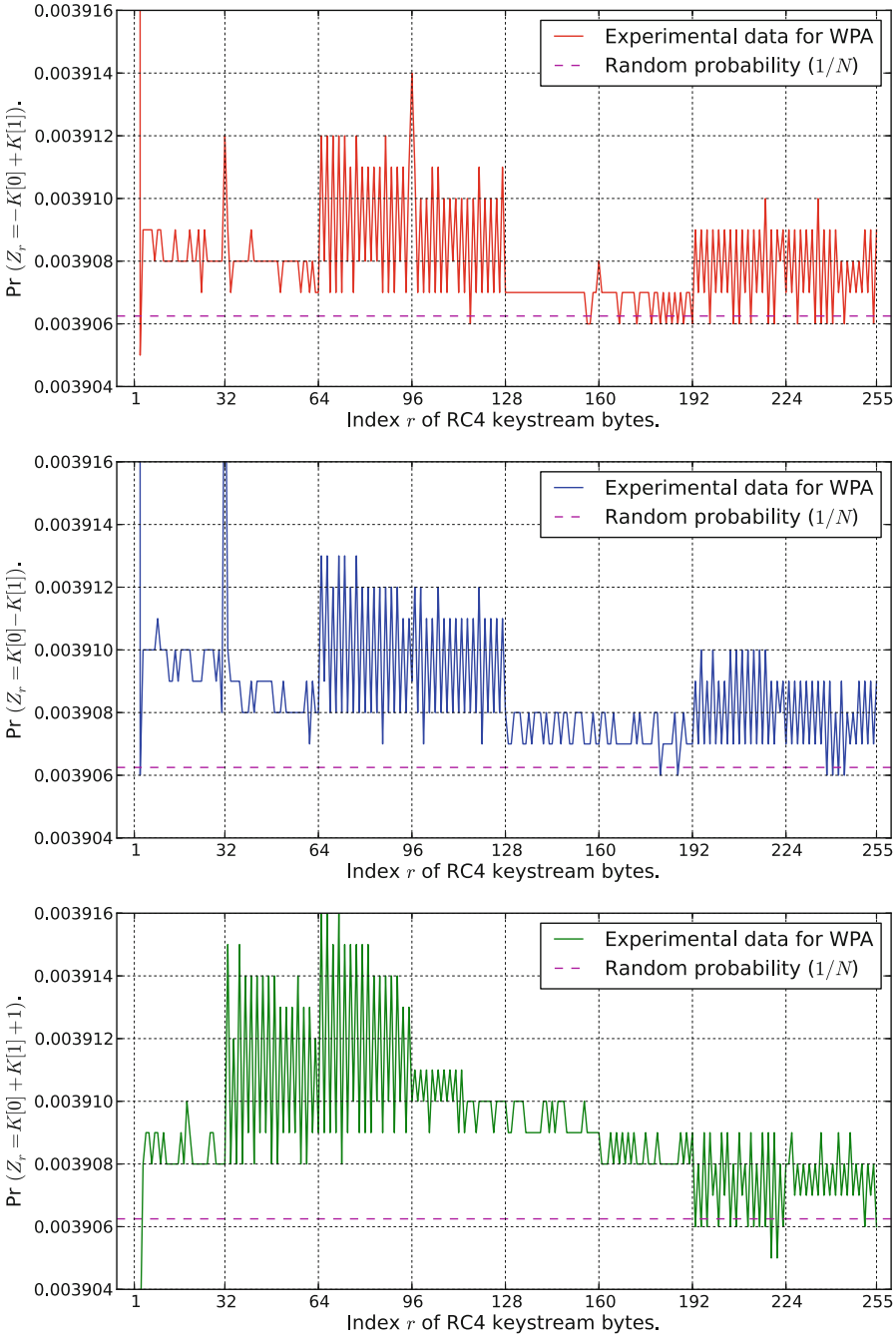


Fig. 7. Linear correlations between the IV and the initial keystream bytes of WPA.

4 Conclusion

In this paper, we present various non-randomness results on RC4 when used in the WPA protocol. We analyze several biases of RC4 and also note how they evolve in WPA as the initial three key bytes are derived from the IV. We prove the interesting sawtooth distribution of the first byte and the similar nature for the biases in $(Z_r = 0)$, as pointed out in [1]. We also improve the theoretical estimate for the $(Z_r = r)$ bias of RC4 to obtain better results than [6].

In another direction, we revisit the correlation of certain keystream bytes to the first three IV bytes in WPA and we notice that they provide much higher biases than what had been presented in [1]. This improves the broadcast attack on WPA significantly towards obtaining certain plaintext bytes. Our combinatorial results complement the existing literature in understanding the reason of some interesting empirical biases in WPA, as well as in adding some new observations and biases in the scenario of broadcast attack against WPA.

Acknowledgments. We are thankful to the anonymous reviewers of FSE 2014 for their detailed review reports containing invaluable feedback, which helped in substantially improving the technical and editorial quality of our paper.

References

1. Alfardan, N., Bernstein, D. J., Paterson, K. G., Poettering, B., Schuldt, J.: On the security of RC4 in TLS. In: USENIX Security Symposium Presented at FSE 2013 as an Invited Talk [2] by Daniel J. Bernstein (2013). http://www.isg.rhul.ac.uk/tls/
2. Bernstein, D. J.: Failures of secret-key cryptography. Invited talk at FSE 2013; session chaired by Bart Preneel (2013)
3. Chaabouni, R.: Break WEP faster with statistical analysis. IACR Cryptology ePrint Arch. **2013**, 425 (2013). http://eprint.iacr.org/2013/425
4. Fluhrer, S.R., Mantin, I., Shamir, A.: Weaknesses in the key scheduling algorithm of RC4. In: Vaudenay, S., Youssef, A.M. (eds.) SAC 2001. LNCS, vol. 2259, p. 1. Springer, Heidelberg (2001)
5. IEEE Computer Society. 802.11iTM - IEEE Standard for Information technology - Telecommunications and information exchange between systems - Local and metropolitan area networks -Specific requirements, Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications, Amendment 6: Medium Access Control (MAC) Security Enhancements, July 2004
6. Isobe, T., Ohigashi, T., Watanabe, Y., Morii, M.: Full plaintext recovery attack on broadcast RC4. In: Moriai, S. (ed.) FSE 2013. LNCS, vol. 8424, pp. 179–202. Springer, Heidelberg (2014)
7. Klein, A.: Attacks on the RC4 stream cipher. Des. Codes Crypt. **48**(3), 269–286 (2008). Published online in 2006
8. Maitra, S., Paul, G.: New form of permutation bias and secret key leakage in keystream bytes of RC4. In: Nyberg, K. (ed.) FSE 2008. LNCS, vol. 5086, pp. 253–269. Springer, Heidelberg (2008)
9. Maitra, S., Paul, G., Sen Gupta, S.: Attack on broadcast RC4 revisited. In: Joux, A. (ed.) FSE 2011. LNCS, vol. 6733, pp. 199–217. Springer, Heidelberg (2011)

10. Mantin, I.: Analysis of the stream cipher RC4. Master's thesis, The Weizmann Institute of Science, Israel (2001). http://www.wisdom.weizmann.ac.il/itsik/RC4/RC4.html
11. Mantin, I., Shamir, A.: A Practical attack on broadcast RC4. In: Matsui, M. (ed.) FSE 2001. LNCS, vol. 2355, pp. 152–164. Springer, Heidelberg (2002)
12. Paterson, K.G., Schuldt, J.C.N., Poettering, B.: Plaintext recovery attacks against WPA/TKIP. In: Cid, C., Rechberger, C. (eds.) FSE 2014, LNCS 8540, pp. 325–349 (2015)
13. Paul, G., Maitra, S.: Permutation after RC4 key scheduling reveals the secret key. In: Adams, C., Miri, A., Wiener, M. (eds.) SAC 2007. LNCS, vol. 4876, pp. 360–377. Springer, Heidelberg (2007)
14. Paul, G., Rathi, S., Maitra, S.: On non-negligible bias of the first output byte of RC4 towards the first three bytes of the secret key. Des. Codes Crypt. **49**(1–3), 123–134 (2008). Initial version in Proceedings of WCC 2007
15. Roos, A.: A class of weak keys in the RC4 stream cipher. Two posts in sci.crypt, 43u1eh$1j3@hermes.is.co.za and 44ebge$llf@hermes.is.co.za (1995). http://www.impic.org/papers/WeakKeys-report.pdf
16. Sarkar, S.: Proving empirical key-correlations in RC4. Inf. Proc. Lett. **114**(5), 234–238 (2014)
17. Sen Gupta, S., Maitra, S., Paul, G., Sarkar, S.: (Non-)random sequences from (non-)random permutations - analysis of RC4 stream cipher. J. Crypt. **27**, 67–108 (2014). doi:10.1007/s00145-012-9138-1. Published online in December 2012
18. Sepehrdad, P.: Statistical and algebraic cryptanalysis of lightweight and ultra-lightweight symmetric primitives. Ph.D. thesis No. 5415, École Polytechnique Fédérale de Lausanne (EPFL) (2012). http://lasecwww.epfl.ch/sepehrdad/Pouyan_Sepehrdad_PhD_Thesis.pdf
19. Sepehrdad, P., Vaudenay, S., Vuagnoux, M.: Discovery and exploitation of new biases in RC4. In: Biryukov, A., Gong, G., Stinson, D.R. (eds.) SAC 2010. LNCS, vol. 6544, pp. 74–91. Springer, Heidelberg (2011)
20. Sepehrdad, P., Vaudenay, S., Vuagnoux, M.: Statistical attack on RC4. In: Paterson, K.G. (ed.) EUROCRYPT 2011. LNCS, vol. 6632, pp. 343–363. Springer, Heidelberg (2011)
21. Tews, E.: Attacks on the WEP protocol. IACR Cryptology ePrint Arch. **2007**, 471 (2007). http://eprint.iacr.org/2007/471
22. Tews, E., Weinmann, R.-P., Pyshkin, A.: Breaking 104 bit WEP in less than 60 seconds. In: Kim, S., Yung, M., Lee, H.-W. (eds.) WISA 2007. LNCS, vol. 4867, pp. 188–202. Springer, Heidelberg (2008)
23. Ohigashi, T., Isobe, T., Watanabe, Y., Morii, M.: How to recover any byte of plaintext on RC4. In: Lange, T., Lauter, K., Lisoněk, P. (eds.) SAC 2013. LNCS, vol. 8282, pp. 155–173. Springer, Heidelberg (2014)
24. Vaudenay, S., Vuagnoux, M.: Passive–only key recovery attacks on RC4. In: Adams, C., Miri, A., Wiener, M. (eds.) SAC 2007. LNCS, vol. 4876, pp. 344–359. Springer, Heidelberg (2007)

Cryptanalysis II

Probabilistic Slide Cryptanalysis and Its Applications to LED-64 and Zorro

Hadi Soleimany(✉)

Department of Information and Computer Science,
Aalto University School of Science, Espoo, Finland
hadi.soleimany@aalto.fi

Abstract. This paper aims to enhance the application of slide attack which is one of the most well-known cryptanalysis methods using self-similarity of a block cipher. The typical countermeasure against slide cryptanalysis is to use round-dependent constants. We present a new probabilistic technique and show how to overcome round-dependent constants in a slide attack against a block cipher based on the general Even-Mansour scheme with a single key. Our technique can potentially break more rounds than any previously known cryptanalysis for a specific class of block ciphers. We show employing round constants is not always sufficient to provide security against slide variant cryptanalysis, but also the relation between the round constants should be taken into account. To demonstrate the impact of our model we provide analysis of two round-reduced block ciphers LED-64 and Zorro, presented in CHES 2011 and CHES 2013, respectively. As a first application we recover the key for 16 rounds of Zorro. This result improves the best cryptanalysis presented by the designers which could be applied upto 12 rounds of its 24 rounds. In the case of LED-64 the cryptanalysis leads to the best results on 2-step reduced LED-64 in the known-plaintext model.

Keywords: Block cipher · Slide attack · Zorro · LED-64 · Even-mansour

1 Introduction

Block ciphers can be attacked using a large variety of attacks employing different properties of the ciphers. Statistical cryptanalysis like differential [4] and linear [25] attacks make use of non-randomness characteristics of the cipher and the complexity of the attack is increased by adding more rounds to the cipher. In contrast, self-similarity cryptanalysis techniques are applicable on a small class of block ciphers and the complexity of the attacks is usually independent of the number of rounds. Self-similarity attacks exploit the weakness of the key schedule rather than non-random statistical properties of the cipher. *Slide cryptanalysis* is a well-known example of such techniques and it utilizes the symmetry properties of the cipher [7]. If an iterative block cipher with identical round functions has a periodic key schedule, it can be presented as a cascade of repeated copies of a single function F_k with an identical key k, where F_k consists of one or more

C. Cid and C. Rechberger (Eds.): FSE 2014, LNCS 8540, pp. 373–389, 2015.
DOI: 10.1007/978-3-662-46706-0_19

rounds of the cipher. Slide attacks are based on the observation that if two plaintexts P and P' satisfy the relation $P' = F_k(P)$ then $C' = F_k(C)$ holds for free. Such a pair $((P, C), (P', C'))$ is called the *slid pair*. Given a slid pair the security of the cipher is reduced to finding the key for the function F_k in the known plaintext model. In general, $2^{n/2}$ known plaintexts are required to find at least one slid pair for an n-bit block cipher. The total time complexity of the cryptanalysis consists of the complexities of three steps: preparing the required data, detecting a slid pair, and finally obtaining the key from the slide pair.

Slide attacks and other self-similarity cryptanalysis can be prevented by a careful design of a key schedule. But a strong key schedule cannot be achieved for free. It has impact on latency, power consumption and size of the implementation. Therefore the designers of lightweight ciphers such as PRINTcipher [23], LED [20], PRINCE [11] and Zorro [17], adopted another direction to establish security against self-similarity cryptanalysis. They introduced round-dependent constants added to the data input at each round to make the rounds different. Then a single master key is simply added after every fixed number of rounds called as *step*. Even if in such a construction each step has the same structure and the key used at every step is the same, the computation at each step is varied by the round constants. In this manner, the slide cryptanalysis can be prevented.

Such a cipher construction can be seen as an instance of the generalized Even-Mansour scheme [15] with a single key. It is defined as $C = E_K(P) = F_s(\cdots F_2(F_1(P \oplus K) \oplus K) \oplus K \cdots \oplus K)$ where F_i are constructed as cascades of the same fixed permutations but with different round constants. In this work, we investigate the security of this cipher structure and develop a new statistical variant of slide cryptanalysis. The idea is to slide one instance of encryption against another instance of encryption by one step and investigate, if for a pair $((P, C), (P', C'))$ it would be possible to predict, with significant probability, the difference $C \oplus F_s(C' \oplus K)$ given the difference $P' \oplus F_1(P \oplus K)$. By taking a probabilistic approach we can circumvent the devastating effect of different round constants in the deterministic slide cryptanalysis.

Potentially, the described attack has two main advantages compared to the classical differential cryptanalysis. The first one is that the attacker has more freedom to control the active S-boxes. If the difference in the round constants and data are identical, they cancel each other, which leads to a smaller number of active S-boxes in the characteristic. Example of such a situation is given in Sect. 4.2 for LED-64. While it is proved that the normal differential characteristic over four rounds has at least 25 S-boxes, we can find differential characteristics of LED-64 over four iterative rounds which in our slide setting have just 13 active S-boxes. We also note that even if we exploit a kind of related-key differential characteristics, our attack is in the single-key model, since we exploit the relation between the round constants instead of keys to control the differential pattern. The second merit is the existence of an efficient key-recovery method for our model. As each step of our target ciphers consists of several rounds, it makes it hard to convert a distinguisher to the key-recovery attack over more steps by guessing just a part of the key. But we present an efficient method to obtain the

key of s steps of the cipher based on the differential property in slide mode for $s-1$ steps.

To demonstrate the effect of our statistical slide framework, we apply key-recovery attack on block ciphers LED-64 and Zorro. We present a key-recovery cryptanalysis on a reduced 16-round version, while the best previous key-recovery cryptanalysis is on 12 rounds. Also we present a novel cryptanalysis on 2-reduced steps of LED-64 which leads to the best attack on this version of the cipher in the known-plaintext model. Our results and the previous results are summarized in Table 1.

Table 1. Summary of single-key attacks on round-reduced LED-64 and Zorro

Cipher	Attack Type	Steps	Data	Time	Memory	Source
Zorro	Impossible differential	2.5	2^{115}CP	2^{115}	2^{115}	[17]
	Meet-in-the-middle	3	2^{2}KP	2^{104}	-	[17]
	Probabilistic slide	4	$2^{123.62}$KP	$2^{123.8}$	$2^{123.62}$	Sect. 4.1
	Probabilistic slide	4	$2^{121.59}$KP	$2^{124.23}$	$2^{121.59}$	Sect. 4.1
	Internal differential[a]	6	$2^{54.25}$CP	$2^{54.25}$	$2^{54.25}$	[19]
LED-64	Meet-in-the-middle	2	2^{8}CP	2^{56}	2^{11}	[21]
	Meet-in-the-middle	2	2^{16}CP	2^{48}	2^{17}	[13]
	Meet-in-the-middle	2	2^{48}KP	2^{48}	2^{48}	[13]
	Generic	2	2^{45}KP	$2^{60.1}$	2^{60}	[14]
	Probabilistic slide	2	$2^{45.5}$KP	$2^{46.5}$	$2^{46.5}$	Sect. 4.2
	Probabilistic slide	2	$2^{41.5}$KP	$2^{51.5}$	$2^{42.5}$	Sect. 4.2
	Generic	3	2^{49}KP	$2^{60.2}$	2^{60}	[14]

[a]This attack is applicable just on 2^{64} keys (out of 2^{128}), CP – Chosen Plaintexts, KP – Known Plaintext.

Some previous cryptanalysis of LED-64 do not exploit any specific property of the cipher. In [27] it is shown that the behavior of the function $x \oplus F(x)$, for a random permutation F, is not ideally random and they exploit this fact in a generic attack on the EM-construction with two alternative keys. The result is improved in [14] via a generic attack on a 3-step EM-construction with a single key. This attack is independent of the permutations but has high complexity. An accelerated exhaustive search for 2-step reduced version of LED-64 is presented in [21] in the chosen-plaintext model. Our attack requires only known plaintext and has much lower time complexity. Recently, parallel to our work, an advanced meet-in-the-middle cryptanalysis is presented for a 2-step version of LED-64 in IACR Eprint Archive [13], but is slightly slower than our cryptanalysis in known-plaintext model.

Zorro was presented recently at CHES 2013. It is a tweaked version of the standard block cipher AES and targets on efficient masking to establish side-channel security with better performance. The best key-recovery cryptanalysis

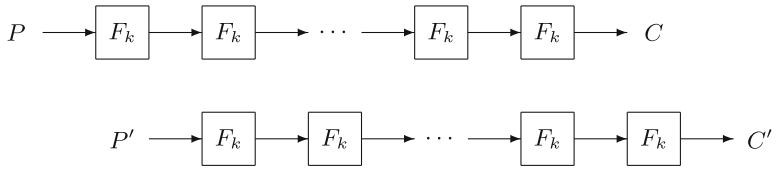


Fig. 1. Slide cryptanalysis

so far was presented by the designers in the classical single-key model. It is a meet-in-the-middle attack on 12 rounds. We carry out the first third-party cryptanalysis of Zorro and mount a key-recovery cryptanalysis on 16 rounds of Zorro. Let us also mention that simultaneously a cryptanalysis of the full Zorro is presented in [19]. It works for a fraction 2^{64} out of 2^{128} keys in the weak-key model and requires chosen plaintexts, whereas our result work for all keys in known-plaintext model.

This paper is organized as follows. In Sect. 2, we start by recalling the typical slide cryptanalysis and previous works, and then continue to introduce the basic concepts of our method. In Sect. 3, we describe the target ciphers Zorro and LED briefly. Section 4 gives an overview of our strategy to construct a distinguisher followed by applications of the probabilistic slide cryptanalysis on step-reduced Zorro and LED-64. We conclude in Sect. 5.

2 Slide Cryptanalysis

2.1 Basic Idea

A typical block cipher can be described as a cipher with iteration of a key-dependent function called as round. Each round is a keyed permutation $R_{k_i}(X)$, where k_i is the ith round key derived from the master key using a key schedule. More precisely, the encryption procedure of the n-bit plaintext P via the iterated application of r rounds is described as $X_i = R_{k_i}(X_{i-1})$, for $i = 1, \cdots, r$ where X_0 and X_r represents plaintext P and corresponding ciphertext C respectively. Let us assume that the cipher can be represented as an iteration of a single permutation F_k. Depending on the key-schedule this permutation may consist of one or more rounds of the cipher. If plaintexts P and P' satisfy the relation $P' = F_k(P)$, then $C' = F_k(C)$ holds for free independently of the number of rounds as illustrated in Fig. 1. A pair $((P, C), (P', C'))$ with this property is called a *slid pair*. For an n-bit block cipher $P' = F(P)$ occurs with probability 2^{-n}. Given $2^{n/2}$ plaintext-ciphertexts (P, C) there exists 2^n pairs which emphasize to expect one slide pair. To convert the distinguisher to a key-recovery cryptanalysis the only requirement for F_k is to be weak enough against known-plaintext cryptanalysis.

2.2 Previous Works

Since the basic slide cryptanalysis is a too restrictive approach, several developments have been proposed to enhance the basic idea. Two advanced techniques

termed as *sliding with a twist* and *complementation slide* are presented by Biryukov and Wagner [8]. If two keys are used alternatively in a Feistel cipher, then we can slide two instances of encryption against each other by one round to cancel the difference between the keys by the complementation slide property. In slide-with-a-twist cryptanalysis an instance of decryption is slid against an instance of encryption. This technique is applicable on another class of ciphers.

This idea is extended to introduce a weak key class of involution block ciphers in [5]. In [28] Feistel ciphers with independent pre- and post-whitening keys are studied and shown that there is a cryptanalysis with time and data complexity $n2^{n/2+1}$. Furuya uses the observation presented in [8] that if (P, P') is a slid pair, then $(F_K(P), F_K(P'))$ is also a slid pair. This technique provides more known plaintexts to mount efficient slide cryptanalysis on more complicated functions F_K [16]. Biham et al. pursue another direction which allows to find a slid pair much faster with the cost of almost the whole codebook [3]. As another direction slide cryptanalysis can be leveraged to a distinguisher on hash functions as it is done for the inner component of SHA-1 [30]. It does not seem useful for collision or (second) preimage cryptanalysis but works for distinguishing and also key recovery in MAC mode [18].

To the best of our knowledge, this paper is the first application of probabilistic slide cryptanalysis in the single-key model. In [8] it is suggested to use a differential property of the identical function to find the key from a slid pair which is a different approach and have not been applied in practice. Also in [29] the method of *realigning slide cryptanalysis* is presented to pass the middle round in a nondeterministic way in related-key scenario.

2.3 Probabilistic Slide Cryptanalysis

In this section we present our new technique which combines a usual slide attack with differential type characteristics. We focus our attention to an n-bit block cipher with general Even-Mansour construction that consists of s different permutations and one key. Analogically to the basic slide distinguisher we consider an encryption instance and slide it against another instance of the same encryption by one step. Due to the differences between round functions the basic slide cryptanalysis is not applicable. Assume there exists a sequence of differences $\mathcal{D} = \{\Delta_r : 0 \le r \le s-1\}$ such that $\Pr[F_r(x) \oplus F_{r-1}(x \oplus \Delta_{r-2}) = \Delta_{r-1}] = 2^{-p_{r-1}}$ where $0 \le p_r$ and $2 \le r \le s$. Thanks to the equality of keys we obtain a differential type characteristic by concatenating the differences in $\mathcal{D}$. This characteristic has probability $2^{-p} = \prod_{r=1}^{s-1} 2^{-p_r}$. So $F_{s-1} \circ \cdots \circ F_1(x) \oplus F_s \circ \cdots \circ F_2(x \oplus \Delta_{in}) = \Delta_{out}$ holds with probability 2^{-p} where $\Delta_{in} = \Delta_0$ and $\Delta_{out} = \Delta_{s-1}$ as illustrated in Fig. 2. In other words, if a pair (P, P') satisfies $F_1(P \oplus K) \oplus P' = \Delta_{in}$ then $C \oplus F_s^{-1}(C' \oplus K) = \Delta_{out}$ occurs with the probability 2^{-p}.

Analogously to the usual slide attack, such a pair for which the characteristic holds is called the *right slid pair*. Given $2^m = 2^{n/2+p/2}$ known plaintext there exist 2^{n+p} pairs of which 2^p are expected to satisfy the relation $F_1(P \oplus K) \oplus P' = \Delta_{in}$ for the unknown key. The right slid pairs are among them. Since the input

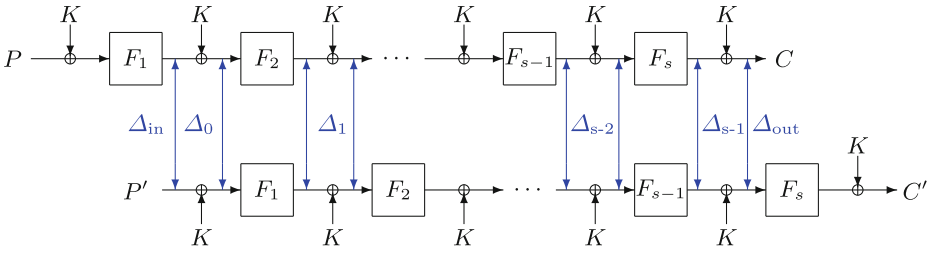


Fig. 2. Slide cryptanalysis on general Even-Mansour scheme with one key

difference Δ_{in} yields the output Δ_{out} with probability 2^{-p}, we expect to get one right slid pair for the characteristic.

Key-Recovery. Next we describe the key-recovery algorithm. For a correct slid pair $((P, C), (P', C'))$ we have

$$C' \oplus F_s(C \oplus \Delta_{out}) = K = P \oplus F_1^{-1}(\Delta_{in} \oplus P',)$$

where K is the correct key. So the correct slid pair satisfies the relation

$$C' \oplus F_1^{-1}(\Delta_{in} \oplus P') = P \oplus F_s(C \oplus \Delta_{out}).$$

We utilize this property to find a slid pair efficiently by storing these values in hash tables for all plaintext-ciphertext pairs. The attack procedure is as follows:

1. Ask for the encryption of $2^m = 2^{n/2+p/2}$ arbitrary plaintexts.
2. For all plaintext-ciphertext pairs (P, C) compute the value of $C \oplus F_1^{-1}(P \oplus \Delta_{in})$ and store the computed value with the corresponding C in the hash table T_1. Sort them according to the value $C \oplus F_1^{-1}(P \oplus \Delta_{in})$.
3. For all plaintext-ciphertext pairs (P, C) compute the value of $P \oplus F_s(\Delta_{out} \oplus C)$ and store the computed value with C in the hash table T_2. Sort them according to the value $P \oplus F_s(\Delta_{out} \oplus C)$. Keep some (P, C) pairs to test the key candidates.
4. For each collision in the hash tables T_1 and T_2 find corresponding ciphertexts C and C' then compute a key candidate $K = C' \oplus F_s(C \oplus \Delta_{out})$. Use a (P, C) to test the key.

Step 1 requires 2^m full encryptions. To prepare each hash tables we compute one step of the cipher for all known plaintexts. So Step 2 and Step 3 requires totally $2 \cdot 2^m/r$ full encryptions. We expect to have $2^{2m-n} = 2^p$ key candidates to try in Step 4 which requires 2^p encryptions. The total time complexity is $2^m + 2^{m+1}/r + 2^p$. To perform the attack, one needs two hash tables T_1 and T_2 to store 2^m ordered pairs of n-bit values in both T_1 and T_2.

More Output Differences. In this part we study the improvements of the described distinguisher. A natural improvement is to consider a differential instead of a single differential characteristic, and try to improve the estimate of the differential probability.

Another approach is to consider L different output differences Δ_{out}^i, $i \in \{1, \cdots, L\}$, to decrease the data requirement by increasing the total probability. This comes with the cost of repeating the attack algorithm L times and a small increase in the memory requirement. Without loss of generality we can assume that each output difference Δ_{out}^i occurs with equal probability 2^{-p} for a fixed input difference Δ_{in}, that is, if $F_1(P \oplus K) \oplus P' = \Delta_{in}$, then $C \oplus F_r^{-1}(C' \oplus K) = \Delta_{out}^i$ holds with probability 2^{-p}, for all $i \in \{1, \cdots, L\}$. If the probabilities are not equal, order the output differences in the decreasing order according to their probabilities.

The attack procedure is described as follows:

1. Get encryptions C of 2^m arbitrary plaintexts P.
2. For all pairs (P, C) compute the value of $C \oplus F_1^{-1}(P \oplus \Delta_{in})$ and store the computed value with P and C in a hash table T_1. Sort them according to the value $C \oplus F_1^{-1}(P \oplus \Delta_{in})$.
3. for $i \in \{0, \cdots, 2^\ell\}$
   - 3.1 Allocate $2 \cdot 2^m$ memory for the hash table T_2.
   - 3.2 For all plaintext-ciphertext pairs (P, C) compute the value of $P \oplus F_s(\Delta_{out}^i \oplus C)$ and store the computed value and the corresponding C in the hash table T_2. Sort them based on the value $P \oplus F_s(\Delta_{out}^i \oplus C)$.
   - 3.3 For each collision in the hash tables T_1 and T_2 find corresponding ciphertexts C and C' then compute the key candidate as $K = C' \oplus F_s(C \oplus \Delta_{out}^i)$. Given some P and C, test the key.
   - 3.4 If all key candidates are wrong, free the allocated memory of T_2.

We denote by ℓ the logarithm of L. Since $\Pr[C \oplus F_r^{-1}(C' \oplus K) = \Delta_{out}^i, \text{for some } i \in \{1, \cdots, 2^\ell\} | F_1(P \oplus K) \oplus P' = \Delta_{in}] = 2^{\ell - p}$ the attack requires $2^m = 2^{n/2 + (p-\ell)/2}$ known plaintexts. Time complexity is $2^m + 2^m/r + 2^\ell(2^m/r + 2^{2m-n})$ encryptions which is dominated by $2^\ell(2^m/r + 2^{2m-n})$. Given L output differences, any number of them can be used in the attack allowing trade-off between data and time complexity. The memory requirement is the same for all $L > 1$. To perform the attack, one needs two hash tables T_1 and T_2 where T_1 is used to store 2^m triplets of n-bit values and T_2 to store 2^m ordered pairs n-bit values.

3 Target Ciphers

In this section we present a brief description of the block ciphers to be analyzed. We describe first the block cipher Zorro and continue with the description of the block cipher LED, and finally introduce the notation to be used in this paper.

3.1 Description of Zorro

Zorro is a 128-bit block cipher and supports 128-bit key. The state can be illustrated as a 4×4 matrix where each cell represents a byte. Zorro is a generalized Even-Mansour cipher consisting 6 steps. There exist no key schedule and after each step the same key is xored to the state. Each step is composed of 4 rounds. One round consists of four transformations. One is the adding of the round constant and the other three are borrowed from the AES and applied in the following order.

1. `SubCells:` A byte-wise transformation that applies an 8-bit S-box to each byte of the first row.
2. `AddConstants:` XOR operation between the first row of the state and the current round constant. Let r be the number of the current round represented as a byte, for $1 \leq r \leq 24$. Then the round constant is defined as $r \parallel r \parallel r \parallel r \ll 3$.
3. `ShiftRows:` A linear transformation that cyclically shifts the i'th row i bytes to the left.
4. `MixColumns:` A linear transformation represented by a 4×4 matrix over $GF(2^8)$.

The last two operations are exactly like in the AES. For the definition of S-box and more details we refer to [17].

3.2 Description of LED

LED is a 64-bit block cipher. Two main variants of the cipher are LED-64 and LED-128, which support the key sizes 64 and 128, respectively. The 64-bit state is represented by a 4×4 matrix, where each cell represents a nibble in $GF(2^4)$. The construction of LED-64 is a generalized Even-Mansour with one key and 8 steps. Each step includes four rounds. Each round consists of four transformations of which three are inspired by the AES.

1. `AddConstants` adds a round-dependent constant to the state. To construct the round constants one proceeds as follows. A string of six bits $(rc_5, rc_4, rc_3, rc_2, rc_1, rc_0)$ is initialized to zero. Then at each round, the bits are shifted to the left by one position, and the new value of rc_0 is computed as $rc_5 \oplus rc_4 \oplus 1$. Let us denote by $(ks_7, ks_6, ...ks_0)$ the bits of the byte that represents the key size. Then the corresponding round constant is defined as follows.

$ks_7 \| \| ks_6 \| \| ks_5 \| \| ks_4$	$0 \| \| rc_5 \| \| rc_4 \| \| rc_3$	0	0
$ks_7 \| \| ks_6 \| \| ks_5 \| \| ks_4 \oplus 1$	$0 \| \| rc_2 \| \| rc_1 \| \| rc_0$	0	0
$ks_3 \| \| ks_2 \| \| ks_1 \oplus 1 \| \| ks_0$	$0 \| \| rc_5 \| \| rc_4 \| \| rc_3$	0	0
$ks_3 \| \| ks_2 \| \| ks_1 \oplus 1 \| \| ks_0 \oplus 1$	$0 \| \| rc_2 \| \| rc_1 \| \| rc_0$	0	0

Table 2. S-box of LED

x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
S-box(x)	C	5	6	B	9	0	A	D	3	E	F	8	4	7	1	2

2. `SubCells` applies the same 4-bit to 4-bit S-box given in Table 2 in parallel on each of the 16 nibbles of the state.
3. `ShiftRows` cyclically rotates the i'th row by i nibble(s) to the left.
4. `MixColumns` multiplies each column by an MDS matrix $M = \begin{bmatrix} 4 & 1 & 2 & 2 \\ 8 & 6 & 5 & 6 \\ B & E & A & 9 \\ 2 & 2 & F & B \end{bmatrix}$, over the field $GF(2^4)$ under the polynomial $x^4 + x + 1$.

3.3 Notations

Thanks to the similarity of Zorro and LED we can use almost the same notations to describe the states or operations for both ciphers. We represent the state as a 4×4 matrix where each cell is a byte or a nibble in Zorro and LED, respectively. We denote by X_r^I the input of the r'th round while X_r^S ,X_r^R, X_r^M and X_r^A denote the intermediate states after the application of SubCells, ShiftRows, MixColumns and AddConstants operations in the r'th round, respectively. We also denote the cell in the ith row and jth column of the state X by $X(i + 4j)$, where $0 \leq i, j \leq 4$. This notation is illustrated in Fig. 3.

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

Fig. 3. State representation of Zorro and LED

Each step of Zorro and LED has four rounds. In our cryptanalysis, we slide two instances of encryption against each other by one step and compare their states. Let us denote by RC_r the round constant in r'th round and by DRC_r the difference between round constants in the rounds r and $r + 4$. Then $DRC_r = RC_r \oplus RC_{r+4}$. Let us note that in Zorro this difference has only four non-zero bytes $DRC_r(0, 1, 2, 3)$ on the first row. Similarly, the round constant difference DRC_r of LED can have non-zero nibbles only on the second column $DRC_r(1, 5, 9, 13)$. Throughout the paper SC, SR, MC and AC stands for SubCells, ShiftRows, MixColumns and AddConstants operations.

4 Applications of the Probabilistic Slide Cryptanalysis

Our aim is to find a differential type characteristic with high probability between two slid instances of the cipher as depicted in Fig. 2. Finding differential characteristics have the highest probability among all possible choices is a challenging task as a general problem since even an automatic search for the whole space is not feasible. There are some techniques to make this simpler and speed up the search effectively. Matsui presents an algorithm to find the best differential characteristic and linear approximation in [26]. The algorithm uses a branch-and-bound method recursively to find the r-round characteristic of DES with highest probability based on the best $r-1$-round characteristics. The algorithm is not feasible for all ciphers but the main principles of the algorithm have been adopted widely in several works (for example look at [1,6,9,10]). The probability of differential characteristic is proportional to the number of involved active S-boxes. One direction in the word-oriented block ciphers and hash functions is to find a general pattern of active and inactive differences holds in the cipher properties such that the number of active S-boxes is as small as possible. Next the differential characteristic with specific differential values for the existence pattern can be found by guess-and-determine methods. In this section we explore this approach for two block ciphers Zorro and LED-64 to construct a distinguisher described in Sect. 2.3 for each cipher separately and proceed by applying key-recovery cryptanalysis. We use the notations introduced in Sect. 3.3 to denote the various intermediate states X of the encryption process of P to C. We use similar notation for the pair (P', C') but now with X'.

4.1 Slide Cryptanalysis on Zorro

We start by an observation about the degrees of freedom of constructing a 2-round differential characteristic of Zorro block cipher. Let us consider a non-zero difference $X'^{I}_{r}(i) \oplus X^{I}_{r+4}(i)$ where $0 \leq i \leq 3$. After the S-box operation it may be transferred to a difference such that at the end of the round $X'^{M}_{r}(i) \oplus X^{M}_{r+4}(i) = 0$ which indicates that we can bypass the next round in the same byte position for free. So potentially for a given differential characteristic on $r-1$ rounds we can extend it over two more rounds with the cost of at most four active S-boxes. We use the following procedure to specify difference $X'^{S}_{r}(i) \oplus X^{S}_{r+4}(i)$ such that $X'^{M}_{r}(i)$ and $X^{M}_{r+4}(i)$ have identical values. Since `SC` and `AC` operations do not change the differences in the last three rows, the differences in the bytes $X'^{R}_{r}(i+4, i+8, i+12) \oplus X^{R}_{r+4}(i+4, i+8, i+12)$ are determined. By assuming $X'^{M}_{r}(i) \oplus X^{M}_{r+4}(i) = 0$, four bytes of the input and output of the Mixcolumn matrix are known and we are able to obtain the difference value $X'^{R}_{r}(i) \oplus X^{R}_{r+4}(i)$. After that $X'^{S}_{r}(i) \oplus X^{S}_{r+4}$ can be determined by coring $X'^{R}_{r}(i) \oplus X^{R}_{r+4}(i)$ with $DRC_r(i)$. This procedure is illustrated in Fig. 4, where we denote the bytes with known differences by '$*$' and aim to find unknown differences denoted by '?'.

Since about one half of all S-box differentials exist, to construct a 2-round differential characteristic as described, a fraction of 2^{-4} choices can match the conditions of four S-boxes. We can start with an initial state such that the bytes

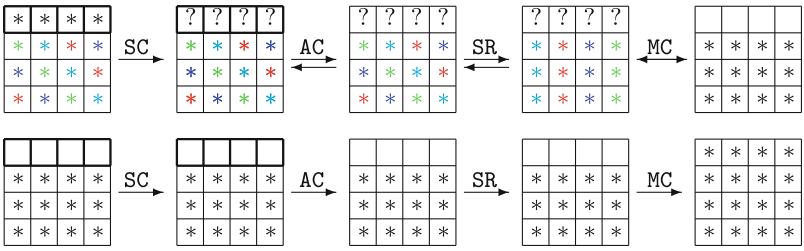


Fig. 4. Two rounds differential characteristic pattern with four active S-boxes

in the first row have no difference. Since there exist 2^{96} different states we expect 2^{96-2r} states to satisfy the pattern of r-rounds. This shows that even for the full round cipher there exist numerous candidates. A naive question arises how one can exploit these degrees of freedom to create a differential characteristic which has still less active S-boxes. Our choice is to select the difference value $\Delta_{in} = X_1'^I \oplus X_5^I$ such that the first three rounds have no active S-boxes. The first row after MC has no difference with probability around $2^{-8\times4} = 2^{-32}$. So if we start with a state by no difference in the first row a fraction $2^{-32\times2}$ out of 2^{96} states can bypass two more rounds with probability one. While trying all 2^{96} states is not feasible as is described in [17] we can find these 3-round characteristics efficiently by utilizing an alternative method. In the remainder of this part we take a detailed look at this technique summarized in Algorithm 1.

We aim to find a 3-round characteristic that holds with probability one. If we guess the difference of bytes $X_2'^I(i+4, i+8) \oplus X_6^I(i+4, i+8)$ located in the i'th column where $0 \leq i \leq 3$, the difference of the third active byte $X_2'^I(i+12) \oplus X_6^I(i+12)$ in the same column can be obtained considering the Mixcolumn matrices of the first round. Hence we can find two first columns of $X_2'^I \oplus X_6^I$ by guessing only four bytes $X_2'^I(4,5,8,9) \oplus X_6^I(4,5,8,9)$. Consequently $X_2'^{IR}(7,10,11,14) \oplus X_6^R(7,10,11,14)$ would be known after SR. By assumption $X_2'^M(2,3) \oplus X_6^M(2,3) = 0$ and using MixColumn matrices of the second round, difference $X_2'^{IR}(7,14) \oplus X_6^R(7,14)$ can be found. We save the values $X_2^I(4,8,9,13) \oplus X_6'^I(4,8,9,13)$ in the row indexed by $X_2'^I(5,7,12,14) \oplus X_6^I(5,7,12,14)$ in the hash table T_1. Similarly in a separate computation by guessing four bytes $X_2'^I(6,7,10,11) \oplus X_6^I(6,7,10,11)$ the differences of four more bytes $X_2'^I(5,12,14,15) \oplus X_6^I(5,12,14,15)$ are obtained. We save the values $X_2'^I(6,10,11,15) \oplus X_6^I(6,10,11,15)$ in the row indexed by $X_2'^I(5,7,12,14) \oplus X_2^I(5,7,12,14)$ in the hash table T_2. Matching two hash tables leads to finding all 3-round differential characteristics which have probability one.

Key Recovery. For all 2^{32} states obtained from Algorithm 1 we extend the characteristic by two rounds iteratively. The best differential characteristic we found for 12 rounds has probability $\Pr[X_{13}'^I \oplus X_{17}^I = \Delta_{out} | X_1'^I \oplus X_5^I = \Delta_{in}] = 2^{-119.24}$ where the values Δ_{in} and Δ_{out} with the details of characteristic are given in Appendix A.2. It leads to the the key-recovery cryptanalysis described

Algorithm 1. Finding 3-round characteristics of Zorro with probability one

for all $X_2'^I(4,8) \oplus X_6^I(4,8)$ **do**
  Find $X_2'^I(12) \oplus X_6^I(12)$ using the MixColumn matrix of the first round.
  for all $X_2'^I(5,9) \oplus X_6^I(5,9)$ **do**
    Find $X_2'^I(13) \oplus X_6^I(13)$ using the MixColumn matrix of the first round.
    Find $X_2'^I(7,14) \oplus X_6^I(7,14)$ using the MixColumn matrix of the second round.
    Save the value $X_2'^I(4,8,9,13) \oplus X_6^I(4,8,9,13)$ in the row is indexed by $X_2'^I(5,7,12,14) \oplus X_6^I(5,7,12,14)$ in T_1.
  end for
end for
for all $X_2'^I(6,10) \oplus X_6^I(6,10)$ **do**
  Find $X_2'^I(14) \oplus X_6^I(14)$using the MixColumn matrix of the first round.
  for all $X_2'^I(7,11) \oplus X_6^I(7,11)$ **do**
    Find $X_2'^I(15) \oplus X_6^I(15)$using the MixColumn matrix of the first round.
    Find $X_2'^I(5,12) \oplus X_6^I(5,12)$using the MixColumn matrix of the second round.
    Save the value $X_2'^I(6,10,11,15) \oplus X_6^I(6,10,11,15)$ in the row is indexed by $X_2'^I(5,7,12,14) \oplus X_6^I(5,7,12,14)$ in T_2.
  end for
end for

in Sect. 2.3 on 16-reduced round of Zorro. The attack requires $2^{64+59.62} = 2^{123.62}$ known plaintexts and the time complexity is $2^{123.62} + 2^{124.62}/4 + 2^{119.24} \simeq 2^{123.8}$ encryptions. To reduce the data complexity we can allow degrees of freedom for an active S-box in the last round of the characteristic. There exist 25 different $\alpha \in GF(2^8)$ such that $\Pr[S(x) \oplus S(x \oplus \texttt{0x76}) = \alpha] = 2^{-6}$. Let us consider the same characteristic in Appendix A.2 while $X_{16}^S(1) \oplus X_{12}'^S(1) = \alpha$. The probability for such a characteristic is $2^{-113.82} \cdot (25 \cdot 2^{-6}) = 2^{-115.17}$ which indicates that the data complexity decreases to $2^{64+57.59} = 2^{121.59}$ with the cost of increasing the time complexity to $25 \cdot (2^{121.59}/4 + 2^{115.17}) \simeq 2^{124.23}$ encryptions.

4.2 Slide Cryptanalysis of LED-64

Let us recall that the difference between RC_r and RC_{r+4} of LED-64 has nonzero value only in the second column. Then we start by looking at the state after `AC` in an arbitrary round r to investigate different scenarios. Since we are interested in characteristic with less active S-boxes, one may think the best case happens when all active nibbles in $X_r'^I \oplus X_{r+4}^I$ get canceled by the difference DRC_r to bypass `SC` with probability one. We note it can activate four nibbles $X_{r+1}'^A(1,5,9,13) \oplus X_{r+5}^A(1,5,9,13)$. Next each active nibble propagates to a different column after `SR` and makes all nibbles active at $X_{r+1}'^M \oplus X_{r+5}^M$, which is against our goal. Another choice is to have just one active nibble $X_r'^A(1) \oplus X_{r+4}^A(1)$ such that after `MC` it transfers to four nibbles which cancel the three out of four nibbles differences injected by round constants in the next round. It can be considered as an iterative 1-round characteristic. After testing this approach we found that due to the differences between round constants we cannot utilize this iteratively for the round constants chosen by the designers,

because it works just for one round. So we chose a moderate strategy to find the best pattern. First we try to hesitate activate the nibbles from the first, third and forth columns of $X_r'^I \oplus X_{r+4}^I$ since we cannot cancel them by the differences of round constants. Secondly we aim to cancel as many nibbles in the second column as possible. To find the best pattern we start from the middle rounds and try to extend it in both backward and forward directions. It is proved in [20] that any differential characteristic over four consecutive rounds of the cipher has at least 25 active S-boxes and probability at most 2^{-50}. Thanks to the differences in the round-constants, the best characteristic we found for four rounds has probability 2^{-27} and just 13 active S-boxes. This demonstrates the superiority of our model in comparison with differential cryptanalysis of LED-64.

Key Recovery. The 4-round differential characteristic illustrated in Appendix A.1 has probability 2^{-27}. This property enables us to retrieve the key of 8-round reduced of LED-64 with the same technique described in Sect. 2.3. The cryptanalysis requires $2^{32+13.5} = 2^{45.5}$ known plaintexts and the time complexity is roughly $2^{45.5} + 2^{46.5}/2 + 2^{27} \simeq 2^{46.5}$ encryptions. To have less data complexity we can consider a truncated type differential for the last round. The differences 2, c, d and 6 can be transferred through the S-box to the set of differences $\mathcal{A}_1 = \{3, 5, 6, a, c, d, e\}$, $\mathcal{A}_2 = \{2, 5, 7, 8, 9, a, e\}$, $\mathcal{A}_3 = \{1, 2, 3, 4, 7, a, b\}$ and $\mathcal{A}_4 = \{2, 6, 8, b, c, f\}$ respectively. If $X_5^I \oplus P' = \Delta_{in}$ holds for the given Δ_{in} in Appendix A.1 then the truncated difference $X_8^S \oplus X_4'^S \in \{0a_1000a_2000a_3000a_400 | a_i \in \mathcal{A}_i, 1 \leq i \leq 4\}$ with its corresponding truncated difference $\Delta_{out} = X_8^M \oplus X_4'^M$ holds with probability 2^{-19}. Using this characteristic the data complexity decreases to $2^{32+9.5} = 2^{41.5}$ known plaintexts while the time complexity increases to $6 \cdot 7^3(2^{41.5}/2 + 2^{19}) \simeq 2^{51.5}$ encryptions.

5 Conclusion

In this paper we provide a new insight into slide cryptanalysis which is illustrated by cryptanalysis of step-reduced block ciphers Zorro and LED-64. We describe a new framework to enhance slide cryptanalysis against general Even-Mansour scheme with one key in a probabilistic setting. Our method exploits some features from related-key differential cryptanalysis to build a kind of differential characteristic that is applicable in the single key model. In the related-key cryptanalysis model [2,22] one can consider the encryption under unknown secret keys but with a determined difference, which allows attacker to control the data difference by differences injected by the key difference. The probabilistic slide cryptanalysis presented in this paper is inspired by the same idea but instead of using two different keys it slides a copy of encryption to take advantage of the round constant differences in a single key model. Since known statistical cryptanalysis is not affected by the values of round constants, choosing their values usually has not been taken into account by the designers of block ciphers (for example look at [24,31]). In this work we shed more light on how round constants can potentially weaken the security of the cipher. One possible direction

of future research is to inquire the application of probabilistic slide cryptanalysis against other block ciphers based on the general Even-Mansour scheme with a single key like PRINCE, PRINTcipher and 3-WAY [12].

Acknowledgments. The author wishes to thank Kaisa Nyberg for the helpful discussions, detailed comments and insightful suggestions that significantly improved the quality of this paper. The program for finding the characteristic used for cryptanalysis of Zorro was performed on the Triton computing cluster provided by the Aalto University Science-IT programme. The work of Hadi Soleimany is supported by Helsinki Doctoral Program in Computer Science - Advanced Computing and Intelligent Systems (HECSE). He was partially supported by the Nokia Foundation which is also gratefully acknowledged.

References

1. Aoki, K., Kobayashi, K., Moriai, S.: Best differential characteristic search of FEAL. In: Biham, E. (ed.) FSE 1997. LNCS, vol. 1267, pp. 41–53. Springer, Heidelberg (1997)
2. Biham, E.: New types of cryptanalytic attacks using related keys. J. Cryptol. **7**(4), 229–246 (1994)
3. Biham, E., Dunkelman, O., Keller, N.: Improved slide attacks. In: Biryukov, A. (ed.) FSE 2007. LNCS, vol. 4593, pp. 153–166. Springer, Heidelberg (2007)
4. Biham, E., Shamir, A.: Differential cryptanalysis of des-like cryptosystems. In: Menezes, A., Vanstone, S.A. (eds.) CRYPTO 1990. LNCS, vol. 537, pp. 2–21. Springer, Heidelberg (1990)
5. Biryukov, A.: Analysis of involutional ciphers: khazad and anubis. In: Johansson, T. (ed.) FSE 2003. LNCS, vol. 2887, pp. 45–53. Springer, Heidelberg (2003)
6. Biryukov, A., Nikolić, I.: Automatic search for related-key differential characteristics in byte-oriented block ciphers: application to AES, camellia, khazad and others. In: Gilbert, H. (ed.) EUROCRYPT 2010. LNCS, vol. 6110, pp. 322–344. Springer, Heidelberg (2010)
7. Biryukov, A., Wagner, D.: Slide attacks. In: Knudsen, L.R. (ed.) FSE 1999. LNCS, vol. 1636, pp. 245–259. Springer, Heidelberg (1999)
8. Biryukov, A., Wagner, D.: Advanced slide attacks. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 589–606. Springer, Heidelberg (2000)
9. Bogdanov, A., Knežević, M., Leander, G., Toz, D., Varıcı, K., Verbauwhede, I.: SPONGENT: a lightweight hash function. In: Preneel, B., Takagi, T. (eds.) CHES 2011. LNCS, vol. 6917, pp. 312–325. Springer, Heidelberg (2011)
10. Bogdanov, A., Knezevic, M., Leander, G., Toz, D., Varici, K., Verbauwhede, I.: Spongent: the design space of lightweight cryptographic hashing. IEEE Trans. Comput. **62**(10), 2041–2053 (2013)
11. Borghoff, J., Canteaut, A., Güneysu, T., Kavun, E.B., Knezevic, M., Knudsen, L.R., Leander, G., Nikov, V., Paar, C., Rechberger, C., Rombouts, P., Thomsen, S.S., Yalçın, T.: PRINCE –a low-latency block cipher for pervasive computing applications. In: Wang, X., Sako, K. (eds.) ASIACRYPT 2012. LNCS, vol. 7658, pp. 208–225. Springer, Heidelberg (2012)
12. Daemen, J., Govaerts, R., Vandewalle, J.: A new approach to block cipher design. In: Anderson, R. (ed.) FSE 1993. LNCS, vol. 809. Springer, Heidelberg (1994)

13. Dinur, I., Dunkelman, O., Keller, N., Shamir, A.: Improved Linear Sieving Techniques withApplications to Step-Reduced LED-64. Cryptology ePrint Archive, Report 2013/634 (2013). http://eprint.iacr.org/
14. Dinur, I., Dunkelman, O., Keller, N., Shamir, A.: Key recovery attacks on 3-round even-mansour, 8-step led-128, and full aes^2. In: Sako, K., Sarkar, P. (eds.) ASIACRYPT 2013, Part I. LNCS, vol. 8269, pp. 337–356. Springer, Heidelberg (2013)
15. Even, S., Mansour, Y.: A construction of a cipher from a single pseudorandom permutation. In: Matsumoto, Tsutomu, Imai, Hideki, Rivest, Ronald L. (eds.) ASIACRYPT 1991. LNCS, vol. 739, pp. 210–224. Springer, Heidelberg (1993)
16. Furuya, Soichi: Slide attacks with a known-plaintext cryptanalysis. In: Kim, Kee-cheon (ed.) ICISC 2001. LNCS, vol. 2288, p. 214. Springer, Heidelberg (2002)
17. Gérard, B., Grosso, V., Naya-Plasencia, M., Standaert, F.-X.: Block ciphers that are easier to mask: how far can we go? In: Bertoni, G., Coron, J.-S. (eds.) CHES 2013. LNCS, vol. 8086, pp. 383–399. Springer, Heidelberg (2013)
18. Gorski, M., Lucks, S., Peyrin, T.: Slide attacks on a class of hash functions. In: Pieprzyk, J. (ed.) ASIACRYPT 2008. LNCS, vol. 5350, pp. 143–160. Springer, Heidelberg (2008)
19. Guo, J., Nikolic, I., Peyrin, T., Wang, L.: Cryptanalysis of Zorro. Cryptology ePrint Archive, Report 2013/713 (2013). http://eprint.iacr.org/
20. Guo, J., Peyrin, T., Poschmann, A., Robshaw, M.: The led block cipher. In: Preneel, B., Takagi, T. (eds.) CHES 2011. LNCS, vol. 6917, pp. 326–341. Springer, Heidelberg (2011)
21. Isobe, T., Shibutani, K.: Security analysis of the lightweight block ciphers XTEA, LED and piccolo. In: Susilo, W., Mu, Y., Seberry, J. (eds.) ACISP 2012. LNCS, vol. 7372, pp. 71–86. Springer, Heidelberg (2012)
22. 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)
23. Knudsen, L., Leander, G., Poschmann, A., Robshaw, M.J.B.: PRINTcipher: a block cipher for ic-printing. In: Mangard, S., Standaert, F.-X. (eds.) CHES 2010. LNCS, vol. 6225, pp. 16–32. Springer, Heidelberg (2010)
24. Leander, G., Abdelraheem, M.A., AlKhzaimi, H., Zenner, E.: A cryptanalysis of PRINTcipher: the invariant subspace attack. In: Rogaway, P. (ed.) CRYPTO 2011. LNCS, vol. 6841, pp. 206–221. Springer, Heidelberg (2011)
25. Matsui, M.: Linear cryptanalysis method for DES cipher. In: Helleseth, T. (ed.) EUROCRYPT 1993. LNCS, vol. 765, pp. 386–397. Springer, Heidelberg (1994)
26. Matsui, M.: On correlation between the order of s-boxes and the strength of DES. In: De Santis, A. (ed.) EUROCRYPT 1994. LNCS, vol. 950, pp. 366–375. Springer, Heidelberg (1995)
27. Nikolić, I., Wang, L., Wu, S.: Cryptanalysis of round-reduced LED. In: Moriai, S. (ed.) FSE 2013. LNCS, vol. 8424, pp. 112–130. Springer, Heidelberg (2014)
28. Onions, P.: On the strength of simply-iterated feistel ciphers with whitening keys. In: Naccache, D. (ed.) CT-RSA 2001. LNCS, vol. 2020, pp. 63–69. Springer, Heidelberg (2001)
29. Phan, R.C.-W.: Advanced slide attacks revisited: realigning slide on des. In: Dawson, E., Vaudenay, S. (eds.) Mycrypt 2005. LNCS, vol. 3715, pp. 263–276. Springer, Heidelberg (2005)
30. 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)

31. Soleimany, H., Blondeau, C., Yu, X., Wu, W., Nyberg, K., Zhang, H., Zhang, L., Wang, Y.: Reflection cryptanalysis of prince-like ciphers. In: Moriai, S. (ed.) FSE 2013. LNCS, vol. 8424, pp. 71–91. Springer, Heidelberg (2014)

A Appendix

A.1 Characteristic Used for Cryptanalysis of LED-64

State	Difference	Probability
$\Delta_{in} = X_5^I \oplus P'$	0250060b33010700	
$X_5^A \oplus X_1'^A$	0150000b30010100	
$X_5^S \oplus X_1'^S$	07c0000860070900	2^{-12}
$X_5^R \oplus X_1'^R$	07c0008007600090	
$X_5^M \oplus X_1'^M$	0100051007000500	
$X_6^A \oplus X_2'^A$	0600001000000000	
$X_6^S \oplus X_2'^S$	0c0000d000000000	2^{-5}
$X_6^R \oplus X_2'^R$	0c000d0000000000	
$X_6^M \oplus X_2'^M$	0800020007000200	
$X_7^A \oplus X_3'^A$	0f00000000000000	
$X_7^S \oplus X_3'^S$	0100000000000000	2^{-2}
$X_7^R \oplus X_3'^R$	0100000000000000	
$X_7^M \oplus X_3'^M$	040008000b000200	
$X_8^A \oplus X_4'^A$	02000c000d000600	
$X_8^S \oplus X_4'^S$	0500050002000b00	2^{-8}
$X_8^R \oplus X_4'^R$	05005000000200b0	
$\Delta_{out} = X_8^M \oplus X_4'^M$	5754defa31c7aa9d	

A.2 Characteristic Used for Cryptanalysis of Zorro

State	Difference	Probability
$\Delta_{in} = X_5^I \oplus P'$	00000000d52c6f72120a92b50c8c2eee	
$X_5^S \oplus X_1'^S$	00000000d52c6f72120a92b50c8c2eee	1
$X_5^A \oplus X_1^A$	04040420d52c6f72120a92b50c8c2eee	
$X_5^R \oplus X_1'^R$	040404202c6f72d592b5120aee0c8c2e	
$X_5^M \oplus X_1'^M$	000000001f125aa13e0edd9375ce6fe3	
$X_6^S \oplus X_2'^S$	000000001f125aa13e0edd9375ce6fe3	1
$X_6^A \oplus X_2'^A$	040404201f125aa13e0edd9375ce6fe3	
$X_6^R \oplus X_2'^R$	04040420125aa11fdd933e0ee375ce6f	
$X_6^M \oplus X_2'^M$	00000000bf6bd16389fc90921e2f14af	

(*Continued*)

(*Continued*)

State	Difference	Probability
$X_7^S \oplus X_3'^S$	00000000bf6bd16389fc90921e2f14af	1
$X_7^A \oplus X_3'^A$	04040420bf6bd16389fc90921e2f14af	
$X_7^R \oplus X_3'^R$	040404206bd163bf909289fcaf1e2f14	
$X_7^M \oplus X_3'^M$	8aec0b72d60e6d4ebec81f40b273b80b	
$X_8^S \oplus X_4'^S$	0fa38ee5d60e6d4ebec81f40b273b80b	$2^{-24.41}$
$X_8^A \oplus X_4'^A$	03af8285d60e6d4ebec81f40b273b80b	
$X_8^R \oplus X_4'^R$	03af82850e6d4ed61f40bec80bb273b8	
$X_8^M \oplus X_4'^M$	000000003507b4c92e8f3e0b02b88be1	
$X_9^S \oplus X_5'^S$	000000003507b4c92e8f3e0b02b88be1	1
$X_9^A \oplus X_5'^A$	0c0c0c603507b4c92e8f3e0b02b88be1	
$X_9^R \oplus X_5'^R$	0c0c0c6007b4c9353e0b2e8fe102b88b	
$X_9^M \oplus X_5'^M$	ced6ce9ba1604f0b4fa84ad6f4af9817	
$X_{10}^S \oplus X_6'^S$	fff8ff04a1604f0b4fa84ad6f4af9817	2^{-26}
$X_{10}^A \oplus X_6'^A$	f3f4f364a1604f0b4fa84ad6f4af9817	
$X_{10}^R \oplus X_6'^R$	f3f4f364604f0ba14ad64fa817f4af98	
$X_{10}^M \oplus X_6'^M$	00000000faff9b463e0b8c3d0a6d0f8e	
$X_{11}^S \oplus X_7'^S$	00000000faff9b463e0b8c3d0a6d0f8e	1
$X_{11}^A \oplus X_7'^A$	0c0c0c60faff9b463e0b8c3d0a6d0f8e	
$X_{11}^R \oplus X_7'^R$	0c0c0c60ff9b46fa8c3d3e0b8e0a6d0f	
$X_{11}^M \oplus X_7'^M$	009981d1e86caf9d79f3819d60a6b64f	
$X_{12}^S \oplus X_8'^S$	0082b813e86caf9d79f3819d60a6b64f	2^{-21}
$X_{12}^A \oplus X_8'^A$	0486bc33e86caf9d79f3819d60a6b64f	
$X_{12}^R \oplus X_8'^R$	0486bc336caf9de8819d79f34f60a6b6	
$X_{12}^M \oplus X_8'^M$	720000000b1fb040a0a822e77f636c39	
$X_{13}^S \oplus X_9'^S$	190000000b1fb040a0a822e77f636c39	2^{-6}
$X_{13}^A \oplus X_9'^A$	1d0404200b1fb040a0a822e77f636c39	
$X_{13}^R \oplus X_9'^R$	1d0404201fb0400b22e7a0a8397f636c	
$X_{13}^M \oplus X_9'^M$	005b0b997c321cb90de0bad468a52a1b	
$X_{14}^S \oplus X_{10}'^S$	004838077c321cb90de0bad468a52a1b	2^{-19}
$X_{14}^A \oplus X_{10}'^A$	044c3c277c321cb90de0bad468a52a1b	
$X_{14}^R \oplus X_{10}'^R$	044c3c27321cb97cbad40de01b68a52a	
$X_{14}^M \oplus X_{10}'^M$	ff000000ae7be7ce745b6bfeb2cca1a1	
$X_{15}^S \oplus X_{11}'^S$	aa000000ae7be7ce745b6bfeb2cca1a1	2^{-7}
$X_{15}^A \oplus X_{11}'^A$	ae040420ae7be7ce745b6bfeb2cca1a1	
$X_{15}^R \oplus X_{11}'^R$	ae0404207be7ceae6bfe745ba1b2cca1	
$X_{15}^M \oplus X_{11}'^M$	0076f953447ad32bfbc96dc0a06a35cc	
$X_{16}^S \oplus X_{12}'^S$	000b84ed447ad32bfbc96dc0a06a35cc	$2^{-15.83}$
$X_{16}^A \oplus X_{12}'^A$	1c17980d447ad32bfbc96dc0a06a35cc	
$X_{16}^R \oplus X_{12}'^R$	1c17980d7ad32b446dc0fbc9cca06a35	
$\Delta_{out} = X_{16}^M \oplus X_{12}'^M$	1720c72a9351b2f0f3a4e09fb071b7f0	

Improved Linear Sieving Techniques with Applications to Step-Reduced LED-64

Itai Dinur[1], Orr Dunkelman[2,4(✉)], Nathan Keller[3,4], and Adi Shamir[4]

[1] Département d'Informatique, École Normale Supérieure, Paris, France
[2] Computer Science Department, University of Haifa, Haifa, Israel
orrd@cs.haifa.ac.il
[3] Department of Mathematics, Bar-Ilan University, Ramat Gan, Israel
[4] Computer Science Department, The Weizmann Institute, Rehovot, Israel

Abstract. In this paper, we present advanced meet-in-the-middle (MITM) attacks against the lightweight block cipher LED-64, improving the best known attacks on several step-reduced variants of the cipher in both single-key and related-key models. In particular, we present a known-plaintext attack on 2-step LED-64 with complexity of 2^{48} and a related-key attack on 3-step LED-64 with complexity of 2^{49}. In both cases, the previously known attacks have complexity of 2^{60}, i.e., only 16 times faster than exhaustive key search.

While our attacks are applied to the specific scheme of LED-64, they contain several general methodological contributions: First, we present the *linear key sieve* technique, which allows to exploit linear dependencies between key bits to obtain filtering conditions in MITM attacks on block ciphers. While similar ideas have been previously used in the domain of hash functions, this is the first time that such a technique is applied in block cipher cryptanalysis. As a second contribution, we demonstrate for the first time that a *splice-and-cut* attack (which so far seemed to be an inherently chosen-plaintext technique) can be used in the known-plaintext model, with data complexity which is significantly below the code-book size. Finally, we extend the differential MITM attack on AES-based designs, and apply it independently in two stages from both sides of the cipher, while using the linear key sieve and other enhancements.

Keywords: Cryptanalysis · LED · AES · Even-Mansour · Meet-in-the-middle attack · Splice-and-cut · Known plaintext splice-and-cut

1 Introduction

Meet-in-the-middle (MITM) attacks on block ciphers were first introduced more than 30 years ago [18]. A block cipher is vulnerable to such attacks if it is possible

I. Dinur—Some of the work presented in this paper was done while the first author was a postdoctoral researcher at the Weizmann Institute, Israel.

O. Dunkelman—The second author was supported in part by the German-Israeli Foundation for Scientific Research and Development through grant No. 2282-2222.6/2011.

N. Keller—The third author was supported by the Alon Fellowship.

C. Cid and C. Rechberger (Eds.): FSE 2014, LNCS 8540, pp. 390–410, 2015.
DOI: 10.1007/978-3-662-46706-0_20

to independently compute a variable of its inner state from the encryption and decryption sides without having to guess the full key. The value of this inner variable (or variables) is used to efficiently sieve the key suggestions obtained from both sides and mount an efficient attack. This motivated block cipher designers to incorporate relatively complex key schedule algorithms into the design, thus assuring very quick diffusion of the full key into the state and making the cipher resistant to MITM attacks. On the other hand, complex key schedule algorithms are difficult to implement in resource-constrained environments (such as RFID tags and wireless sensors). Thus, in recent years, with the rise of lightweight cryptography, designers have proposed many schemes with simplified key schedule algorithms. At the extreme end of the scale lie block ciphers such as LED-64 [13], Zorro [11] and PRINCE [4] which have no key schedule at all, and simply XOR the key to the internal state of the cipher several times during the encryption process.

Naturally, the tendency to simplify the key schedule of block ciphers was accompanied by the development of interesting new techniques in MITM attacks in order to break these schemes. One of the most notable techniques is splice-and-cut [1,20], initially applied to hash functions, but quickly shown to be applicable to block ciphers as well. Splice-and-cut attacks are adaptations of Merkle and Hellman's attack on 2K-3DES [18] to single encryption. The main idea is to obtain the encryptions of several chosen plaintexts in order to view the first and the last rounds of the cryptosystem as consecutive rounds. As a result, the adversary can split the cipher into two parts in an unconventional way, and mount an efficient MITM attack in cases where such an attack seems difficult otherwise.

Another important technique used in several MITM attacks exploits the ability to independently and efficiently compute linear combinations of variables of the inner state (rather than the actual variables) of the cipher from the encryption and decryption sides. The MITM attack is then applied through a linear layer of the block cipher, in the same way as in a number of attacks on SHA-1, SHA-2 and AES-based designs, e.g., [2,5,14,19]. This technique is often referred to as indirect partial matching; it also corresponds to the linear case of the sieve-in-the-middle method, described in [6].

In this paper, we also exploit some additional filtering conditions derived from the linear dependencies in the key suggestions that are computed from both sides of the MITM attack. Similar ideas where previously used in MITM attacks against SHA-1 in [2,15] (which exploited the linear message schedule in the hash function to obtain filtering conditions). However, this is the first time that the technique is used in the domain of block ciphers, and we call it a *linear key sieve*.

Although we do not expect such linear dependencies to exist in block ciphers with complex non-linear key schedules, they are much more likely to occur in lightweight designs with simple key schedules. We apply the linear key sieve technique to LED-64 [13], and use it to improve some of the best previously known attacks on step-reduced variants of this block cipher in both the single-key and related-key models.

The lightweight block cipher LED was presented at CHES 2011 [13], and due to its elegant AES-based design, it has been the target of significant cryptanalytic effort in the past few years. In the single-key chosen plaintext model, the best previously known attack on 2 steps of LED-64 (reduced from the full 8) was presented in [14], and we reduce its time complexity from 2^{56} to 2^{48}. Both the previous attack and our new attack apply the splice-and-cut technique in order to mount a MITM attack on the cipher. The main element that enables us to improve the previous attack of [14] is the linear key sieve, which we use in order to filter the key suggestions obtained during the attack in a more efficient way.

In addition to the chosen plaintext attack, we describe a known plaintext attack on 2-step LED-64 which improves both the time and memory complexities of the previous best attack[1] (presented at Asiacrypt 2013 [8]) from 2^{60} to 2^{48}. The main novelty of this attack is that it uses, for the first time, the splice-and-cut technique (which seems to require chosen messages in an inherent way[2]) in the *known plaintext model*. Once again, in this attack we use the linear key sieve technique, and it enables our known plaintext attack to maintain the same running time as our chosen plaintext attack.

Finally, in the stronger related-key model, we analyze 3-step LED-64, on which the best previously known attack[3] used a classical differential method [17]. In this model, we extend the differential MITM attack[4] on AES-based designs of [5], and apply it in two independent stages from both sides of the cipher. We use this technique, it in addition to the linear key sieve, in order to improve the previous attack of [17] in all the complexity parameters of time/memory/data from 2^{60} to 2^{49}. We summarize and compare our results in Table 1.

The paper is organized as follows: in Sect. 2, we briefly describe LED-64, and in Sect. 3, we describe the notations and conventions that are used in this paper. Our new chosen plaintext, known plaintext and related-key attacks are described in Sects. 4, 5 and 6, respectively. Finally, we conclude in Sect. 7.

2 Description of LED-64

LED [13] is a 64-bit block cipher built using several public permutations, interleaved with round-key additions over $GF(2)$ (i.e., XOR operations).

This construction is generally known as iterated Even-Mansour (see Fig. 1), which generalizes the original one-round construction [10]. In the case of LED,

[1] We also mention the attack on 3-step LED-64 given in [8]. However, despite its theoretical significance, the attack is non-practical with respect to the memory complexity (which is 2^{60}) and it is only about 16 times faster than exhaustive search.

[2] As explicitly required in the first generic application of the splice-and-cut technique to block ciphers [22], and in subsequent splice-and-cut attacks on concrete block ciphers such as in [14].

[3] We also mention the related-key attack on 4-step LED-64 given in [17]. However, despite its theoretic interest, it is very marginal with respect to the time, memory and data complexities, which are all about 2^{63}.

[4] Differential MITM is a classical technique used, e.g., in [9], and was recently formalized in [15].

Table 1. Attacks of step-reduced LED-64

Reference	Model	Steps	Time	Data	Memory
[17]	Single-key	2	2^{60}	2^{60} CP	2^{60}
[14]	Single-key	2	2^{56}	2^{8} CP	2^{8}
Section 4.2	Single-key	2	2^{48}	2^{16} CP	2^{17}
Section 5	Single-key	2	2^{48}	2^{48} KP	2^{48}
[8]	Single-key	3	$2^{60.2}$	2^{49} KP	2^{60}
[17]	Related-Key	3	2^{60}	2^{60} CP	2^{60}
Section 6.2	Related-Key	3	2^{49}	2^{49} CP	2^{49}
[17]	Related-Key	4	2^{63}	2^{63} CP	2^{63}

The data complexity is given in chosen plaintexts (CP), or in known plaintexts (KP).

the public permutations are called steps, and each step is composed of 4 rounds. A round of LED uses an AES-like design, where given a 64-bit input X, it is treated as a concatenation of 16 four-bit nibbles $X[0]\|X[1]\|\ldots\|X[15]$, which are (conceptually) arranged in a 4×4 array:

$X[0]$	$X[1]$	$X[2]$	$X[3]$
$X[4]$	$X[5]$	$X[6]$	$X[7]$
$X[8]$	$X[9]$	$X[10]$	$X[11]$
$X[12]$	$X[13]$	$X[14]$	$X[15]$

The round function uses 4 AES-like mappings AddConstants (AC), SubCells (SC), ShiftRows (SR), and MixColumnsSerial (MCS). The structural properties of these mappings (given below) are similar to those of the AES mappings AddRoundKey, SubBytes, ShiftRows and MixColumns, respectively, and these are the only properties which are exploited by our attacks. For the complete implementation details of the LED mappings, refer to [13].

1. AddConstants adds (over GF(2)) a round-dependent constant to each cell of the first two columns.
2. SubCells applies a 4-bit Sbox to every cell of the internal state.
3. ShiftRows rotates each cell located in row i by i positions to the left.
4. MixColumnsSerial independently applies an MDS (Maximum Distance Separable) matrix over $GF(2^4)$ to each column.

LED has two main variants, LED-64 and LED-128, which differ according to the key size. In this paper, we are mainly interested in the 64-bit version, which uses 32 rounds (or 8 steps). The key schedule of LED-64 simply adds the 64-bit key K before rounds $4i + 1$ for $i = 0, 1, \ldots, 7$, and again after the final round.

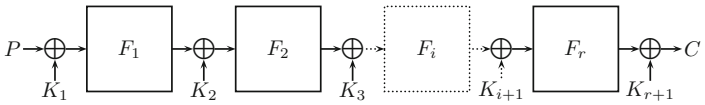


Fig. 1. Iterated Even-Mansour

3 Notations and Conventions

Notations. We denote by R_r the public function of round r of LED-64 (without the key addition), i.e., given a 64-bit state X, $R_r(X) \triangleq MCS(SR(SC(AC(X)))$. We denote by F_i the public function of step i, i.e., given a 64-bit state X, $F_i(X) \triangleq R_{4i+4}(R_{4i+3}(R_{4i+2}(R_{4i+1}(X))))$. The functions R_r^{-1} and F_i^{-1} are defined as the inverses of R_r and F_i, respectively.

Given a plaintext-ciphertext pair (P, C), we denote the state after r encryption rounds by X_r (e.g., $X_0 = P$ and X_1 is the state after one round of LED-64). In order to simplify our notation, we define $\hat{X}_{4i} = X_{4i} \oplus K$, and so $F_i(\hat{X}_{4i}) = X_{4(i+1)}$. In some of our attacks, in addition to obtaining plaintext-ciphertext pairs, we independently evaluate the public step function F_i (for some i) on some input states $\hat{Y}_{4i}$, and we define $F_i(\hat{Y}_{4i}) = Y_{4(i+1)}$ (i.e., $\hat{Y}_{4i} = Y_{4i} \oplus K_i$).

We denote the j'th column of X_i by $X_{i,|j|}$, i.e., $X_{i,|1|}$ is composed of nibbles $\{1, 5, 9, 13\}$. Similarly, we denote by $X_{i,|j,l|}$ columns j and l of X_i. We define two more column-related sets: the first is the *diagonal* $X_{i,/j/}$ which is composed of the nibbles in X_i corresponding to the places after the ShiftRows operation on column j, e.g., $X_{i,/1/}$ is composed of nibbles {1,4,11,13}. The second set is *inverse diagonal* $X_{i,\backslash j\backslash}$ which is composed of the nibbles in the positions of column j after having applied the inverse ShiftRows operation (see Fig. 2).

Conventions. Throughout this paper, we use the standard conventions and calculate the time complexity of our attacks in terms of evaluations of the full cipher, while calculating their memory complexity in terms of 64-bit words (since the block size of LED-64 is 64 bits). Some of the attacks presented in this paper involve basic linear algebra algorithm (such as solving a system of linear equations with a few dozen variables[5] over $GF(2)$). Since our attacks execute these

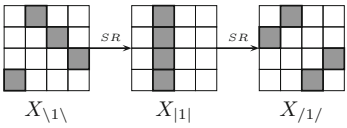


Fig. 2. An inverse diagonal, a column and a diagonal

[5] In order to reduce the $O(n^3)$ bit operations required to solve a system of n linear equations $Ax = b$ in the online phase of the attack, we compute A^{-1} offline. Given the vector b in the online phase, we simply compute $x = A^{-1}b$ in $O(n^2)$ bit operations.

basic linear algebra algorithms no more than a few times per evaluation of the full cipher, we can ignore them in our time complexity analysis.

4 The Linear Key Sieve Technique – A Chosen Plaintext Attack on 2-Step LED-64

In this section, we introduce the linear key sieve technique, and apply it (combined with splice-and-cut) to 2-step LED-64. Our attack improves the time complexity of the previously best known attack on 2-step LED-64 [14] (which is based on the techniques of [19]) from 2^{56} to 2^{48}.

To simplify the description of the technique, before presenting the full 2-step attack, we introduce the linear key sieve in a simple example of a basic MITM attack on 1-step LED-64.

4.1 A Meet-in-the-Middle Attack on 1-step LED-64

We describe a basic MITM attack on 4-round (1-step) LED-64, using a single known plaintext-ciphertext pair ($P = X_0, C = \hat{X}_4$). The attack is based on a few simple and well-known observations on AES-based constructions:

1. The order of the linear operations ARK and MCS is interchangeable, i.e., $MCS^{-1}(ARK^{-1}(C)) = ARK'^{-1}(MCS^{-1}(C))$, where ARK' adds the key $K' \triangleq MCS^{-1}(K)$ to the state. As in many attacks on AES-based constructions, we can thus apply MCS^{-1} to the ciphertext C, and "peel-off" the last-round MCS operation.
2. Given an inverse diagonal $X_{r,\backslash i\backslash}$ (at the beginning of any round r), we can fully compute the diagonal $SR(SC(AC(R_r(X_r))))_{/i/} = MCS^{-1}(X_{r+2})_{/i/}$ after the first 7 operations. Similarly, given a diagonal $MCS^{-1}(X_{r+2})_{/i/}$, we can fully compute the inverse diagonal $X_{r,\backslash i\backslash}$ after the 7 inverse operations. Such a permutation, mapping 4 nibbles to 4 nibbles of the state through a "round and a half" is called a "Super-Sbox" of LED (a term which was originally defined for AES [7]).
3. Given knowledge of any b_1 bits of the state X, we can compute the values of b_1 linear combinations (over $GF(2)$) on the bits of the state $MCS(X)$.

Observation 2 implies that, given any two inverse diagonals $\hat{X}_{0,\backslash i,j\backslash}$, we can compute 32 bits of $MCS^{-1}(X_2)_{/i,j/}$, namely the two diagonals of indices i and j. Combined with Observation 3, they correspond to 32 linear combinations of the bits of the state X_2, spanning a subspace of dimension 32. Similarly, from the decryption side, the knowledge of three diagonals of $MCS^{-1}(ARK^{-1}(C))_{/l,m,n/}$ gives us the knowledge of 48 bits of X_2, namely, the corresponding inverse diagonals $X_{2,\backslash l,m,n\backslash}$, as shown in Fig. 3.

Since the full state contains 64 bits, the intersection of these two subspaces is a linear subspace of dimension[6] $32 + 48 - 64 = 16$. The basis of this subspace

[6] In general, the dimension of the intersection can be bigger. However, in AES-based constructions (where the MixColumns operation is implemented using an MDS matrix), the dimension of the intersection is exactly $32 + 48 - 64 = 16$.

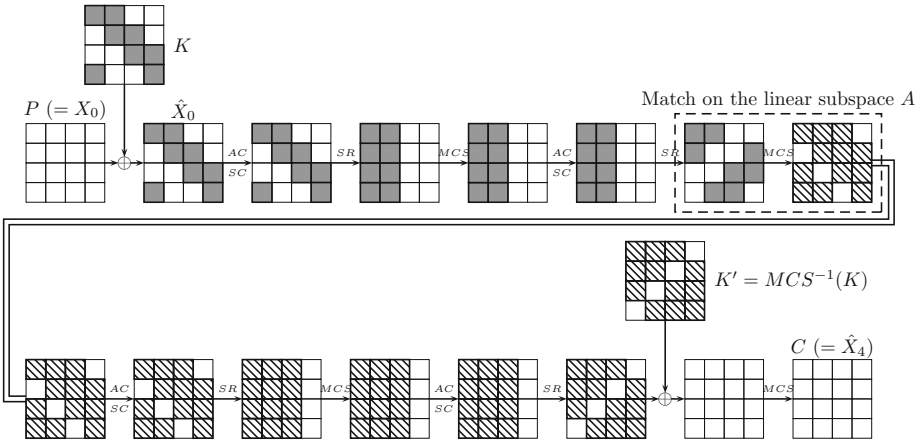


Fig. 3. A meet-in-the-middle attack on 1-Step LED-64

gives rise to 16 linearly independent combinations in the bits of the state X_2 (denoted by A, as shown in Fig. 3) whose values are computable independently from the known 2 inverse diagonals (from the encryption side), and from the known 3 diagonals (from the decryption side).

The computation of the $|A| = 16$ joint linear combinations, requires the knowledge of $16 \cdot 2 = 32$ bits of K from one side and $16 \cdot 3 = 48$ bits of $K' = MCS^{-1}(K)$ from the other side. For the correct guess of these bits, the values of the joint linear combintations in A match with probability 1, whereas for an arbitrary incorrect suggestion, based on standard randomness assumptions, the values of the linear combinations in A match with probability $2^{-|A|} = 2^{-16}$. Therefore, incorrect suggestions in the MITM attack are discarded (as in the related attacks of [5,14,19]).

In standard MITM attacks, if one treats the keys K and K' as independent, there are $2^{32+48} = 2^{80}$ suggestions for the key from both sides. Given the 16 bits of the sieve on the state, we expect about 2^{64} suggestions to remain, which we need to further analyze. Thus, in its current form, this attack is not faster than exhaustive search.

The Linear Key Sieve. A possible solution to the problem of insufficient filtering is to use an additional plaintext-ciphertext pair, which will offer an additional 16-bit filtering condition. However, we now introduce the *linear key sieve* which provides these 16 bit-conditions with *no additional data*, by exploiting the linear dependency of K and K'. This observation (exploited by [2] in the domain of hash functions, but which [14] did not use) is at the basis of our improved attack on 2-step LED-64 (described in the next section), and all the other attacks presented in the paper.

Recall that the MITM attack requires 32 bits of K and 48 bits of $K' = MCS^{-1}(K)$, which are linear combinations in the bits of K. Just as the state subspaces intersect (and allow us to obtain the sieve on the state), so do the two

linear subspaces spanned by the linear combinations in the bits of K and K' that we guess. The intersection is a linear subspace of dimension $32 + 48 - 64 = 16$, giving rise to 16 linearly independent combinations in the bits of K (denoted by B_3), whose values are computable independently from both sides. The linear combinations in B_3 are used in order to filter our wrong key guesses (for the right key they agree with probability 1, and for wrong key guesses they agree with probability $2^{-|B_3|}$), and thus we call this set of linear combinations a *linear key sieve*.

Let B_1 be additional 16 linear combinations of K needed for the attack[7] (i.e., B_1 and B_3 determine the partial encryption of two inverse diagonals), and let B_2 be additional 32 linear combinations of K' needed for the attack (i.e., B_2 and B_3 determine the partial decryption of three diagonals). Our MITM attack is composed of an outer loop, iterating over the linear subspace spanned by B_3, where in each iteration, we independently iterate over the linear subspaces spanned by B_1 and B_2.[8] Thus, we *force* the key suggestions obtained from both sides of the attack to agree on B_3 (rather than randomly achieving agreement). The resultant attack (described for two arbitrarily columns i, j in the forward direction and three arbitrary columns l, m, n in the backward direction) is as follows:

1. For each value of the 16 linear combinations of B_3:
   (a) For each value of the 16 linear combinations of B_1:
      i. Compute $K_{\backslash i,j \backslash}$, and use it to compute $\hat{X}_{0,\backslash i,j \backslash}$.
      ii. Compute the values of the 16 linear combinations of A, and store them in a sorted list L, next to the value of the 16 linear combinations of B_1.
   (b) For each value of the 32 linear combinations of B_2:
      i. Compute $K'_{/l,m,n/}$, and use it to compute $MCS^{-1}(ARK^{-1}(C))_{/l,m,n/}$.
      ii. Compute the values of the 16 linear combinations of A, and search for matches in the list L.
      iii. For each match:
         A. Obtain the value of the 16 linear combinations of B_1.
         B. Compute K using linear algebra, given the values of B_1, B_2 and B_3.
         C. Test K using a trial encryption, and if it succeeds, return the key.

The list L contains 2^{16} values, and thus we expect a single match for each value of the 16 linear combinations of A in Step 1.(b).ii. This implies that the expected time complexity of each iteration of Step 1 is about 2^{32}, and thus the expected time complexity of the whole attack is 2^{48}, which is faster than exhaustive search by a factor of 2^{16}. The memory complexity of the attack is about 2^{16}, which is required for storing the list L. Note that the memory needed for storing L in each iteration of Step 1 can be reused.

[7] There are many options for the basis B_1, and we choose one arbitrarily.

[8] We note that the approach of taking out shared bits to an outer loop is a very common practice in saving memory. The main improvement in this attack compared to [14] is the fact that we take out shared *linear combinations*.

4.2 The Improved Chosen Plaintext Single-Key Attack on 2-Step LED-64

Our attack on 2-Step LED-64 follows the same general structure as the previous one of [14,19]. We use the splice-and-cut technique on 4 rounds (1 step) of the cipher. The advantage of our attack comes from the linear key sieve (missing from [14]), i.e., using the linear relations between K and $K' = MCS^{-1}(K)$.

In order to apply splice-and-cut to 2-step LED-64, we (as in the previous attack [14]) partition the indices of 64-bit state into two lexicographically ordered sets, S_1 and S_2. The attack requires the encryptions of $2^{|S_1|}$ plaintexts $P^1, P^2, \ldots$ in which all the bits of S_2 are fixed to zero,[9] and the bits of S_1 range over all the possible values. Independently, we evaluate the first 4 key-less rounds of LED-64 (i.e., F_1) on $2^{|S_2|}$ inputs $\hat{Y}_0^1, \hat{Y}_0^2, \ldots$ in which the bits of S_1 are fixed to zero, and the bits of S_2 range over all the possible values, and obtain the corresponding outputs $Y_4^1, Y_4^2, \ldots$.

The aim of the splice-and-cut technique is to find a plaintext $P^i = X_0^i$ and an internal state $\hat{Y}_0^j$ such that $\hat{X}_0^i = X_0^i \oplus K = \hat{Y}_0^j$. This occurs if and only if P^i and K "agree" on the bits of S_1 and $\hat{Y}_0^j$ and K "agree" on the bits of S_2, or formally $P^i_{|S_1} = K_{|S_1}$ and $\hat{Y}^j_{0|S_2} = K_{|S_2}$ (where $W_{|S}$ denotes the $|S|$-bit value of the word W on the indices of the ordered set S.). In other words, each plaintext P^i is associated with a potential value of $K_{|S_1}$, and each state $\hat{Y}_0^j$ is associated with a potential value of $K_{|S_2}$. For each value of K, there is only one such correct pair, and we denote its plaintext by P, and its evaluated state by $\hat{Y}_0 = P \oplus K$. Thus, finding the pair (i, j) is equivalent to recovering the key K.

Consider the correct pair $(P, \hat{Y}_0)$. Applying F_1 to $\hat{Y}_0$ gives $Y_4 = X_4$. Thus, if we consider all $F_1(\hat{Y}_0^j) = Y_4^j$ values, one of them is indeed X_4. As a result, given the ciphertext C that corresponds to P and X_4, the splice-and-cut technique reduces the problem to attacking 4 rounds of LED-64. Hence, as shown in Fig. 4, we choose the bits of S_2 to be the 48 bits of 3 inverse diagonals (and thus S_1 contains the 16 bits of the remaining inverse diagonal). As a result, we can take each Y_4^j value (associated with a suggestion for 3 inverse diagonals of K) and continue its partial encryption (as per Observation 2), resulting in the knowledge of three diagonals before the MCS operation of round 6, i.e., three diagonals of $MCS^{-1}(Y_6)$. This knowledge gives rise to suggestions for the values of 48 linear combinations of Y_6. Independently, we try all possible values of 32 bits of two

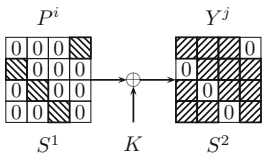


Fig. 4. The sets S_1 and S_2

[9] The technique can be applied in a similar way for any fixed value of the bits of S_2.

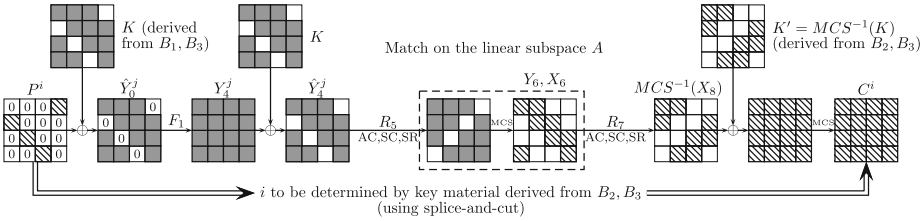


Fig. 5. Our improved chosen plaintext attack on 2-step LED-64

diagonals of K', and partially decrypt all the ciphertexts to obtain suggestions of 32 bits of X_6.

Now, we can apply the 4-round attack (as $Y_6 = X_6$), obtaining a sieve of the state of $48 + 32 - 64 = 16$ linear combinations on the state bits (each independently computed from a different side), denoted by A. To obtain more filtering conditions, we again use the linear key sieve: each ciphertext is associated with 16 bits of K, and thus each suggestion for the 32 bits of X_6 is associated with the values of $16 + 32 = 48$ linear combinations on the bits of K. Since each suggestions for Y_6 depends on 48 bits of K, we have $48 + 48 - 64 = 32$ linear combinations which we use as a linear key sieve (denoted by B_3). Similarly to our basic MITM attack, we complement the 32 linear combinations of B_3 to a basis of the subspace spanned by the 48 bits of S_2 using 16 additional linear combinations, denoted by B_2. Similarly, we complement B_3 to a basis of the 48-dimensional subspace spanned by the 16 bits of S_1 and the 32 bits of the 2 diagonals of K', using 16 additional linear combinations, denoted by B_1.

The attack proceeds as follows (see Fig. 5):

1. Request the encryptions of the 2^{16} plaintexts P^i such that $P^i_{\backslash 0,1,2\backslash} = 0$, and store all plaintext-ciphertext pairs.
2. For each value of the 32 linear combinations of B_3:
   (a) For each value of the 16 linear combinations of B_2:
      i. Using the values of B_2 and B_3, compute a suggestion for $K_{\backslash 0,1,2\backslash}$.
      ii. Let $\hat{Y}_0^j$ be the state such that $Y^j_{0,\backslash 3\backslash} = 0$ and $Y^j_{0,\backslash 0,1,2\backslash} = K_{\backslash 0,1,2\backslash}$.
      iii. Compute $F_1(\hat{Y}_0^j) = Y_4^j$, and use the partial knowledge of $K_{\backslash 0,1,2\backslash}$ to obtain a suggestion for the values of the 16 linear combinations of A. Store the suggestion for the values of A in a sorted list L, next to the value of B_2.
   (b) For each value of the 16 linear combinations of B_1:
      i. Using the values of B_1 and B_3, compute a suggestion for $K_{\backslash 3\backslash}$ and $K'_{/0,1/}$.
      ii. Let (P^i, C^i) be the plaintext-ciphertext pair such that $P^i_{\backslash 3\backslash} = K_{\backslash 3\backslash}$ (recall that $P^i_{\backslash 0,1,2\backslash} = 0$).
      iii. Compute a suggestion for the values of the 16 linear combinations of A using C^i and $K'_{/0,1/}$.
      iv. Search for the suggestion for the values of A in the list L.

v. For each match, obtain the value of B_2, use it to obtain a suggestion for the key K and test it using a trial encryption. If the trial succeeds, return the key.

The data complexity of the attack is 2^{16} chosen plaintexts. The memory complexity of the attack is about 2^{17}, required in order to store the plaintext-ciphertext pairs, and in order to store the list L. Since L contains 2^{16} values per iteration of Step 2.(a), we expect one match in Step 2.(b).iv, and a total of 2^{16} matches per iteration of Step 2. Thus, the time complexity of an iteration of Step 2 is equivalent to about 2^{16} 2-step LED-64 encryptions, and the total time complexity of the attack is about 2^{48} encryptions.

5 Known-Plaintext Splice-and-Cut Attack – Application to 2-Step LED-64

In this section, we describe the first splice-and-cut attack (which was believed to be inherently a chosen message technique) in the known plaintext model. In particular, we improve the time and memory complexities of the best attack on 2-step LED-64 [8] in the known plaintext model from 2^{60} to 2^{48}. In fact, due to the efficient sieving techniques, this attack has the same time complexity as our chosen plaintext attack, presented in the previous section.

The classical chosen-plaintext splice-and-cut attack considers two sets: a set of plaintexts P^i of size 2^m and a set of states $\hat{Y}_0^j$ of size 2^{n-m} such that the set $P^i \oplus \hat{Y}_0^j$ covers F_2^n, i.e., there exists a unique pair (i,j) such that $P^i \oplus \hat{Y}_0^j = K$. However, the coverage of F_2^n by $P^i \oplus \hat{Y}_0^j$ cannot be guaranteed with probability 1 if the set of 2^m plaintexts (such that $1 \gg 2^m \gg 2^n$) is composed of arbitrary elements. Consequently, in our known-plaintext variant, the existence of some pair (i,j) such that $P^i \oplus \hat{Y}_0^j = K$ is now guaranteed with a good probability by the birthday paradox (assuming that the plaintexts are uniformly distributed).[10]

The main difficulty in applying our splice-and-cut technique in the known plaintext model, is that $P_{|S_2}$ is not fixed, and thus we "lose" the association of an input $\hat{Y}_0^j$ to a potential value of $K_{|S_2}$. Consequently, for each $\hat{Y}_0^j$, we need to guess additional key bits in order to obtain the necessary filtering conditions on Y_6. Since an input $\hat{Y}_0^j$ is now associated with many key guesses, we are forced to evaluate fewer such inputs in order to obtain an efficient attack. Indeed, while in the chosen plaintext attack, we evaluated 2^{48} such inputs, here we evaluate only 2^{16} inputs, and for each one we guess the 32 bits of $K_{\backslash 0,1\backslash}$ and obtain a suggestion for the values of 32 linear combinations on Y_6. According to the birthday paradox, we need 2^{48} known plaintexts-ciphertext pairs (P^i, C^i) $(i \in \{1, 2, \ldots, 2^{48}\})$ to obtain with good probability a pair (i,j), such that $P^i \oplus \hat{Y}_0^j = K$.

The values $\hat{Y}_0^1, \hat{Y}_0^2, \ldots$ which we evaluate are defined by the 16-dimensional linear subspace $\{\hat{Y}_0^j | MCS^{-1}(\hat{Y}_0^j)_{/1,2,3/} = 0\}$. Thus, a plaintext P^i is implicitly

[10] We note that in case that the plaintexts are not uniformly distributed, we can apply a similar attack assuming that the ciphertexts are uniformly distributed (by exchanging the roles of encryption and decryption).

associated with a partial key value $P^i_{/1,2,3/} = K'_{/1,2,3/}$. This implies that we can partially decrypt C^i in order to obtain the values of 48 bits of X^i_6 without additional key guesses. Note that this is not the traditional way in which splice-and-cut is applied, as all previous attacks (including our previous chosen plaintext attack) directly partitioned the bits of the state into two groups S_1 and S_2. Instead, in this attack we work with linear subspaces constructed to exploit the linear dependency between K and K' in order to be able to partially decrypt C^i without additional key guesses (whereas our previous chosen plaintext attack did not directly exploit this dependency).

As in the chosen plaintext attack, we have a sieve on the state of $32 + 48 - 64 = 16$ linear combinations on the bits of $X_6 = Y_6$ which are independently computable from each side, and we denote it by A_1. Each value of the linear combinations of A_1, computed from an input $\hat{Y}^j_0$, is associated with a suggestion for $K_{\backslash 0,1\backslash}$, and as a result, we can also compute $Y^j_{0,\backslash 0,1\backslash} = \hat{Y}^j_{0,\backslash 0,1\backslash} \oplus K_{\backslash 0,1\backslash}$. Since $P^i = X^i_0$, the 32-bit value of $Y^j_{0,\backslash 0,1\backslash}$ can be directly matched with each plaintext, and we denote this sieve by A_2. We note that since $MCS^{-1}(\hat{Y}^j_0)_{/1,2,3/} = 0$, then Y^j (and $Y^j_{0,\backslash 0,1\backslash}$), can only attain 2^{16} values, and thus effectively, the 32 bits of A_2 give only 16 bits of filtering.

From the decryption side, each value computed from (P^i, C^i) is associated with a suggestion for $K'_{/1,2,3/}$. Thus, we can identify $32 + 48 - 64 = 16$ linear combinations (i.e., a linear key sieve) which are independently computable from each side, and we denote this sieve by B. In total, we have 48 bits of filtering, as A_1 gives us 16 bits, A_2 (effectively) gives us 16 bits of filtering, and B gives us additional 16 bits.

The attack proceeds as follows:

1. For each of the 2^{16} possible values of $\hat{Y}^j_0$ such that $MCS^{-1}(\hat{Y}^j_0)_{/1,2,3/} = 0$:
   (a) Compute $F_1(\hat{Y}^j_0) = Y^j_4$
   (b) For each of the 2^{32} values of $K_{\backslash 0,1\backslash}$:
      i. Compute the values of the 16 linear combinations of A_1, the values of the bits of A_2, and the values of the 16 linear combinations of B. Store these values in a sorted list L, next to the values of $K_{\backslash 0,1\backslash}$.
2. For each plaintext-ciphertext pair (P^i, C^i):
   (a) Assume that $K'_{/1,2,3/} = MCS^{-1}(P^i)_{/1,2,3/}$, compute the values of A_1, A_2 and B, and search the list L for matches.
   (b) For each match, obtain $K_{\backslash 0,1\backslash}$, compute a suggestion for the full key K, and test it using a trial encryption. If the trial succeeds return the key.

Since we evaluate 2^{16} inputs $\hat{Y}^j_0$, we expect that after obtaining the encryptions of about 2^{48} arbitrary plaintexts, we will have a pair $(P^i, \hat{Y}^j_0)$ satisfying $P^i \oplus K = \hat{Y}^j_0$, which will enable us to recover the correct key.[11] Thus, the expected

[11] We note that unlike our chosen plaintext attack (that succeeds in finding the key with probability 1), our known plaintext attack succeeds with probability of about 0.63, which is the probability suggested by the birthday paradox (given 2^{48} known plaintext-ciphertext pairs).

data complexity of the attack is 2^{48} known plaintexts. Since the size of L is 2^{48}, and (effectively) we have 48 bits of filtering conditions, we expect one match for each plaintext in Step 2.(a), and thus the time and memory complexities of the attack are 2^{48} as well.

Data/Memory/Time Tradeoffs. The attack can be applied by evaluating $\hat{Y}_0^1, \hat{Y}_0^2, \ldots$ in a linear subspace $\{\hat{Y}_0^j | MCS^{-1}(\hat{Y}_0^j)_{/1,2,3/} = const\}$ for *any* value of *const*, with the appropriate changes to the algorithm. This suggests data/memory/time tradeoffs for the attack, in which we first obtain the plaintext-ciphertext pairs and store them in memory (thus exchanging the order of some computations performed in Steps 1 and 2 above). Then, we perform the computations of Step 1 (previously used in order to build the list L) "on-the-fly", and search the stored data for matches according to the filtering conditions.

By using this revised algorithm, we can reduce the amount of known plaintexts by a factor of 2, repeating the attack once with $const = 0$, and once for an arbitrary value $const \neq 0$. In this case, the time complexity of the attack is increased by a factor of 2, and its memory complexity is reduced by a factor of 2. Similarly, for any $d \leq 16$, given 2^{48-d} known plaintexts, the time complexity of the attack increases to 2^{48+d}, and its memory complexity decreases to 2^{48-d}. In other words, we can attack the scheme (with a similar success rate) given $D \leq 2^{48}$ known plaintexts and time complexity T such that $DT = 2^{96}$ (with memory $M = D$).

6 Enhanced Differential Meet-in-the-Middle – A Related-Key Attack on 3-Step LED

In this section, we describe a related-key attack on 3-step LED-64 using two related keys. The attack improves the previously best known attack on this scheme, described in [17], in all the complexities parameters of time/memory/data from 2^{60} to 2^{49}. The 3-step attack uses the linear key sieve technique on top of a rather involved differential MITM attack. Before describing the full attack, we describe a simple differential MITM attack on 1-step LED-64 in the single-key model, which serves as background to our 3-step related-key attack. We note that in the case of 1-step LED-64, our simple attack is closely related to the attack on Pelican-MAC described in [5].

6.1 A Differential Single-Key Meet-in-the-Middle Attack on 1-Step LED-64

The simple differential MITM attack on 4-round (1-step) LED-64 requires 2 chosen plaintexts, and its memory and time complexities are slightly more than 2^{16}. In order to obtain an efficient attack, we compute and use the difference distribution tables for the LED Super-Sboxes (spanning the third round and part of the fourth round), using similar technique to those published in [7,12,16].

Namely, given an entry $[\delta_{in}, \delta_{out}]$, specifying a 16-bit input/output difference to the Super-Sbox, the table stores the actual pairs of values that conform to this entry. A single full table can be easily computed during preprocessing in 2^{32} simple operations, and it requires about 2^{32} words of memory. However, in this simple attack, the output difference δ_{out} to each Super-Sbox is fixed by the ciphertexts, and thus we only need a single column in each table. Such a column is computed in the online phase (after obtaining the encryptions of the plaintexts) in 2^{16} time, using 2^{16} storage.

The details of the attack are given below.

1. Obtain the encryptions of P^1 and P^2, chosen such that $(P^1 \oplus P^2)_{\backslash 0,1,2 \backslash} = 0$.
2. Denote $\Delta_r \triangleq X_r^1 \oplus X_r^2$, i.e., Δ_r is the 64-bit state difference after round r.
3. Compute 3 columns in the difference distribution tables of the LED Super-Sboxes, corresponding to the output differences specified by the three diagonals $MCS^{-1}(\Delta_4)_{/1,2,3/}$.
4. For each value of $K'_{/0/}$:
   (a) Use C^1 to compute $X^1_{2,\backslash 0 \backslash}$ and C^2 to compute $X^2_{2,\backslash 0 \backslash}$, and calculate $\Delta_{2,\backslash 0 \backslash}$.
   (b) Given $\Delta_{2,\backslash 0 \backslash}$, and the fact that $MCS^{-1}(\Delta_2)_{/0,1,2/} = 0$, calculate the full Δ_2 by solving a system of linear equations.
   (c) Given the input difference Δ_2 and $MCS^{-1}(\Delta_4)$, use the Super-Sbox (partial) difference distribution tables to obtain the possible values for $ARK'(MCS^{-1}(C^1))_{/1,2,3/}$, and use these values to obtain suggestions for the full K', thus obtaining suggestions for K.
   (d) Test each suggestion for K using a trial encryption, and if it succeeds return the key.

Since we expect, on average, a single suggestion for K in Step 3.(c), the time complexity of Step 3 is about 2^{16}, which is the time complexity of the full attack. The memory complexity is about 2^{16}, required in order to store the columns of the difference distribution tables for the Super-Sboxes.

We note that this attack is faster than the attack of Sect. 4.1 since the collision on the diagonals of P allows us to obtain a suggestion for the full key after guessing only 16 key bits (enabling us to compute the full state difference Δ_2). This observation will be further exploited in the next section.

6.2 The Improved Related-Key Attack on 3-Step LED-64

In this section, we describe the details of our related-key attack on 12-round (3-step) LED-64 which assumes that we can obtain the encryptions of plaintexts with keys K_1 and K_2, such that $K_1 \oplus K_2 = \Delta$ is known (in fact, as explained below, we only need the ability to partially choose the value of Δ). During the online phase of the attack, we request the encryptions of 2^{48} chosen plaintexts encrypted with K_1 and 2^{48} (different) chosen plaintexts encrypted with K_2. The time and memory complexities of the attack are about 2^{49}.

Our attack uses the basic framework of [17] for related-key attacks on iterated Even-Mansour schemes. Namely, we ask for the encryptions of pairs of plaintexts $P^{i,1}$ and $P^{i,2} = P^{i,1} \oplus \Delta$, encrypted with K_1 and $K_2 = K_1 \oplus \Delta$, respectively. Considering the encryption process of these two plaintexts, the input difference to the public F_1 function is zero, which implies that the output difference of F_1 is zero, and after the second key addition, the input difference to F_2 is $\hat{\Delta}_4 = \Delta$ (namely, $\hat{X}_4^{i,1} \oplus \hat{X}_4^{i,2} = \hat{\Delta}_4 = \Delta$). At this point, our algorithm diverges from [17] (which assumes that the function F_2 has some high-probability differential characteristic).

Our attack is based on the 4-round differential MITM attack of the previous section. Here, we apply a similar attack to F_3 by processing plaintext pairs whose ciphertexts collide on a diagonal (before the MCS operation). However, unlike the 4-round attack, we do not know the input difference to the public function on which we perform the MITM attack (F_3 in this case). Thus, we preprocess F_2 by computing and storing pairs of inputs to this function with an input difference of Δ, and we use a birthday argument to claim that one of the input pairs will collide with a plaintext pair with high probability. However, this is insufficient, as we were not able to find parameters for which the differential MITM algorithm yields an efficient attack, under the constraint that we store sufficiently many pairs of inputs to F_2 (in order to obtain a collision with a processed plaintext pair). The problem is that we need to guess too many key bits before we can compute filtering conditions and eliminate some key guesses.

In order to reduce the number of key guesses, we again exploit collisions on (inverse) diagonals, but this time at the input of F_3. Namely, we require that the difference at the output of F_2 on some (inverse) diagonals cancels out after the key addition. More specifically, we preprocess F_2, and find 2^{31} pairs of inputs to this function, ($\hat{Y}_4^{j,1}$ and $\hat{Y}_4^{j,2} = \hat{Y}_4^{j,1} \oplus \Delta$), such that their output difference is equal to Δ in two inverse diagonals (i.e., $(Y_8^{j,1} \oplus Y_8^{j,2})_{\backslash 0,1 \backslash} = \Delta_{\backslash 0,1 \backslash}$, implying that $(\hat{Y}_8^{j,1} \oplus \hat{Y}_8^{j,2})_{\backslash 0,1 \backslash} = 0$). We expect that 2^{31} such pairs indeed exist, since there are 2^{63} unordered input pairs to F_2 with an input difference of Δ, and based on standard randomness assumptions, about 2^{31} of them satisfy the 32-bit condition on the output difference (a slightly smaller number will only slightly increase the complexity of the attack). The trivial algorithm to find these pairs is to exhaustively enumerate all the 2^{63} input pairs, however, this is wasteful as it requires 2^{64} time (and our model does not allow free precomputation). Instead, we use yet again a MITM approach, and devise an auxiliary preprocessing algorithm that finds the required pairs in about 2^{48} time. In order to run efficiently, our algorithm requires that 48 specific bits of Δ are zero, and thus we have to assume that we can partially choose the key difference[12] Δ. The details of this preprocessing step are specified in Appendix A.

[12] The related key attack of [17] required that we can choose the full 64 bits of Δ, so our attack is slightly more generic.

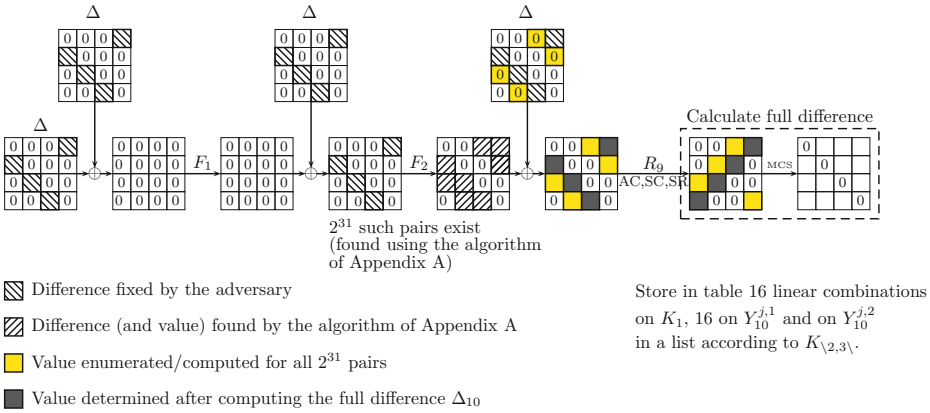


Fig. 6. The preprocessing algorithm (depicting differences)

The full preprocessing algorithm (which calls the algorithm of Appendix A) is given below and depicted in Fig. 6.[13] In this attack, we assume that we have computed during preprocessing the full difference distribution tables for the LED Super-Sboxes, using about 2^{32} simple operations and 2^{32} memory.

1. Use the auxiliary preprocessing algorithm of Appendix A to obtain 2^{31} pairs $(\hat{Y}_4^{j,1}, \hat{Y}_4^{j,2} = \hat{Y}_4^{j,1} \oplus \Delta)$, such that $(Y_8^{j,1} \oplus Y_8^{j,2})_{\backslash 0,1\backslash} = \Delta_{\backslash 0,1\backslash}$.
2. For each of the 2^{31} pairs $(\hat{Y}_4^{j,1}, \hat{Y}_4^{j,2})$:
   (a) For each value of $(K_1)_{\backslash 2\backslash}$:
      i. Compute $(K_2)_{\backslash 2\backslash} = (K_1)_{\backslash 2\backslash} \oplus \Delta_{\backslash 2\backslash}$. Assume that $\hat{Y}_4^{j,1}$ is encrypted with K_1 and $\hat{Y}_4^{j,2}$ is encrypted with K_2, and let $\Delta_r = Y_r^{j,1} \oplus Y_r^{j,2}$. Use the LED Super-Sbox to compute $MCS^{-1}(\Delta_{10})_{/2/} = MCS^{-1}(Y_{10}^{j,1})_{/2/} \oplus MCS^{-1}(Y_{10}^{j,2})_{/2/}$.
      ii. Since $MCS^{-1}(\Delta_{10})_{/0,1/} = 0$ and $MCS^{-1}(\Delta_{10})_{/2/}$ is known, we know 48 bits of $MCS^{-1}(\Delta_{10})$. We now assume that $\Delta_{10,\backslash 0\backslash} = 0$, and compute the full $MCS^{-1}(\Delta_{10})$ using linear algebra (as MCS is a linear operation).
      iii. Use the difference distribution table for the LED Super-Sbox, and the knowledge of $\hat{\Delta}_8 \triangleq \hat{Y}_8^{j,1} \oplus \hat{Y}_8^{j,2}$ and $MCS^{-1}(\Delta_{10})$ to compute suggestions for the actual $MCS^{-1}(Y_{10}^{j,1})_{/3/}$, $MCS^{-1}(Y_{10}^{j,2})_{/3/}$ and $(K_1)_{\backslash 3\backslash}$.
      iv. We now have suggestions for 32 bits of K_1, 32 bits of $MCS^{-1}(Y_{10}^{j,1})$ and 32 bits of $MCS^{-1}(Y_{10}^{j,2})$. During the online phase, we will obtain suggestions for (specific) 48 bits of K_1', 48 bits of $X_{10}^{i,1}$ and 48 bits of $X_{10}^{i,2}$ (the encryption values of some $P^{i,1}$ and $P^{i,2}$ after 10 rounds). Thus, we can compute the values of a total of 48 linear combinations

[13] Since the preprocessing algorithm of this attack is more involved than in the previous attacks, we describe it separately from the online algorithm.

to serve as filtering bits: 16 linear combinations on the bits of K_1 (a linear key sieve), 16 linear combinations on $Y_{10}^{j,1}$ and 16 linear combinations on $Y_{10}^{j,2}$. We store the values of these linear combinations in a sorted list L, next to $(K_1)_{\backslash 2,3\backslash}$.

As described in Appendix A, the time complexity of Step 1 is about 2^{48} evaluations of 1-step LED. On average, we expect a single suggestion for the values computed in Step 2.(a).iii (using the difference distribution table of the LED Super-Sbox). Thus, we perform only a few simple operations for each of the 2^{31} pairs (computed in Step 1) and the 2^{16} possible values of $(K_1)_{\backslash 2\backslash}$, implying that the total time complexity of the preprocessing algorithm is about 2^{48} evaluations of 1-step LED. In order to slightly reduce the data complexity of the online algorithm (at the expense of using slightly more memory), we repeat Step 2.(a) twice for each ordered pair, exchanging the roles of $\hat{Y}_4^{j,1}$ and $\hat{Y}_4^{j,2}$ (i.e., by assuming that $\hat{Y}_4^{j,1}$ is encrypted with K_2). Thus, the time complexity of the preprocessing algorithm is about 2^{49} evaluations of 1-step LED, and its memory complexity is about 2^{49}.

The online algorithm of the attack is given below (and depicted in Fig. 7)

1. For 2^{48} arbitrary values of the plaintext $P^{i,1}$:
   (a) Ask for the encryption of $P^{i,1}$ under the key K_1, and for the encryption of $P^{i,2} = P^{i,1} \oplus \Delta$ under the key $K_2 = K_1 \oplus \Delta$. Let $\Lambda_r \triangleq X_r^{j,1} \oplus X_r^{j,2}$.
   (b) Compute $C'^{i,1} = MCS^{-1}(C^{i,1})$ and $C'^{i,2} = MCS^{-1}(C^{i,1})$. Check if $(C'^{i,1} \oplus C'^{i,2})_{/0/} = 0$, and if not, return to Step 1.
   (c) For each value of $(K_1')_{/1/}$:
      i. Compute $(K_2')_{/1/} = (K_1')_{/1/} \oplus MCS^{-1}(\Delta)_{/1/}$ and use the LED Super-Sbox to compute $\Lambda_{10,\backslash 1\backslash} = X_{10,\backslash 1\backslash}^{i,1} \oplus X_{10,\backslash 1\backslash}^{i,2}$.

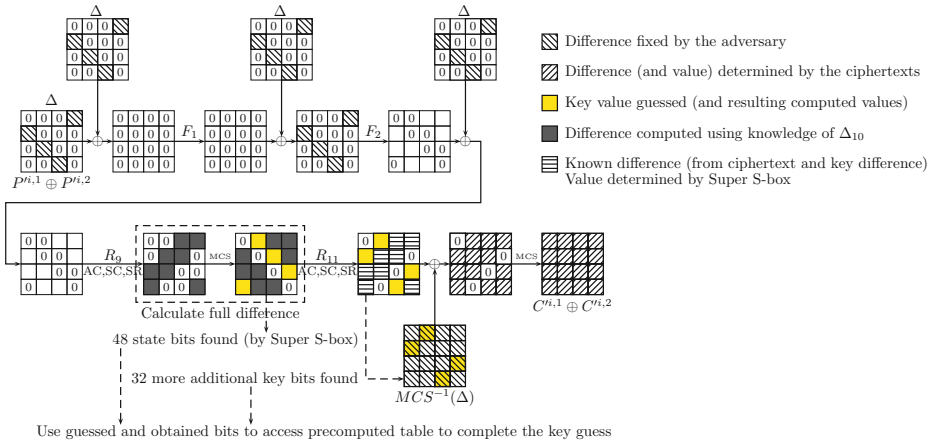


Fig. 7. The preprocessing algorithm (depicting differences)

ii. Assume that $MCS^{-1}(\Lambda_{10})_{/0,1/} = 0$, and compute Λ_{10} (using the fact that $\Lambda_{10,\backslash 0\backslash} = 0$ and $\Lambda_{10,\backslash 1\backslash}$ is known).
iii. Use the difference distribution table for the LED Super-Sbox, and the knowledge of Λ_{10} and $(C'^{j,1} \oplus C'^{j,2})$ to compute suggestions for $X^{i,1}_{10,\backslash 2\backslash}$, $X^{i,2}_{10,\backslash 2\backslash}$ and $(K'_1)_{/2/}$, and similarly compute suggestions for $X^{i,1}_{10,\backslash 3\backslash}$, $X^{i,2}_{10,\backslash 3\backslash}$ and $(K'_1)_{/3/}$.
iv. From the knowledge of 48 bits of K'_1, 48 bits of $X^{i,1}_{10}$ and 48 bits of $X^{i,2}_{10}$, compute the filtering values of the 48 linear combinations, and look for matches in the list L.
v. For each match, obtain $(K_1)_{\backslash 2,3\backslash}$, compute a suggestion for K_1 and test it.

A pair $((P^{i,1}, C^{i,1}), (P^{i,2}, C^{i,2}))$ passes the 16-bit filtering condition of Step 1.(b) with probability 2^{-16}, and thus we expect to process about $2^{48-16} = 2^{32}$ pairs in Step 1.(c). As we store 2^{32} ordered input pairs of values for F_2 with an input difference of Δ, and each of the processed 2^{32} plaintext pairs has a difference of Δ at the input to F_2, we expect a collision between these two groups of pairs. For such a collision, the assumptions made during the online and preprocessing algorithms hold ($\Delta_{10,\backslash 0\backslash} = 0$ is assumed in preprocessing Step 2.(a).ii, and $MCS^{-1}(\Lambda_{10})_{/0,1/} = 0$ is assumed in online Step 1.(c).ii). Thus, this collision will yield a match in Step 1.(c).iv, suggesting the correct value of K_1.

Since we expect a single suggestion for the values computed in Step 1.(c).iii, we perform a few simple operations for each of the 2^{32} processed pairs in Step 1.(b) and the 2^{16} possible values of $(K'_1)_{/1/}$, on which we iterate in Step 1.(c). Thus, the total time complexity of the online algorithm is about 2^{49} evaluations of 1-step LED. Including the preprocessing time, the total time complexity of the attack is about 2^{49} evaluations of 2-step LED (which is a bit less than 2^{49} evaluations of the full 3-step scheme). The data complexity of the attack is 2^{49}, and its memory complexity is 2^{49}, required in order to store the list L.

A Single-Key Attack on a Variant of 2-Step LED with Independent Keys. Consider a variant of 8-round LED, where the three round keys K_1, K_2 and K_3 are independent. We now show how to adapt the above attack to this scheme with about the same time/memory/data complexities. We note that this construction has a similar structure to the block cipher AES^2 [3], which is composed of two key-less AES-128 permutations such that K_1 and K_2 are added before and after the first permutation and K_3 is added after the second permutation. Since our techniques only exploit the AES structure of the LED step function, this attack can also be applied to AES^2, reduced from 20 rounds to 8 rounds (with the complexity of the attack adjusted to the 128-bit cipher).

In this attack, we select Δ in a similar way to the related-key attack above. However, in the single-key attack we cannot inject a pair of different messages such that they have a zero difference at the input to F_1. Thus, we now preprocess F_1 (instead of F_2) and find 2^{31} pairs of inputs to this function, $\hat{Y}^{j,1}_0, \hat{Y}^{j,2}_0 = \hat{Y}^{j,1}_0 \oplus \Delta$, such that $(\hat{Y}^{j,1}_4 \oplus \hat{Y}^{j,2}_4)_{\backslash 0,1\backslash} = 0$. Consequently, in the online algorithm,

we request the encryptions of 2^{48} pairs of plaintexts with an input difference of Δ, and apply the differential MITM technique to F_2.

Another difference between this attack and the previous related-key attack on LED-64, is that now K_1, K_2 and K_3 are independent, and thus we do not have any filtering conditions on the key when matching the suggestions during the MITM phase (i.e., we do not have a linear key sieve). This implies that if we use only one pair of plaintexts, the complexity of the attack will be at least 2^{64}. In order to speed up the attack, we use the same idea used in [8] in the attack on AES^2: we choose an arbitrary non-zero difference $\Delta' \neq \Delta$, and ask for the encryption of another plaintext $P^{i,1} \oplus \Delta'$ for each pair $P^{i,1}$ and $P^{i,1} \oplus \Delta$. Similarly, we attach another evaluation of $\hat{Y}_0^{j,1} \oplus \Delta'$ to each evaluated pair $\hat{Y}_0^{j,1}$ and $\hat{Y}_0^{j,1} \oplus \Delta$. This allows us to obtain the required filtering values such that the attack has similar time/memory/data complexities to the related-key attack. The full details of the filtering technique are found in [8].

7 Conclusions

In this paper, we introduced various techniques in MITM attacks including the linear key sieve technique, a known plaintext splice-and-cut attack, and new techniques for differential MITM. We applied these techniques to step-reduced LED-64 and obtained the best known results on this block cipher, both in the single-key and related-key models. Although our techniques are mainly applied to LED-64, we believe that they will be useful in the analysis of other cryptosystems, in particular AES-based cryptosystems and lightweight block ciphers.

References

1. Aoki, K., Sasaki, Y.: Preimage attacks on one-block MD4, 63-step MD5 and more. In: Avanzi, R.M., Keliher, L., Sica, F. (eds.) SAC 2008. LNCS, vol. 5381, pp. 103–119. Springer, Heidelberg (2008)
2. Aoki, K., Sasaki, Y.: Meet-in-the-middle preimage attacks against reduced SHA-0 and SHA-1. In: Halevi, S. (ed.) CRYPTO 2009. LNCS, vol. 5677, pp. 70–89. Springer, Heidelberg (2009)
3. Bogdanov, A., Knudsen, L.R., Leander, G., Standaert, F.-X., Steinberger, J., Tischhauser, E.: Key-alternating ciphers in a provable setting: encryption using a small number of public permutations. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT 2012. LNCS, vol. 7237, pp. 45–62. Springer, Heidelberg (2012)
4. Borghoff, J., Canteaut, A., Güneysu, T., Kavun, E.B., Knezevic, M., Knudsen, L.R., Leander, G., Nikov, V., Paar, C., Rechberger, C., Rombouts, P., Thomsen, S.S., Yalçın, T.: PRINCE – A low-latency block cipher for pervasive computing applications. In: Wang, X., Sako, K. (eds.) ASIACRYPT 2012. LNCS, vol. 7658, pp. 208–225. Springer, Heidelberg (2012). [21]
5. Bouillaguet, C., Derbez, P., Fouque, P.-A.: Automatic search of attacks on round-reduced AES and applications. In: Rogaway, P. (ed.) CRYPTO 2011. LNCS, vol. 6841, pp. 169–187. Springer, Heidelberg (2011)

6. Canteaut, A., Naya-Plasencia, M., Vayssière, B.: Sieve-in-the-middle: Improved MITM attacks. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013, Part I. LNCS, vol. 8042, pp. 222–240. Springer, Heidelberg (2013)
7. Daemen, J., Rijmen, V.: Understanding two-round differentials in AES. In: De Prisco, R., Yung, M. (eds.) SCN 2006. LNCS, vol. 4116, pp. 78–94. Springer, Heidelberg (2006)
8. Dinur, I., Dunkelman, O., Keller, N., Shamir, A.: Key recovery attacks on 3-round even-mansour, 8-step LED-128, and full AES 2. In: Sako, K., Sarkar, P. (eds.) ASIACRYPT 2013, Part I. LNCS, vol. 8269, pp. 337–356. Springer, Heidelberg (2013)
9. Dunkelman, O., Sekar, G., Preneel, B.: Improved meet-in-the-middle attacks on reduced-round DES. In: Srinathan, K., Rangan, C.P., Yung, M. (eds.) INDOCRYPT 2007. LNCS, vol. 4859, pp. 86–100. Springer, Heidelberg (2007)
10. Even, S., Mansour, Y.: A construction of a cipher from a single pseudorandom permutation. J. Cryptol. **10**(3), 151–162 (1997)
11. Gérard, B., Grosso, V., Naya-Plasencia, M., Standaert, F.-X.: Block ciphers that are easier to mask: How far can we go? In: Bertoni, G., Coron, J.-S. (eds.) CHES 2013. LNCS, vol. 8086, pp. 383–399. Springer, Heidelberg (2013)
12. Gilbert, H., Peyrin, T.: Super-sbox cryptanalysis: Improved attacks for AES-like permutations. In: Hong, S., Iwata, T. (eds.) FSE 2010. LNCS, vol. 6147, pp. 365–383. Springer, Heidelberg (2010)
13. Guo, J., Peyrin, T., Poschmann, A., Robshaw, M.: The LED block cipher. In: Preneel, B., Takagi, T. (eds.) CHES 2011. LNCS, vol. 6917, pp. 326–341. Springer, Heidelberg (2011)
14. Isobe, T., Shibutani, K.: Security analysis of the lightweight block ciphers XTEA, LED and piccolo. In: Susilo, W., Mu, Y., Seberry, J. (eds.) ACISP 2012. LNCS, vol. 7372, pp. 71–86. Springer, Heidelberg (2012)
15. Knellwolf, S., Khovratovich, D.: New preimage attacks against reduced SHA-1. In: Safavi-Naini, R., Canetti, R. (eds.) CRYPTO 2012. LNCS, vol. 7417, pp. 367–383. Springer, Heidelberg (2012)
16. Lamberger, M., Mendel, F., Rechberger, C., Rijmen, V., Schläffer, M.: Rebound distinguishers: Results on the full whirlpool compression function. In: Matsui, M. (ed.) ASIACRYPT 2009. LNCS, vol. 5912, pp. 126–143. Springer, Heidelberg (2009)
17. Mendel, F., Rijmen, V., Toz, D., Varici, K.: Differential analysis of the LED block cipher. In: Wang, X., Sako, K. (eds.) ASIACRYPT 2012. LNCS, vol. 7658, pp. 190–207. Springer, Heidelberg (2012). [21]
18. Merkle, R.C., Hellman, M.E.: On the security of multiple encryption. Commun. ACM **24**(7), 465–467 (1981)
19. Sasaki, Y.: Meet-in-the-middle preimage attacks onAES hashing modes and an application to whirlpool. In: Joux, A. (ed.) FSE 2011. LNCS, vol. 6733, pp. 378–396. Springer, Heidelberg (2011)
20. Sasaki, Y., Aoki, K.: Preimage attacks on step-reduced MD5. In: Mu, Y., Susilo, W., Seberry, J. (eds.) ACISP 2008. LNCS, vol. 5107, pp. 282–296. Springer, Heidelberg (2008)
21. Wang, X., Sako, K. (eds.): ASIACRYPT 2012. LNCS, vol. 7658. Springer, Heidelberg (2012)
22. Wei, L., Rechberger, C., Guo, J., Hongjun, W., Wang, H., Ling, S.: Improved Meet-in-the-Middle Cryptanalysis of KTANTAN. IACR Cryptology ePrint Archive (2011)

A The Auxiliary Preprocessing Algorithm of the Related-Key Attack on 3-Step LED-64

Our goal in the auxiliary preprocessing algorithm is to find (about) 2^{31} pairs $(\hat{Y}_4^{j,1}, \hat{Y}_4^{j,2} = \hat{Y}_4^{j,1} \oplus \Delta)$, such that $(Y_8^{j,1} \oplus Y_8^{j,2})_{\backslash 0,1 \backslash} = \Delta_{\backslash 0,1 \backslash}$. In order to run in time of about 2^{48} evaluations of 1-step LED, we assume that $\Delta_{\backslash 0,1,2 \backslash} = 0$, and the value of $\Delta_{\backslash 3 \backslash}$ is arbitrary (but non-zero).

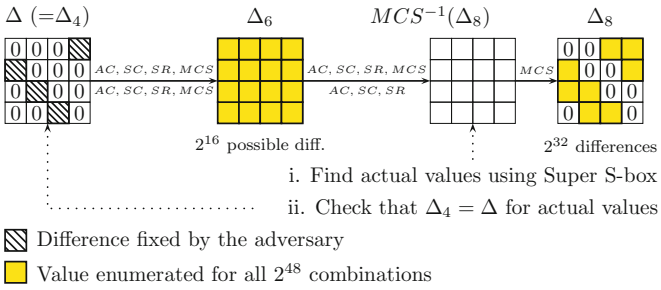


Fig. 8. The auxilary algorithm used in preprocessing (depicting differences)

Let $\Delta_r = Y_r^{j,1} \oplus Y_r^{j,2}$. The difference Δ_8 can obtain $2^{32} - 1$ non-zero values (since we require that $\Delta_{8,\backslash 0,1 \backslash} = (Y_8^{j,1} \oplus Y_8^{j,2})_{\backslash 0,1 \backslash} = \Delta_{\backslash 0,1 \backslash} = 0$), and in addition $MCS^{-1}(\Delta_6)$ can obtain (at most) $2^{16} - 1$ non-zero values, implying that Δ_6 can obtain $2^{16} - 1$ non-zero values. The algorithm is given below.

1. For each of the possible $2^{16} - 1$ non-zero value of Δ_6:
   (a) For each non-zero value of Δ_8 such that $\Delta_{8,\backslash 0,1 \backslash} = 0$:
      i. Calculate $MCS^{-1}(\Delta_8)$. Given Δ_6, use the difference distribution tables for the LED Super-Sboxes to obtain the actual values $Y_8^{j,1}, Y_8^{j,2}$.
      ii. Compute $\hat{Y}_4^{j,1} = F_2^{-1}(Y_8^{j,1})$ and $\hat{Y}_4^{j,2} = F_2^{-1}(Y_8^{j,2})$, and if $(\hat{Y}_4^{j,1} \oplus \hat{Y}_4^{j,2})_{\backslash 3 \backslash} = \Delta_{\backslash 3 \backslash}$, store the pair.

Since we obtain an average of one value for $Y_8^{j,1}$ and $Y_8^{j,2}$ in Step 1.(a).i (when considering ordered pairs), the expected time complexity of the algorithm is 2^{48}. The condition of Step 1.(a).ii holds for about 2^{-16} of the pairs, and thus we expect to return 2^{32} ordered pairs, or 2^{31} unordered pairs as claimed. We note that it is possible to implement the algorithm such that it always returns (at least) 2^{31} pairs. This can be achieved by storing all the 2^{48} pairs in Step 1.(a).ii, and finally setting the value of $\Delta_{\backslash 3 \backslash}$ to a value for which there is a maximal number of pairs (Fig. 8).

Differential-Linear Cryptanalysis Revisited

Céline Blondeau[1(✉)], Gregor Leander[2], and Kaisa Nyberg[1]

[1] Department of Information and Computer Science,
Aalto University School of Science, Espoo, Finland
{celine.blondeau,kaisa.nyberg}@aalto.fi
[2] Faculty of Electrical Engineering and Information Technology,
Ruhr Universität Bochum, Bochum, Germany
gregor.leander@rub.de

Abstract. Block ciphers are arguably the most widely used type of cryptographic primitives. We are not able to assess the security of a block cipher as such, but only its security against known attacks. The two main classes of attacks are linear and differential attacks and their variants. While a fundamental link between differential and linear cryptanalysis was already given in 1994 by Chabaud and Vaudenay, these attacks have been studied independently. Only recently, in 2013, Blondeau and Nyberg used the link to compute the probability of a differential given the correlations of many linear approximations. On the cryptanalytical side, differential and linear attacks have been applied on different parts of the cipher and then combined to one distinguisher over the cipher. This method is known since 1994 when Langford and Hellman presented the first differential-linear cryptanalysis of the DES. In this paper we take the natural step and apply the theoretical link between linear and differential cryptanalysis to differential-linear cryptanalysis to develop a concise theory of this method. We give an exact expression of the bias of a differential-linear approximation in a closed form under the sole assumption that the two parts of the cipher are independent. We also show how, under a clear assumption, to approximate the bias efficiently, and perform experiments on it. In this sense, by stating minimal assumptions, we hereby complement and unify the previous approaches proposed by Biham et al. in 2002-2003, Liu et al. in 2009, and Lu in 2012, to the study of the method of differential-linear cryptanalysis.

Keywords: Block cipher · Differential cryptanalysis · Linear cryptanalysis · Truncated differential · Multidimensional linear approximation · Bias of differential-linear approximation

1 Introduction

We are facing a fundamental change with respect to computing and information technologies. For a few years the computing world has begun to move towards the "many computers – one user" paradigm, in which the computing devices are often every day devices – a situation frequently referred to as the Internet of

C. Cid and C. Rechberger (Eds.): FSE 2014, LNCS 8540, pp. 411–430, 2015.
DOI: 10.1007/978-3-662-46706-0_21

Things (IoT). At the same time, security has become an increasingly important issue for many IoT applications as more and more sensitive personal data is transferred in a wireless manner. This implies that the use of cryptography primitives in daily life plays an increasingly crucial role. Among the different primitives, block ciphers are arguably the most widely used ones.

Great progress has been made in designing and analyzing block ciphers, especially with the introduction of the AES, but also more recently with many block ciphers appearing in the area of lightweight cryptography. However, there is still research on fundamental aspects of these ciphers going on and important questions are still not understood. For instance we are not able to assess the security of a block cipher as such, but only its security against known attacks. The two main classes to be considered here are linear and differential attacks and their variants.

Differential Cryptanalysis. The first type of attacks that is applicable to a large set of block ciphers is the differential attack introduced by Biham and Shamir in [8]. Since its invention in the early nineties several variants, tweaks and generalizations have been discussed. In 1994, Knudsen introduced so-called *truncated differentials attacks* [25]. This relaxation of classical differential attacks has since then been applied to many (round-reduced) block ciphers. In the same paper, Knudsen furthermore introduced the concept of higher-order differentials, an attack vector based on initial consideration by Lai in [27]. Another variant of differential cryptanalysis (again by Knudsen) is *impossible differentials* cryptanalysis which uses differentials with probability zero. This concept, introduced in [26] has later been successfully applied numerously, e.g. to (almost) break the cipher Skipjack [3][1]. In 1999, Wagner introduced the boomerang attack, which allows to connect two differentials over parts of a cipher that do not coincide in the middle. This attack allowed, among others, to break the cipher COCONUT98 [39]. Later, the boomerang attack itself has been generalized to amplified boomerang attack [24] and rectangle attack [4].

Linear Cryptanalysis. The second general applicable attack on block ciphers is the Matsui's linear attack [34]. Similarly to differential attacks, since its introduction many extensions and improvements have been made, and we mention a selection here. A more precise estimate for the success probability and the data complexity are given by Selçuk [38]. The effect of using more than one linear trail, referred to as linear hulls, has been introduced by Nyberg [37]; see also Daemen and Rijmen [21]. Multidimensional linear attacks have been studied by Hermelin, Cho, and Nyberg [23] as a way to further reduce the data complexity of the basic attack. These approaches have been used for example by Cho [20]. More recently, the zero-correlation attacks introduced by Bogdanov et al. in [15] have become popular. These attacks, which can be seen as the natural counterpart of the impossible differential attacks, are based on linear approximations

[1] The term *impossible differential* appeared first in [3].

with probability exactly 1/2. A further generalization of zero-correlation attacks, namely attacks based on key-invariant biases, was presented in [13].

Theoretical Links Between Linear and Differential Cryptanalysis. Most of the work has been done independently for linear and differential cryptanalysis and there are examples of ciphers that are more resistant against one type than against the other. However, the concepts are closely related. A first fundamental link between them was already given in 1994 by Chabaud and Vaudenay (see [19]), where it was shown that the probability of a differential can be expressed in terms of a sum of correlations of linear approximations. Interestingly, this link was for a long time not used in practice due to its large computational complexity. Only in 2013, Blondeau and Nyberg used the link in [11] to compute the probability of a differential given the correlations of many linear approximations. As a second result [12], Blondeau and Nyberg generalized the link to the case of multidimensional linear distinguishers and truncated differential distinguishers.

Differential-Linear Cryptanalysis. On the cryptanalytical side, differential and linear attacks have been used jointly for the first time by Langford and Hellman [30]. The basic idea of *differential-linear cryptanalysis* is to split the cipher under consideration into two parts. The split should be such that, for the first part of the cipher there exists a strong truncated differential and for the second part there exists a strongly biased linear approximation. In [30], the particular case where the differential over the first part holds with probability one has been introduced. Later on, Biham et al. [5,29] generalized this attack using a probabilistic truncated differential on the first rounds of the distinguisher.

More recently in 2012 [33], Lu studied the validity of the model proposed by Biham et al. with the aim of minimizing the assumptions needed for the validity of the attack.

Wagner presented ideas towards a unified view of statistical block cipher cryptanalysis [40]. While concentrating on structural similarities between different attacks in a Markov setting he relied, albeit with some doubts, on the previously made heuristic assumptions under which the differential-linear attacks had been claimed to work.

It is very remarkable that in none of the previous work on differential-linear cryptanalysis, the theoretical link presented in [19] between linear and differential attacks is used to model –and understand better– the general behavior of differential-linear cryptanalysis.

Our Contribution. In this paper we take the natural step and apply the theoretical link between linear and differential cryptanalysis to differential-linear cryptanalysis. This, not surprisingly, has a couple of nice consequences.

To the best of our knowledge, we are, for the first time, able to exactly express the bias of a differential-linear approximation by a closed expression. The formula is exact under the sole assumption that the two parts of the cipher are independent. In particular it is exact when averaging over all round-keys.

While evaluating this exact expression is (for full-scale ciphers) computational unfeasible, the formulation given in Theorem 2 allows, under clear assumption, to approximate the bias efficiently. In this sense we hereby complement the work of Lu by stating minimal assumptions.

Moreover, given this exact expression and –along with this– a deeper understanding of differential-linear attacks allows us to substantially generalize the attack vector. In particular, we study the possibility to take into consideration the hull of a differential-linear approximation and introduce a multidimensional generalization of differential-linear cryptanalysis which is defined for multiple input differences and multidimensional linear output masks.

Note that, we do not propose new concrete attacks. But rather we provide a sound framework for previous and future work on differential-linear cryptanalysis.

Organization of the Paper. In Sect. 2, we fix our notations and state several general results on differential and linear cryptanalysis. The related work is resumed in Sect. 3. In Sect. 4, we develop the exact expression for the bias of the differential-linear distinguisher (cf. Theorem 2) and outline its meaning with an example using the block cipher Serpent. Furthermore, we elaborate more on the comparison with previous work. In Sect. 5, we derive conditions on how and if it is possible to obtain good and practical estimations of the exact expression. We back-up our assumption with experiments using small scale variants of the cipher PRESENT. Finally, in Sect. 6, we generalize the concept of differential-linear cryptanalysis to the case of multiple differentials and multiple linear approximations and derive expressions for the biases and the attack complexities for this generalization. Sect. 7 concludes the paper.

2 Basics of Linear and Differential Cryptanalysis

2.1 Linear Correlation and Differential Probability

In differential cryptanalysis [8], the attacker is interested in identifying and exploiting non-uniformity in occurrences of plaintext and ciphertext differences. Given a vectorial Boolean function $F : \mathbb{F}_2^n \to \mathbb{F}_2^n$, a differential is given by a pair (δ, Δ) of an input difference $\delta \in \mathbb{F}_2^n$ and an output difference $\Delta \in \mathbb{F}_2^n$ and its probability is defined as

$$\mathbf{P}[\delta \xrightarrow{F} \Delta] = 2^{-n} \#\{x \in \mathbb{F}_2^n \mid F(x) + F(x+\delta) = \Delta\}.$$

Linear cryptanalysis [34] uses a linear relation between bits from plaintexts, corresponding ciphertexts and encryption key. A linear relation of bits of data $x \in \mathbb{F}_2^n$ is determined by a mask $a \in \mathbb{F}_2^n$ and is given as a Boolean function $f(x) = a \cdot x$ where "$\cdot$" is the natural inner product of the vectors a and x in $\mathbb{F}_2^n$. The strength of a linear relation is measured by its correlation.

The correlation of a Boolean function $f : \mathbb{F}_2^n \to \mathbb{F}_2$ is defined as

$$\mathbf{cor}(f) = \mathbf{cor}(f(x)) = 2^{-n}\Big[\#\{x \in \mathbb{F}_2^n | f(x) = 0\} - \#\{x \in \mathbb{F}_2^n | f(x) = 1\}\Big],$$

where the quantity within brackets correspond to the Fourier coefficient of f at zero, and can be computed using the Walsh transform of f, see e.g. [18].

In this paper, a block cipher or a part of it with a fixed key and block size n is considered as a bijective vector-valued Boolean function $F : \mathbb{F}_2^n \to \mathbb{F}_2^n$. In the general model of differential-linear cryptanalysis to be built in this paper, we consider a set of input differences to the cipher which form a linear subspace of $\mathbb{F}_2^n$. Given a subspace U of $\mathbb{F}_2^n$, let us denote by $U^\perp$ the orthogonal subspace of U with respect to the inner product of $\mathbb{F}_2^n$. Then

$$U^\perp = \{v \in \mathbb{F}_2^n \mid u \cdot v = 0, \text{ for all } u \in U\}.$$

Let us denote by $0_\ell \in \mathbb{F}_2^\ell$ the all-zero string of length ℓ. If $U = \mathbb{F}_2^s \times \{0_t\}$, for some positive integers s and t, where $s + t = n$, then $U^\perp = \{0_s\} \times \mathbb{F}_2^t$. In this manner we obtain a splitting of $\mathbb{F}_2^n$ to two mutually orthogonal subspaces, whose intersection is $\{0_n\}$. Another type of example of orthogonal subspaces is obtained for $U = \{(0,0),(1,1)\} \times \{0_{n-2}\}$. Then $U^\perp = \{(0,0),(1,1)\} \times \mathbb{F}_2^{n-2}$, in which case $U \subset U^\perp$. In any case, the dimensions of U and $U^\perp$ add up to n.

A truncated differential [25] over a vectorial Boolean function $F : \mathbb{F}_2^n \to \mathbb{F}_2^n$ is a set of ordinary differentials (δ, Δ) where the input differences $\delta \in U^\perp$ and the output differences $\Delta \in V^\perp$. In this paper we assume that U and V are linear subspaces of $\mathbb{F}_2^n$. In this manner, the truncated differential is determined by a pair of linear spaces U and V. The strength of a truncated differential is often measured by the number of solutions $(x, \delta, \Delta) \in \mathbb{F}_2^n \times (U^\perp \setminus \{0\}) \times V^\perp$ to the equation

$$F(x + \delta) + F(x) = \Delta. \tag{1}$$

To facilitate the derivations in this paper we will use a different but closely related quantity, which allows the zero difference in the input. It is straightforward to show that, the number of solutions $(x, \delta, \Delta) \in \mathbb{F}_2^n \times U^\perp \times V^\perp$ of (1) can be computed as

$$\sum_{\delta \in U^\perp, \Delta \in V^\perp} \#\{x \in \mathbb{F}_2^n \mid F(x + \delta) + F(x) = \Delta\}.$$

We denote by $\mathbf{P}[U^\perp \xrightarrow{F} V^\perp]$ the probability that a pair of inputs $(x, x+\delta)$, where x is picked uniformly at random in $\mathbb{F}_2^n$ and $\delta \in U^\perp$, gives an output difference $\Delta \in V^\perp$.

Proposition 1. *Let U and V be linear subspaces of $\mathbb{F}_2^n$, we have*

$$\begin{aligned}\mathbf{P}[U^\perp \xrightarrow{F} V^\perp] &= \frac{1}{2^n |U^\perp|} \#\{(x, \delta, \Delta) \in \mathbb{F}_2^n \times U^\perp \times V^\perp \mid F(x + \delta) + F(x) = \Delta\} \\ &= \frac{1}{|U^\perp|} \sum_{\delta \in U^\perp,\, \Delta \in V^\perp} \mathbf{P}[\delta \xrightarrow{F} \Delta]. \end{aligned} \tag{2}$$

The probability $\mathbf{P}[U^\perp \xrightarrow{F} V^\perp]$ which can be expressed in the two different ways shown in Proposition 1 will be called the truncated differential probability.

Let us denote by $\mathbf{P}[U^\perp \setminus \{0\} \xrightarrow{F} V^\perp]$ the probability for the truncated differential derived analogically as above but without allowing the zero input difference. Then we have the following relation:

$$|U^\perp| \cdot \mathbf{P}[U^\perp \xrightarrow{F} V^\perp] = 1 + (|U^\perp| - 1) \cdot \mathbf{P}[U^\perp \setminus \{0\} \xrightarrow{F} V^\perp]. \tag{3}$$

In particular, for the ordinary differential probability, we have

$$\mathbf{P}[\delta \xrightarrow{F} \Delta] = 2 \cdot \mathbf{P}[sp(\delta) \xrightarrow{F} \Delta] - 1,$$

for all $\delta, \Delta \in \mathbb{F}_2^n$, $\delta \neq 0$. Here, as well as later in the paper, we use the notation $sp(a)$ to denote the vector subspace $\{0, a\} \subset \mathbb{F}_2^n$ spanned by a.

Recalling the symmetry of the probability of single differential for a bijective function F

$$\mathbf{P}[\delta \xrightarrow{F} \Delta] = \mathbf{P}[\Delta \xrightarrow{F^{-1}} \delta],$$

let us note that the truncated differential probability is not symmetric, except in the case when $|U| = |V|$. In general, we have

$$|U^\perp| \cdot \mathbf{P}[U^\perp \xrightarrow{F} V^\perp] = |V^\perp| \cdot \mathbf{P}[V^\perp \xrightarrow{F^{-1}} U^\perp].$$

Let us recall the link between the differential probabilities and the squared correlations of linear approximations of vectorial Boolean functions presented by Chabaud and Vaudenay [19]. In the context of this paper we write it as follows.

$$\mathbf{P}[\delta \xrightarrow{F} \Delta] = 2^{-n} \sum_{u \in \mathbb{F}_2^n} \sum_{v \in \mathbb{F}_2^n} (-1)^{u \cdot \delta + v \cdot \Delta} \mathbf{cor}^2 \left(u \cdot x + v \cdot F(x)\right), \tag{4}$$

where $F : \mathbb{F}_2^n \to \mathbb{F}_2^n$ is a vectorial Boolean function, and $(\delta, \Delta) \in \mathbb{F}_2^n \times \mathbb{F}_2^n$. By applying this link for all $\delta \in U^\perp$ and $\Delta \in V^\perp$ in (2) we obtain the following result which is a generalization of [11,12].

Theorem 1. *The probability of a truncated differential with input differences in $U^\perp$ and output differences in $V^\perp$ can be computed as a sum of squared correlations with input masks in U and output masks in V as*

$$\mathbf{P}[U^\perp \xrightarrow{F} V^\perp] = \frac{1}{|V|} \sum_{u \in U, v \in V} \mathbf{cor}^2 \left(u \cdot x + v \cdot F(x)\right).$$

Proof. If for $u \in \mathbb{F}_2^n$ we have $u \cdot \delta = 1$ for some $\delta \in U^\perp$ then the linear function $\delta \mapsto u \cdot \delta$ is non-zero, and hence balanced on $U^\perp$. Thus in this case $\sum_{\delta \in U^\perp} (-1)^{u \cdot \delta} = 0$. This is not the case exactly if we have $u \in U$, and then $\sum_{\delta \in U^\perp} (-1)^{u \cdot \delta} = |U^\perp|$. Then, applying the same reasoning for all $v \in \mathbb{F}_2^n$ gives the claim. □

Corollary 1. *For all* $w \in \mathbb{F}_2^n \setminus \{0\}$ *and* $\Delta \in \mathbb{F}_2^n \setminus \{0\}$ *we have*

$$\mathbf{P}[\Delta \xrightarrow{F} sp(w)^\perp] = \sum_{v \in sp(\Delta)^\perp} \mathbf{cor}^2(v \cdot x + w \cdot F(x)).$$

Proof. From (3) and Theorem 1, we have

$$\begin{aligned}
&\mathbf{P}[\Delta \xrightarrow{F} sp(w)^\perp] = 2 \cdot \mathbf{P}[sp(\Delta) \xrightarrow{F} sp(w)^\perp] - 1 \\
&= 2 \cdot \frac{1}{2} \cdot \sum_{v \in sp(\Delta)^\perp, b \in sp(w)} \mathbf{cor}^2(v \cdot x + b \cdot F(x)) - 1 \\
&= \sum_{v \in sp(\Delta)^\perp} \mathbf{cor}^2(v \cdot x + w \cdot F(x)).
\end{aligned}$$

□

2.2 Round Independence

Computation of differential probabilities or linear correlations over an iterated cipher is often done assuming that the rounds of the cipher behave independently.

Definition 1. Two parts E_0 and E_1 of an n-bit block cipher $E = E_1 \circ E_0$ are said to be *differentially round independent* if for all $(\delta, \Omega) \in \mathbb{F}_2^n \times \mathbb{F}_2^n$ the following holds

$$\mathbf{P}[\delta \xrightarrow{E} \Omega] = \sum_{\Delta \in \mathbb{F}_2^n} \mathbf{P}[\delta \xrightarrow{E_0} \Delta] \mathbf{P}[\Delta \xrightarrow{E_1} \Omega].$$

Analogically, the parts E_0 and E_1 are said to be *linearly round independent* if for all $(u, w) \in \mathbb{F}_2^n \times \mathbb{F}_2^n$ the following holds

$$\mathbf{cor}^2(u \cdot x + w \cdot E(x)) = \sum_{v \in \mathbb{F}_2^n} \mathbf{cor}^2(u \cdot x + v \cdot E_0(x)) \mathbf{cor}^2(v \cdot y + w \cdot E_1(y)).$$

It was proved in [2] that the rounds of a Markov cipher [28] are both differentially and linearly round independent. Next we show that differential and linear round independence are equivalent concepts for any cipher.

Proposition 2. *Two parts* E_0 *and* E_1 *of an* n*-bit block cipher* $E = E_1 \circ E_0$ *are differentially round independent if and only if they are linearly round independent.*

Proof. Let us start by stating (4) in the following equivalent form

$$\sum_{\delta \in \mathbb{F}_2^n} (-1)^{u \cdot \delta} \mathbf{P}[\delta \xrightarrow{F} \Delta] = \sum_{v \in \mathbb{F}_2^n} (-1)^{v \cdot \Delta} \mathbf{cor}^2 \left(u \cdot x + v \cdot F(x)\right).$$

This is obtained by applying the inverse Fourier transform to the input difference. By applying it to the output difference, another equivalent form can be given

where the first summation is taken over Δ and the second summation over u. We refer to these equations as partial inverses of (4). A further variant is obtained by applying the inverse Fourier transform on both differences. We call it the inverse of (4).

Let us now assume that the parts of the cipher are differentially round independent. Then using the inverse of (4) and the assumption of differential round independence, we get

$$\begin{aligned}\mathbf{cor}^2\left(u\cdot x+w\cdot E(x)\right) &= 2^{-n}\sum_{\delta\in\mathbb{F}_2^n}\sum_{\Omega\in\mathbb{F}_2^n}(-1)^{u\cdot\delta+w\cdot\Omega}\mathbf{P}[\delta\xrightarrow{E}\Omega]\\ &= 2^{-n}\sum_{\Delta\in\mathbb{F}_2^n}\sum_{\delta\in\mathbb{F}_2^n}(-1)^{u\cdot\delta}\mathbf{P}[\delta\xrightarrow{E_0}\Delta]\sum_{\Omega\in\mathbb{F}_2^n}(-1)^{w\cdot\Omega}\mathbf{P}[\Delta\xrightarrow{E_1}\Omega].\end{aligned}$$

Then using the both partial inverses of (4) we obtain

$$\begin{aligned}&\mathbf{cor}^2\left(u\cdot x+w\cdot E(x)\right)\\ &= 2^{-n}\sum_{\Delta\in\mathbb{F}_2^n}\sum_{v\in\mathbb{F}_2^n}(-1)^{v\cdot\Delta}\mathbf{cor}^2(u\cdot x+v\cdot E_0(x))\sum_{v'\in\mathbb{F}_2^n}(-1)^{v'\cdot\Delta}\mathbf{cor}^2\left(v'\cdot y+w\cdot E_1(y)\right)\\ &= 2^{-n}\sum_{v\in\mathbb{F}_2^n}\sum_{v'\in\mathbb{F}_2^n}\mathbf{cor}^2(u\cdot x+v\cdot E_0(x))\,\mathbf{cor}^2\left(v'\cdot y+w\cdot E_1(y)\right)\sum_{\Delta\in\mathbb{F}_2^n}(-1)^{(v+v')\cdot\Delta}.\end{aligned}$$

The sum over Δ is non-zero if and only if $v=v'$ and the value of this sum, 2^n, cancels with the factor 2^{-n}. We can then see that the condition of linear round independence is satisfied. The converse proof is analogous. □

Few ciphers satisfy round independence in the strict sense of Definition 1. On the other hand, n-bit ciphers of the form $E_K(x)=E_1(E_0(x)+K)$ with n-bit key K are round independent on average over the key. For simplicity, the results given in this paper will be stated in terms of strict round independence, but can be reformulated using average round independence for such ciphers.

3 Previous Work

Let $E:\mathbb{F}_2^n\to\mathbb{F}_2^n$ be a cipher. When applying the technique of differential-linear cryptanalysis the iterated block cipher is presented as a composition $E=E_1\circ E_0$ of two parts. The first part E_0 is chosen in such a way that there is some strong truncated differential over E_0. Let U and V be the subspaces that define the truncated differential. Typically, the input difference space U is selected so that $U^\perp$ is one-dimensional. The output difference space $V^\perp$ is usually larger. It is then assumed that there is a strong linear approximation (v,w) over E_1, where $v\in V$, which means that $v\cdot\Delta=0$ for all $\Delta\in V^\perp$.

In this section, we assume that the input-difference space $U^\perp$ is one-dimensional. Let δ be the sole non-zero element in $U^\perp$. Then the bias of the differential-linear approximation is defined as

$$\mathcal{E}_{\delta,w}:=\mathbf{P}[w\cdot(E(x+\delta)+E(x))=0]-\frac{1}{2}.$$

In the previous treatments [5,30,33], $\mathcal{E}_{\delta,w}$ is evaluated using the Piling-up lemma [34] by decomposing the Boolean variable $w\cdot(E(x+\delta)+E(x))$ as a sum of three variables

$$\begin{aligned} w\cdot(E(x+\delta)+E(x)) &= v\cdot E_0(x+\delta)+w\cdot E(x+\delta)\\ &\quad + v\cdot(E_0(x+\delta)+E_0(x))\\ &\quad + v\cdot E_0(x)+w\cdot E(x), \end{aligned} \tag{5}$$

which are assumed to be independent as x varies.

By using the following notation for the involved biases

$$\begin{aligned} \epsilon_{v,w} &= \mathbf{P}[v\cdot y + w\cdot E_1(y)=0]-\frac{1}{2}\\ \varepsilon_{\delta,v} &= \mathbf{P}[v\cdot(E_0(x+\delta)+E_0(x))=0]-\frac{1}{2} = \mathbf{P}[\delta \overset{E_0}{\to} sp(v)^{\perp}]-\frac{1}{2}, \end{aligned} \tag{6}$$

the Piling-up lemma gives

$$\mathcal{E}_{\delta,w} = 4\varepsilon_{\delta,v}\epsilon_{v,w}^2. \tag{7}$$

It remains to determine $\varepsilon_{\delta,v}$ given the truncated differential probability $\mathbf{P}[\delta \overset{E_0}{\to} V^{\perp}]$. This is where the previous studies differ. In [30], $\mathbf{P}[\delta \overset{E_0}{\to} V^{\perp}]=1$, in which case $\mathbf{P}[\delta \overset{E_0}{\to} sp(v)^{\perp}]=1$, since $v\in V$. According to Biham et al. [7] this was generalized first in [29] and later by Biham et al. [5] to the case where $\mathbf{P}[\delta \overset{E_0}{\to} V^{\perp}]<1$. In [5], Biham et al. denote this probability by p' and by assuming that when $\Delta\notin V^{\perp}$ the parities of $v\cdot\Delta$ are balanced they obtain the estimate

$$\mathbf{P}[\delta \overset{E_0}{\to} sp(v)^{\perp}] \approx p' + (1-p')\frac{1}{2}.$$

This exact equality holds if $p'=1$. In general, it gives only an approximation, for the simple reason that if a linear function $v\cdot y$ vanishes in $V^{\perp}$, it cannot be balanced outside $V^{\perp}$. The approximation is better, if $V^{\perp}$ is small, which is the case studied in [5]. This approximation becomes worse, however, as $V^{\perp}$ increases. The extreme case is $sp(v)=V$. Then $v\cdot y=1$, for all $y\notin V^{\perp}$.

This problem was observed by Lu and suggested to be solved in his study [33] by restricting to the case where the output difference space of the truncated differential is the hyperplane $sp(v)^{\perp}$.

As in practice, $V^{\perp}$ is often smaller than a zero space of a linear Boolean function, we have that $\mathbf{P}[\delta \overset{E_0}{\to} V^{\perp}]$ is less than or equal to $\mathbf{P}[\delta \overset{E_0}{\to} sp(v)^{\perp}]$. It can also be strictly less, in which case replacing the latter by the former in the estimation of the bias (6) may lead to a wrong result for $\mathcal{E}_{\delta,w}$. Biham et al. suggest that the other output differences $\Delta\in sp(v)\setminus V^{\perp}$ may occur with high probability and affect their approximation and stress the importance to do experimental verification.

Note that it would be possible to fix the assumption by Biham et al. by correcting the probability of zero parity outside $V^{\perp}$ to $(2^{n-1}-|V^{\perp}|)/(2^n-|V^{\perp}|)$.

In [32], the authors mention the possibility of using multiple linear approximations in order to improve the complexity of a differential-linear distinguisher. Their study, which is based on the differential-linear model of Biham et al. [6] and on the multiple linear model of Biryukov et al. [9], assume that the distinguisher is built from the combination of only one truncated differential with independent linear approximations.

The goal of this paper is to analyze in more detail what is happening in the intermediate layer of the differential-linear approximation and take into account not only more high-probability output differences from E_0 but also more, not necessarily independent, linear approximations over E_1. Still, many differences and linear masks in the intermediate layer must be left out. To handle them in Theorem 3, we make an assumption analogical to the one of Biham et al. but corrected.

4 Differential-Linear Hull

The basic tool in examining the intermediate layer between E_0 and E_1 is the following theorem. We use the notation $\mathcal{E}_{\delta,w}$ and $\varepsilon_{\delta,v}$ introduced in the preceding section, and denote the correlation of the linear approximation $v \cdot y + w \cdot E_1(y)$ by $c_{v,w}$. Then $c_{v,w} = 2\epsilon_{v,w}$ in relation to the notation used in the preceding section.

Theorem 2. *Assume that the parts E_0 and E_1 of the block cipher $E = E_1 \circ E_0$ are independent. Using the notation previously defined, for all $\delta \in \mathbb{F}_2^n \setminus \{0\}$ and $w \in \mathbb{F}_2^n \setminus \{0\}$, we have*

$$\mathcal{E}_{\delta,w} = \sum_{v \in \mathbb{F}_2^n} \varepsilon_{\delta,v} c_{v,w}^2. \tag{8}$$

Proof. First, we apply the assumption of independence to the probability $\mathbf{P}[\delta \xrightarrow{E} sp(w)^\perp]$ and then, the link given by Corollary 1 to the differential probability over E_1.

$$\begin{aligned}
\mathbf{P}[\delta \xrightarrow{E} sp(w)^\perp] &= \sum_{\Delta \in \mathbb{F}_2^n} \mathbf{P}[\delta \xrightarrow{E_0} \Delta] \mathbf{P}[\Delta \xrightarrow{E_1} sp(w)^\perp] \\
&= \sum_{\Delta \in \mathbb{F}_2^n} \mathbf{P}[\delta \xrightarrow{E_0} \Delta] \sum_{v \in sp(\Delta)^\perp} \mathbf{cor}^2(v \cdot y + w \cdot E_1(y)) \\
&= \sum_{v \in \mathbb{F}_2^n} \sum_{\Delta \in sp(v)^\perp} \mathbf{P}[\delta \xrightarrow{E_0} \Delta] \mathbf{cor}^2(v \cdot y + w \cdot E_1(y)) \\
&= \sum_{v \in \mathbb{F}_2^n} \mathbf{P}[\delta \xrightarrow{E_0} sp(v)^\perp] \mathbf{cor}^2(v \cdot y + w \cdot E_1(y)),
\end{aligned}$$

where changing the order of summation is possible since

$$\{(v, \Delta) \,|\, \Delta \in \mathbb{F}_2^n,\, v \in sp(\Delta)^\perp\} = \{(v, \Delta) \,|\, v \in \mathbb{F}_2^n,\, \Delta \in sp(v)^\perp\}.$$

Now by subtracting $\frac{1}{2}$ from both of the sides of the obtained equality and using Parseval's theorem gives the result. □

We call the expression (8) the differential-linear hull of $E = E_1 \circ E_0$. The differential-linear method has been previously applied in cases, where only one correlation $c_{v,w}$ has been identified to have a large absolute value but the output differential space of the truncated differential is smaller than the zero space of v. Consequently more than one trail must be taken into account when estimating the bias of the differential-linear approximation. We illustrate this in the context of an attack on the Serpent cipher [1].

Example on Serpent. Differential-linear cryptanalysis [6,22] which has been applied to many ciphers, remains with the multidimensional linear cryptanalysis [35,36] the most powerful attack on the Serpent cipher [1]. In this section, we summarize, in our notation, the distinguisher proposed in [6], on 9 rounds of Serpent.

To be useful in a key-recovery attack, the distinguisher was defined as starting from the second round of the cipher. First a truncated differential is defined on 3 rounds of Serpent. In this attack, only one input difference is taken into consideration meaning that $U^\perp$ is one-dimensional. The output space of the truncated differential consists of all differences which have the bits number 1 and 117 equal to zero. Hence it is the orthogonal of the two-dimensional space V spanned by the bits (taken as basis vectors) number 1 and 117. The truncated differential probability $\mathbf{P}[\delta \xrightarrow{E_0} V^\perp]$ being large, it can, as typically in differential-linear cryptanalysis, be computed experimentally and was evaluated in [6] to $2^{-1} + 2^{-6}$. The strong linear approximation over the six following rounds has input mask $\nu \in V$ where both bits number 1 and 117 are equal to 1. The output mask is denoted by w. The correlation of this linear approximation is estimated to $c_{\nu,w} = 2^{-26}$.

The resulting differential-linear relation spans over 9 rounds of Serpent. In [6], its bias was estimated to $\varepsilon_{\delta,\nu} c_{\nu,w}^2$ with $\varepsilon_{\delta,\nu} = 2^{-7}$ to obtain

$$\mathcal{E}_{\delta,w} \approx \varepsilon_{\delta,\nu} c_{\nu,w}^2 = 2^{-7} \cdot 2^{-52}. \tag{9}$$

Later, in [22], another similar distinguisher on Serpent was provided. The only difference was that a new and stronger truncated differential over the three rounds of E_0 was used.

Our aim is to analyze the conditions under which the approximation (9) is justified. From Theorem 2 we deduce that $\mathcal{E}_{\delta,w}$ can be computed as

$$\mathcal{E}_{\delta,w} = \sum_{v \in V} \varepsilon_{\delta,v} c_{v,w}^2 + \sum_{v \in \mathbb{F}_2^n \setminus V} \varepsilon_{\delta,v} c_{v,w}^2. \tag{10}$$

We observe that for the two masks $v \in V$, for which only one bit, either number 1 or 117, is equal to 1, the correlations $c_{v,w}$ are equal to zero. Then it follows that the first sum on the right side of (10) is, indeed, equal to $\varepsilon_{\delta,\nu} c_{\nu,w}^2$. It remains to examine under which assumptions the sum (9) is an underestimate of the actual bias (10). This will be done in a more general setting in Sect. 5.1.

5 Intermediate Space

5.1 Estimation of the Bias

In this section, we aim to analyze whether we can obtain a good estimate of the bias of a differential-linear relation. For a better illustration, the analysis provided in this section is based on Theorem 2 where the differential-linear approximation is defined for one input difference δ and one output mask w. A generalization of this result for sets of input differences and output masks will be given in Sect. 6.

In the differential context, it is well known that the expected probability of a differential is underestimated if we are only able to collect a small number of differential characteristics relative to the differential.

As recalled in Sect. 3, in the few available analyses in the differential-linear context, the bias of the differential-linear approximation is estimated as the combination of one strong truncated differential with one strong linear approximation. In this section, we discuss the possibility of generalizing this result to obtain a better estimate of the bias $\mathcal{E}_{\delta,w}$ by using the hull of a differential-linear approximation more efficiently. From Theorem 2, we know that an accurate computation of the bias of a differential-linear approximation requires the knowledge of the correlations over E_1 for all input masks $v \in \mathbb{F}_2^n$, which is impossible in practice for many ciphers.

From Theorem 2 and as given in Eq. (10) the bias of a differential-linear approximation can be decomposed into two sums with respect to a set V.

$$\mathcal{E}_{\delta,w} = \sum_{v \in V, v \neq 0} \varepsilon_{\delta,v} c_{v,w}^2 + \sum_{v \notin V} \varepsilon_{\delta,v} c_{v,w}^2.$$

Notice that the bias of a different-linear equation can be, as in the linear context, positive or negative. As the complexity of the underlying attack is independent of the sign, we talk, as in the linear context, of absolute bias $|\mathcal{E}_{\delta,w}|$.

Assumption 1. *Given a set V we assume that*

$$\Big| \sum_{v \in V, v \neq 0} \varepsilon_{\delta,v} c_{v,w}^2 \Big| \leq |\mathcal{E}_{\delta,w}|,$$

meaning that $|\sum_{v \in V} \varepsilon_{\delta,v} c_{v,w}^2|$ is an underestimate of the bias of the differential-linear approximation with input difference δ and output mask w.

The only way to check if we have an under or over estimate of the actual probability consists in experimentally computing the bias of a differential-linear approximation on a reduced number of round of the cipher. These experiments should be done in respect to the intermediate space V. In [5,6], experiments of this type where already conducted to check the validity of their results.

If the intermediate space V is large, it is infeasible to compute the biases $\varepsilon_{\delta,v}$ over E_0 or the correlations $c_{v,w}$ over E_1 for all $v \in V$. Next, based on the assumption that some probabilities over E_0 are equal, we show that $\sum_{v \in V} \varepsilon_{\delta,v} c_{v,w}^2$ can be estimated from the product of one truncated differential probability with the capacity of one multidimensional-linear approximation.

Theorem 3. *Let* $\varepsilon_{\delta,V} = \mathbf{P}[\delta \xrightarrow{E_0} V^\perp] - \dfrac{1}{|V|}$ *be the bias of a truncated differential with one non-zero input difference* δ *and output differences in* $V^\perp$*. Further, we denote by* $C_{V,w} = \sum_{v \in V, v \neq 0} c_{v,w}^2$ *the capacity of the multidimensional linear approximation with all input masks* v *in* V *and one output mask* $w \neq 0$*.*

If then, for all $\Delta \notin V^\perp$*, the probabilities* $\mathbf{P}[\delta \xrightarrow{E_0} \Delta]$ *are equal, we have*

$$\sum_{v \in V} \varepsilon_{\delta,v} c_{v,w}^2 = \frac{1}{2} \frac{|V|}{|V|-1} \varepsilon_{\delta,V} C_{V,w}. \tag{11}$$

Proof. For a purpose of clarity, let us denote $Q = \mathbf{P}[\delta \xrightarrow{E_0} V^\perp]$. We denote by p the common value of the probabilities $\mathbf{P}[\delta \xrightarrow{E_0} \Delta]$ for $\Delta \notin V^\perp$. Then by $\sum_{\Delta \in \mathbb{F}_2^n} \mathbf{P}[\delta \xrightarrow{E_0} \Delta] = 1$ we deduce that $p = \dfrac{1-Q}{2^n - |V^\perp|}$.
Since $V^\perp \subset sp(v)^\perp$ holds for all $v \in V$, we have

$$\begin{aligned}\mathbf{P}[\delta \xrightarrow{E_0} sp(v)^\perp] &= \mathbf{P}[\delta \xrightarrow{E_0} V^\perp] + \sum_{\Delta \in sp(v)^\perp,\ \Delta \notin V^\perp} \mathbf{P}[\delta \xrightarrow{E_0} \Delta] \\ &= Q + (2^{n-1} - |V^\perp|) \cdot \frac{1-Q}{2^n - |V^\perp|}.\end{aligned}$$

Therefore, for all $v \in V$, we have

$$\begin{aligned}\varepsilon_{\delta,v} &= \mathbf{P}[\delta \xrightarrow{E_0} sp(v)^\perp] - \frac{1}{2} = Q + \left(2^{n-1} - |V^\perp|\right) \frac{1-Q}{2^n - |V^\perp|} - \frac{1}{2} \\ &= \frac{1}{2} \cdot \frac{2^n Q - |V^\perp|}{2^n - |V^\perp|} = \frac{1}{2} \cdot \frac{Q - |V|^{-1}}{1 - |V|^{-1}} \\ &= \frac{1}{2} \cdot \frac{|V|}{|V|-1} \left(Q - \frac{1}{|V|}\right) = \frac{1}{2} \cdot \frac{|V|}{|V|-1} \varepsilon_{\delta,V}.\end{aligned}$$

And we deduce

$$\sum_{v \in V} \varepsilon_{\delta,v} c_{v,w} = \frac{1}{2} \frac{|V|}{|V|-1} \varepsilon_{\delta,V} \sum_{v \in V} c_{v,w}^2 = \frac{1}{2} \frac{|V|}{|V|-1} \varepsilon_{\delta,V} C_{V,w}.$$

□

Let us note that if $|V| = 2$, we have $\sum_{v \in V} \varepsilon_{\delta,v} c_{v,w}^2 = \varepsilon_{\delta,V} C_{V,w}$. The larger the size of $|V|$, the closer to $\dfrac{1}{2} \varepsilon_{\delta,V} C_{V,w}$ we are.

5.2 Experiments

The experiments of this section have been performed on a 32-bit scaled version of PRESENT [14,31] called SmallPresent-[8]. The differential-linear approximations are defined for one input difference δ and one output mask w. To limit

the number of assumptions, the bias $\varepsilon_{\delta,v}$ and the correlations $c_{v,w}$ are computed experimentally using 2^{30} plaintexts and averaged over 200 keys. When using Theorem 2, round independence is only required between E_0 and E_1.

The purpose of these experiments was to check the accuracy of Assumption 1. In each of the figures of this section, we plotted as a reference the experimental bias

$$\mathcal{E}_{\delta,w} = \mathbf{P}[\delta \to sp(w)^{\perp}] - 2^{-1}, \tag{12}$$

over 8 rounds of SmallPresent-[8] and given a space V, compare it with

$$\sum_{v \in V} \varepsilon_{\delta,v} c_{v,w}^2. \tag{13}$$

While experiments have been performed for many differential-linear approximations on 8 rounds of SmallPresent-[8], we present results for the input difference $\delta =$ `0x1` and the output mask $w =$ `0x80000000`. The bias of this differential-linear approximation is positive and we are expecting under Assumption 1 to find that (13) is an underestimate of the actual bias. In Fig. 1, resp. in Fig. 2, the differentials are taken over 3 rounds, resp. 4 rounds, and the correlations are taken over 5 rounds, resp. 4 rounds of SmallPresent-[8]. The space V is chosen to be linear.

As the accuracy of these approximations depends mostly on the size of the intermediate space, we study the evolution of (13) in regards to $\log(|V|)$.

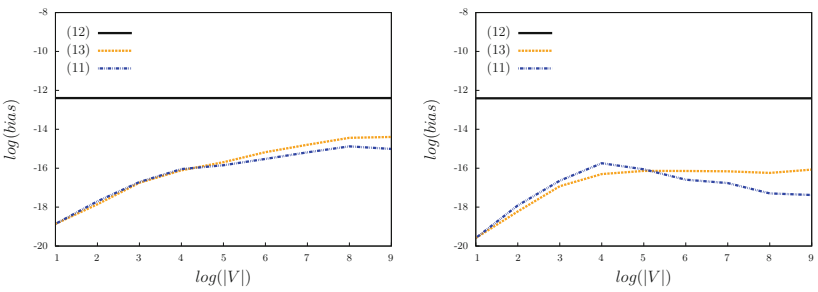


Fig. 1. Estimation of the bias a differential-linear approximation on 3+5 rounds of SmallPresent-[8] for two different chains of intermediate spaces.

Result of the different experiments show that in the case of SmallPresent-[8], (13) gives as expected an underestimate of the actual bias $\mathcal{E}_{\delta,w}$. In most of the cases by increasing the size of the intermediate space V, we have a better estimate of the bias (in this experiments, the initial spaces V are subset of the larger ones). Nevertheless as the second sum of (10) is not always positive we observe that this gain can be somewhat relative. When experiments are conducted for a fixed key instead of averaged over keys, we strictly observe that (13) is not an increasing function of $|V|$.

In Theorem 3, based on the assumption that for all $\Delta \notin V^{\perp}$, the probabilities $\mathbf{P}[\delta \xrightarrow{E_0} \Delta]$ are equal, we propose an estimate of (13). This one is relatively easier to compute since, independently of the size of V, only one truncated differential probability and one capacity need to be computed. The blue curves in Figs. 1 and 2 correspond to the computation of the expression on the right side of (11). While this expression seems to be a correct estimate of (13) for V of small size, the assumption that for all $\Delta \notin V^{\perp}$, the probabilities $\mathbf{P}[\delta \xrightarrow{E_0} \Delta]$ are equal, is getting less realistic when increasing the size of the space V.

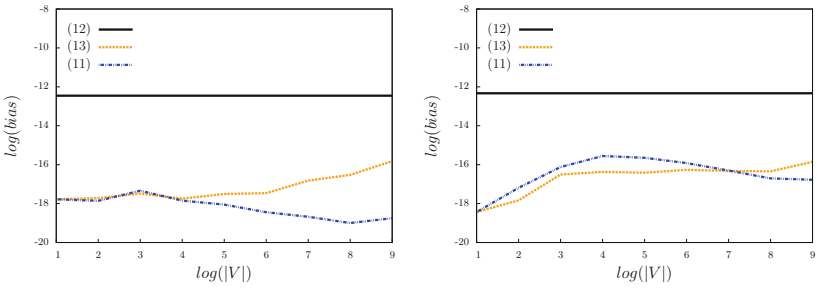


Fig. 2. Estimation of the bias of a differential-linear approximation on 4+4 rounds of SmallPresent-[8] for two different chains of intermediate spaces.

This phenomenon also appears when multiplying truncated differential probabilities over rounds of the cipher to obtain the probability of a truncated distinguisher and has been experimentally tested for instance in [17].

6 Multidimensional Differential-Linear Distinguisher

6.1 The Model

The idea of taking advantage of multiple differentials or multiple linear approximations is widely spread out in the cryptographic community. To generalize the results of Sect. 4, let us now consider the case where the space $U^{\perp}$ of possible input differences is an arbitrary subspace of $\mathbb{F}_2^n$. The linear approximation over E_1 is assumed to be multidimensional such that the output masks form a linear subspace W of $\mathbb{F}_2^n$. We denote its orthogonal space by $W^{\perp}$.

The conditions on which it would be possible to combine such a truncated differential and multidimensional linear approximation to a strong truncated differential over the full cipher are similar to the ones in the one-dimensional case expressed in Sect. 5.

We express here the generalization of Theorem 2 to compute the bias

$$\mathcal{E}_{U,W} = \mathbf{P}[U^{\perp} \setminus \{0\} \xrightarrow{E} W^{\perp}] - \frac{1}{|W|}, \tag{14}$$

of a multidimensional differential-linear approximation.

Theorem 4. *Let $\mathcal{E}_{U,W}$ as in (14). Assume that the parts E_0 and E_1 of the block cipher $E = E_1 \circ E_0$ are independent. Then*

$$\mathcal{E}_{U,W} = \frac{2}{|W|} \sum_{v \in \mathbb{F}_2^n, v \neq 0} \varepsilon_{U,v} C_{v,W}, \tag{15}$$

where $\varepsilon_{U,v} = \mathbf{P}[U^\perp \setminus \{0\} \xrightarrow{E_0} sp(v)^\perp] - 1/2$, and $C_{v,W} = \sum_{w \in W, w \neq 0} \mathbf{cor}^2(v \cdot y + w \cdot E_1(y))$, is for $v \neq 0$, the capacity of the multidimensional linear approximation with input mask v and all nonzero output masks w in W.

Proof. First, let us state the following generalization of Corollary 1. Given a bijective function $F : \mathbb{F}_2^n \to \mathbb{F}_2^n$, a subspace $U \subset \mathbb{F}_2^n$ and a mask vector $v \in \mathbb{F}_2^n$, we have

$$2\mathbf{P}[U^\perp \xrightarrow{F} sp(v)^\perp] - 1 = \sum_{u \in U} \mathbf{cor}^2(u \cdot x + v \cdot F(x)).$$

Using Theorem 1 to write the truncated differential probability in terms of squared correlations, we apply this result together with Proposition 2 to obtain

$$\begin{aligned}
\mathbf{P}[U^\perp \xrightarrow{E} W^\perp] &= \frac{1}{|W|} \sum_{u \in U, v \in \mathbb{F}_2^n, w \in W} \mathbf{cor}^2(u \cdot x + v \cdot E_0(x)) \mathbf{cor}^2(v \cdot y + w \cdot E_1(y)) \\
&= \frac{1}{|W|} \sum_{v \in \mathbb{F}_2^n} \left(2\mathbf{P}[U^\perp \xrightarrow{E_0} sp(v)^\perp] - 1 \right) \sum_{w \in W} \mathbf{cor}^2(v \cdot y + w \cdot E_1(y)).
\end{aligned}$$

The next step consists at removing the zero from the possible input differences. To use relation (3) we multiply the probabilities on the first and second line by $|U|$ and then subtract $1 = \frac{1}{|W|} \sum_{w \in W} \sum_{v \in \mathbb{F}_2^n} \mathbf{cor}^2(v \cdot y + w \cdot E_1(y))$ to get

$$\begin{aligned}
(|U| - 1)\mathbf{P}[U^\perp \setminus \{0\} \xrightarrow{E} W^\perp] &= \frac{1}{|W|} \sum_{v \in \mathbb{F}_2^n} \left(2|U|\mathbf{P}[U^\perp \xrightarrow{E_0} sp(v)^\perp] - |U| - 1 \right) C_{v,W} \\
&= \frac{1}{|W|} \sum_{v \in \mathbb{F}_2^n} \left(2(|U|\mathbf{P}[U^\perp \xrightarrow{E_0} sp(v)^\perp] - 1) - |U| + 1 \right) C_{v,W} \\
&= \frac{1}{|W|} \sum_{v \in \mathbb{F}_2^n} \left((|U| - 1)(2\mathbf{P}[U^\perp \setminus \{0\} \xrightarrow{E_0} sp(v)^\perp] - 1) \right) C_{v,W}.
\end{aligned}$$

We obtain the claim by dividing the first and last expression in this chain of equalities by $|U| - 1$ and then observing that the term for $v = 0$ in the last expression is equal to $1/|W|$. □

6.2 Complexity of a Distinguishing Attack

When the differential-linear approximation is characterized by only one output mask w, as the data complexity is inverse proportional to the square of the bias $\mathcal{E}_{\delta,w}$, larger its absolute value is, less costly the underlined distinguishing attack

is. When using multiple output masks, the differential-linear probability should be distinguishable from the uniform probability $\frac{1}{|W|}$ and the data complexity of the differential-linear distinguisher depends of the number $|W|$ of output masks. As classically done in the differential context, using multiple input differences allows the construction of structures and divides the data complexity of the distinguisher by $|U^{\perp}|$.

Proposition 3. *Using the framework of [10,16,38], the data complexity of a "multidimensional" differential-linear distinguisher with input differences in $U^{\perp}$ and output masks in W is proportional to*

$$\frac{2}{|U^{\perp}|}\frac{|W|^{-1}}{\mathcal{E}_{U,W}^2} = \frac{|W|}{2|U^{\perp}|}\frac{1}{(\sum_v \varepsilon_{U,v}C_{v,W})^2}. \tag{16}$$

When increasing the number of output masks, for each $v \in \mathbb{F}_2^n$ the capacity $C_{v,W}$ increases. In general, the data complexity, as indicated by (16), depends on the balance between the factor $|W|$ and the effect of the capacity $C_{v,W}$ on the squared differential-linear bias.

6.3 Estimation of the Bias

As in the one-dimensional case, we discuss in this section some conditions on which we can compute the bias of a multidimensional differential-linear approximation. The approach is similar to the one of Sect. 5.

Given a set V, the sum (15) can be decomposed into two sums:

$$\mathcal{E}_{U,W} = \frac{2}{|W|}\sum_{v\in V, v\neq 0} \varepsilon_{U,v}C_{v,W} + \frac{2}{|W|}\sum_{v\notin V} \varepsilon_{U,v}C_{v,W} \tag{17}$$

Practical computation of the bias of a multidimensional differential-linear approximation relies on the fact that computing only the first partial sum gives us an underestimate of the absolute bias $|\mathcal{E}_{U,W}|$.

Assumption 2. *We assume that*

$$|\mathcal{E}_{U,W}| \geq \left|\frac{2}{|W|}\sum_{v\in V, v\neq 0} \varepsilon_{U,v}C_{v,W}\right|.$$

It is straightforward to generalize Theorem 3 to the multidimensional case.

Corollary 2. *Let $\mathcal{E}_{U,V} = \mathbf{P}[U^{\perp}\setminus\{0\} \xrightarrow{E_0} V^{\perp}] - \frac{1}{|V|}$ be the bias of a truncated differential with non-zero input differences in $U^{\perp}$ and output differences in $V^{\perp}$. Further, we denote by $C_{V,W} = \sum_{w\in W, w\neq 0}\sum_{v\in V} c_{v,w}^2$ the capacity of the multidimensional linear approximation.*
If then, for all $\Delta \notin V^{\perp}$ the probabilities $\mathbf{P}[U^{\perp}\setminus\{0\} \xrightarrow{E_0} \Delta]$ are equal, we have

$$\frac{2}{|W|}\sum_{v\in V}\varepsilon_{U,v}C_{v,W} = \frac{1}{|W|}\frac{|V|}{|V|-1}\varepsilon_{U,V}C_{V,W}.$$

To test the validity of the results presented in this section, similar experiments than the ones presented in Sect. 5.2 have been conducted on SmallPresent[8]. Conclusion of these experiments are similar to the ones in the one-dimensional case. In the case of the PRESENT cipher, these experiments show that

$$\left| \frac{2}{|W|} \sum_{v \in V} \varepsilon_{U,v} C_{v,W} \right|,$$

is an underestimate of the absolute bias of the multidimensional differential-linear approximation. As in Sect. 5.2, we observe that the assumption about the equality of the probabilities $\mathbf{P}[U^{\perp} \setminus \{0\} \xrightarrow{E_0} \Delta]$ made in Corollary 2, influences the computational result when $|V|$ is large.

7 Conclusion

In this paper, we studied and generalized the differential-linear cryptanalysis. Starting from the observation that any differential-linear relation can be regarded as a truncated differential or a multidimensional linear approximation we derive a general expression of its bias based on the link between differential probabilities and linear correlations provided by Chabaud and Vaudenay.

We also revisit previous studies and applications of differential-linear cryptanalysis, where the bias of the differential-linear approximation has often been estimated under some heuristic assumptions, implicitly or explicitly present in the derivations. We derive our general formula of the bias under the sole assumption of round independence of the parts of the cipher, and identify new additional assumptions for computing efficient estimates of it. Extensive experiments have been performed to test the validity of these assumptions. Although no new applications of differential-linear cryptanalysis are presented in this paper, the potential and generality of our sound framework is demonstrated by its ability to explain existing examples of differential-linear cryptanalysis.

References

1. Anderson, R., Biham, E., Knudsen, L.R.: Serpent: A Proposal for the Advanced Encryption Standard. In: NIST AES Proposal (1998)
2. Baignères, T.: *Quantitative Security of Block Ciphers*: Designs and Cryptanalysis Tools. Ph.D. thesis, École polytechnique fédérale de Lausanne (2008)
3. 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)
4. 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)
5. 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)

6. Biham, E., Dunkelman, O., Keller, N.: Differential-Linear Cryptanalysis of Serpent. In: Johansson, T. (ed.) FSE 2003. LNCS, vol. 2887, pp. 9–21. Springer, Heidelberg (2003)
7. Biham, E., Dunkelman, O., Keller, N.: New Combined Attacks on Block Ciphers. In: Gilbert, H., Handschuh, H. (eds.) FSE 2005. LNCS, vol. 3557, pp. 126–144. Springer, Heidelberg (2005)
8. Biham, E., Shamir, A.: Differential Cryptanalysis of DES-Like Cryptosystems. In: Menezes, A., Vanstone, S.A. (eds.) CRYPTO 1990. LNCS, vol. 537, pp. 2–21. Springer, Heidelberg (1991)
9. Biryukov, A., De Cannière, C., Quisquater, M.: On Multiple Linear Approximations. In: Franklin, M. (ed.) CRYPTO 2004. LNCS, vol. 3152, pp. 1–22. Springer, Heidelberg (2004)
10. Blondeau, C., Gérard, B., Tillich, J.-P.: Accurate Estimates of the Data Complexity and Success Probability for Various Cryptanalyses. Des. Codes Crypt. **59**(1–3), 3–34 (2011)
11. Blondeau, C., Nyberg, K.: New Links Between Differential and Linear Cryptanalysis. In: Johansson, T., Nguyen, P.Q. (eds.) EUROCRYPT 2013. LNCS, vol. 7881, pp. 388–404. Springer, Heidelberg (2013)
12. Blondeau, C., Nyberg, K.: Links Between Truncated Differential and Multidimensional Linear Properties of Block Ciphers and Underlying Attack Complexities. In: Nguyen, P.Q., Oswald, E. (eds.) EUROCRYPT 2014. LNCS, vol. 8441, pp. 165–182. Springer, Heidelberg (2014)
13. Bogdanov, A., Boura, C., Rijmen, V., Wang, M., Wen, L., Zhao, J.: Key Difference Invariant Bias in Block Ciphers. In: Sako, K., Sarkar, P. (eds.) ASIACRYPT 2013, Part I. LNCS, vol. 8269, pp. 357–376. Springer, Heidelberg (2013)
14. Bogdanov, A.A., Knudsen, L.R., Leander, G., Paar, C., Poschmann, A., Robshaw, M., Seurin, Y., Vikkelsoe, C.: PRESENT: An Ultra-Lightweight Block Cipher. In: Paillier, P., Verbauwhede, I. (eds.) CHES 2007. LNCS, vol. 4727, pp. 450–466. springer, Heidelberg (2007)
15. Bogdanov, A., Rijmen, V.: Zero-Correlation Linear Cryptanalysis of Block Ciphers. IACR Cryptology ePrint Archive 2011:123 (2011)
16. Bogdanov, A., Tischhauser, E.: On the Wrong Key Randomisation and Key Equivalence Hypotheses in Matsui's Algorithm 2. In: Moriai, S. (ed.) FSE 2013. LNCS, vol. 8424, pp. 19–38. Springer, Heidelberg (2014)
17. Borst, J., Knudsen, L.R., Rijmen, V.: Two Attacks on Reduced IDEA. In: Fumy, W. (ed.) EUROCRYPT 1997. LNCS, vol. 1233, pp. 1–13. Springer, Heidelberg (1997)
18. Carlet, C.: Boolean Functions for Cryptography and Error Correcting. In: Crama, Y., Hammer, P.L. (eds.) Boolean Models and Methods in Mathematics. Computer Science, and Engineering, pp. 257–397. Cambridge University Press, Oxford (2010)
19. 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)
20. Cho, J.Y.: Linear Cryptanalysis of Reduced-Round PRESENT. In: Pieprzyk, J. (ed.) CT-RSA 2010. LNCS, vol. 5985, pp. 302–317. Springer, Heidelberg (2010)
21. Daemen, J., Rijmen, V.: The Design of Rijndael: AES - The Advanced Encryption Standard. Springer, New York (2002)
22. Dunkelman, O., Indesteege, S., Keller, N.: A Differential-Linear Attack on 12-Round Serpent. In: Chowdhury, D.R., Rijmen, V., Das, A. (eds.) INDOCRYPT 2008. LNCS, vol. 5365, pp. 308–321. Springer, Heidelberg (2008)

23. Hermelin, M., Cho, J.Y., Nyberg, K.: Multidimensional Extension of Matsui's Algorithm 2. In: Dunkelman, O. (ed.) FSE 2009. LNCS, vol. 5665, pp. 209–227. Springer, Heidelberg (2009)
24. 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)
25. Knudsen, L.R.: Truncated ana Higher Order Differentials. In: Preneel, B. (ed.) FSE 1994. LNCS, vol. 1008, pp. 196–211. Springer, Heidelberg (1995)
26. Knudsen, L.R.: DEAL- A 128-bit Block-Cipher. In: NIST AES Proposal (1998)
27. Lai, X.: Higher Order Derivatives and Differential Cryptanalysis. In: Blahut, R.E., Costello, D.J., Maurer, U., Mittelholzer, T. (eds.) Communications and Cryptography. The Springer International Series in Engineering and Computer Science, vol. 276, pp. 227–233. Springer, US (1994)
28. Lai, X., Massey, J.L.: Markov Ciphers and Differential Cryptanalysis. In: Davies, D.W. (ed.) EUROCRYPT 1991. LNCS, vol. 547, pp. 17–38. Springer, Heidelberg (1991)
29. Langford, S.K.: Differential-Linear Cryptanalysis and Threshold Signatures. Ph.D. Thesis (1995)
30. 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)
31. Leander, G.: Small Scale Variants Of The Block Cipher PRESENT. IACR Cryptology ePrint Archive 2010:143 (2010)
32. Liu, Z., Gu, D., Zhang, J., Li, W.: Differential-Multiple Linear Cryptanalysis. In: Bao, F., Yung, M., Lin, D., Jing, J. (eds.) Inscrypt 2009. LNCS, vol. 6151, pp. 35–49. Springer, Heidelberg (2010)
33. Lu, J.: A Methodology for Differential-Linear Cryptanalysis and Its Applications. In: Canteaut, A. (ed.) FSE 2012. LNCS, vol. 7549, pp. 69–89. Springer, Heidelberg (2012)
34. Matsui, M.: Linear Cryptanalysis Method for DES Cipher. In: Helleseth, T. (ed.) EUROCRYPT 1993. LNCS, vol. 765, pp. 386–397. Springer, Heidelberg (1994)
35. McLaughlin, J., Clark, J.A.: Filtered nonlinear cryptanalysis of reduced-round Serpent, and the Wrong-Key Randomization Hypothesis. Cryptology ePrint Archive, Report 2013/089 (2013)
36. Nguyen, P.H., Wu, H., Wang, H.: Improving the Algorithm 2 in Multidimensional Linear Cryptanalysis. In: Parampalli, U., Hawkes, P. (eds.) ACISP 2011. LNCS, vol. 6812, pp. 61–74. Springer, Heidelberg (2011)
37. Nyberg, K.: Linear Approximation of Block Ciphers. In: De Santis, A. (ed.) EUROCRYPT 1994. LNCS, vol. 950, pp. 439–444. Springer, Heidelberg (1995)
38. Selçuk, A.A.: On Probability of Success in Linear and Differential Cryptanalysis. J. Crypt. **21**(1), 131–147 (2008)
39. Wagner, D.: The Boomerang Attack. In: Knudsen, L.R. (ed.) FSE 1999. LNCS, vol. 1636, pp. 156–170. Springer, Heidelberg (1999)
40. Wagner, D.: Towards a Unifying View of Block Cipher Cryptanalysis. In: Roy, B., Meier, W. (eds.) FSE 2004. LNCS, vol. 3017, pp. 16–33. Springer, Heidelberg (2004)

Improved Slender-Set Linear Cryptanalysis

Guo-Qiang Liu[1(✉)], Chen-Hui Jin[1], and Chuan-Da Qi[2]

[1] Information Science and Technology Institute, Zhengzhou 450000, Henan, China
liuguoqiang87@hotmail.com
jinchenhui@126.com
[2] Xinyang Normal University, Xinyang 464000, Henan, China
qichuanda@sina.com

Abstract. In 2013, Borghoff *et al.* introduced a slender-set linear cryptanalysis on PRESENT-like ciphers with key-dependent secret S-boxes. In this paper, we propose an improved slender-set linear attack to PRESENT-like ciphers with secret S-boxes. We investigate three new cryptanalytic techniques, and use them to recover the secret S-boxes efficiently. Our first new idea is that we propose a new technique to support consistency of partitions of the input to the secret S-boxes. Our second new technique is that we present a more efficient method to recover the coordinate functions of secret S-boxes based on more information than that of Borghoff's attack. The third new technique is that we propose a method of constructing all correct coordinate function of secret S-boxes by pruning search algorithm. In particular, we implemented a successful linear attack on the full round Maya in practice. In our experiments, the correct S-box can be recovered with 2^{36} known plaintexts, $2^{18.9}$ time complexity and negligible memory complexity at a success rate of 87.5 % based on 200 independent trials. Our attack is the improvement and sequel of Borghoff's work on PRESENT-like cipher with secret S-boxes.

Keywords: Block cipher · Linear cryptanalysis · PRESENT-like · Secret S-box

1 Introduction

Block ciphers are a class of cryptographic algorithms which are widely used. After the establishment of Advanced Encryption Standard (AES), there has been a need for cryptographic solutions which can provide low cost security for small device, such as RFID tags or sensor networks. This opens up the study field of lightweight block cipher.

In recent years, many lightweight block ciphers have been developed, such as mCrypton [14], HIGHT [11], SEA [20], DESXL [13], KATAN [8], MIBS [12]. PRESENT [4] is the most remarkable representative lightweight block cipher. It is proposed by Bogdanov *et al.* in CHES 2007. PRESENT cipher is an iterated 31 rounds SPN block cipher with a 64-bit block size. PRESENT has two variants of key, one with an 80-bit and the other with a 128-bit. Each round of PRESENT

C. Cid and C. Rechberger (Eds.): FSE 2014, LNCS 8540, pp. 431–450, 2015.
DOI: 10.1007/978-3-662-46706-0_22

cipher has three layers. The first layer is a substitution layer, which consists of 16 parallel applications of the same 4-bit S-box. The second layer is a permutation layer, which consists of a bit-wise permutation of 64-bit. The last layer is a key addition layer, where a round key is exclusive-ored to the text. Due to the small 4-bit S-box, bit-wise permutation and Xor addition, PRESENT has a fast and compact hardware implementation. In recent years, the cryptanalysis on PRESENT has been actively performed so far. The best known cryptanalysis attack on PRESENT is a linear attack on 26 of the 31 rounds presented by Cho [9] in CT-RSA 2010. This attack could break the 26 rounds PRESENT with 2^{64} data complexity, 2^{32} memory accesses complexity and 2^{72} time complexity. Although this attack is hardly to perform in practice, the number of rounds used should not be dramatically reduced.

In order to strengthen the PRESENT cipher and reduce the number of rounds, Gomathisankaran and Lee [15] presented a PRESENT-like cipher with secret S-boxes which is named Maya. The Maya cipher is a 16 rounds SPN cipher similar to PRESENT. The difference between PRESENT and Maya is that the substitution layer of Maya consists of 16 different S-boxes which are key-dependent and unknown.

Linear cryptanalysis [16,17] is a classical tool for analyzing block ciphers. It was introduced by Matsui in 1993. Since for a random permutation, the probability of any linear relation between the plaintext and corresponding ciphertext bits should be balanced at 1/2. But this is not always the same in the case of block ciphers. The attacker can construct a distinguisher for a key-recovery attack based on this information leakage.

Nowadays, there are only a few papers investigating the security of PRESENT-like cipher with secret S-boxes. In 2011, Borghoff *et al.* [6] proposed a slender-set differential cryptanalysis of PRESENT-like cipher with randomly chosen secret S-boxes. In 2013, Borghoff *et al.* [7] outline how to attack this type of cipher with a linear attack by using slender-set. However, the slender-set linear cryptanalysis proposed in [7] was simply described and has something to be improved. In this paper, we present an improved slender-set linear cryptanalysis on PRESENT-like cipher.

Our Results. In this paper, we investigate an improved slender-set linear cryptanalysis of PRESENT-like cipher with secret S-boxes. Our contributions are threefold. First, we present a new technique to support consistency of partitions of the input to the secret S-box of the first S-box layer. This modification makes it possible to transform the original vectors into binary vectors more reasonably. Our second new idea is that we propose a new method to get the coordinate functions of secret S-boxes efficiently based on more information. This method considers all the information that comes from the candidate coordinate functions together instead of the three longest candidate coordinate functions. The third new technique is that we present a method of constructing the correct coordinate functions by pruning search algorithm, which can help us to recover the secret S-boxes more efficiently. Finally, we focus on the settings of PRESENT-like cipher where the secret S-boxes are key-dependent and are repeated for the

first and last rounds. We propose an effective method of determining the correct S-box from the equivalent S-boxes with low time complexity. In particular, we implemented an improved slender-set linear attack to the full round Maya successfully. In our experiments, the correct S-box can be recovered with 2^{36} data complexity, $2^{18.9}$ time complexity and negligible memory complexity at a success rate of 87.5 %. The correct S-box is usually found in less than a few minutes on a standard PC. Our attack is the improvement and sequel of Borghoff's work on PRESENT-like cipher with secret S-boxes.

Related Work. There are lots of design and analysis to ciphers with secret S-boxes. In 1994, Gilbert and Chauvaud [10] presented a differential attack on the cipher Khufu which used secret S-boxes. Biham and Biryukov [1] described methods of strengthening DES, without slowing encryption speed, in which key-dependency is added by simply XORing key material before and after the S-boxes calculation. They studied the strengthened version of DES against exhaustive search, differential, and linear attacks that can be used existing hardware. In 1996, Vaudenay [23] provided a cryptanalysis of reduced-round variants of Blowfish, in which the S-boxes were randomly generated from the private key. In 1998, Schneier and Kelsey *et al.* [19] studied a new encryption algorithm with key-dependent S-boxes. In 2001, Biryukov and Shamir [2,3] investigated the security of iterated ciphers, in which the substitutions and permutations are all secret and key-dependent. In 2009, Borghoff *et al.* [5] researched on the cipher C2, which has a secret S-box. In 2011, Szaban and Seredynski [22] proposed cryptanalysis of a methodology to design dynamic cellular automata (CA)-based S-boxes. Stoianov [21] presented and analyzed an approach to change the S-boxes used in the algorithm AES. In 2013, Peng and Jin [18] proposed a cryptanalysis of a new method to design key-dependent S-boxes by using hyperchaotic Chen system.

Organization. The paper is organized as follows. Section 2 introduces the structure of PRESENT-like cipher. Section 3 outlines the slender-set linear attack on PRESENT-like cipher described in [7]. Section 4 presents our improved slender-set linear cryptanalysis on PRESENT-like cipher with secret S-boxes. Section 5 gives experimental results of our attack on Maya cipher. Finally, Sect. 6 concludes the paper.

2 The PRESENT-like Cipher

The PRESENT-like cipher is an n-bit SPN block cipher. The round function consists of round key, S-boxes and permutations. Assuming that the number of rounds is N.

(1) Round key K: n-bit round key is Xored to the text.

(2) S-box S: n/m key dependent and different m-bit secret S-boxes.

(3) P-box P: The bit-wise permutation between the S-box layers is public or secret.

The PRESENT-like cipher can be described in Algorithm 1.

Algorithm 1. N-rounds PRESENT-like cipher

Require: n-bit plaintext X; main key K
Ensure: n-bit ciphertext $C = E_K(X)$
1: Derive n/m m-bit S-boxes S_i and round keys K_i $(1 \leq i \leq n/m)$ from K
2: $STATE = X$
3: **for** $i = 1$ to N **do**
4: Parse $STATE$ as $STATE_1||STATE_2||\cdots||STATE_{n/m}$
5: **for** $j = 1$ to n/m **do**
6: $STATE_j = S_j(STATE_j)$
7: Apply bit permutation to $STATE$
8: Add round key K_i to $STATE$
9: **end for**
10: **end for**
11: **return**

Maya is a typical example of the PRESENT-like cipher described in Algorithm 1 with $n = 64$ and $m = 4$

3 Borghoff's Slender-Set Linear Attack

In this section, we review the Borghoff's slender-set linear attack of recovering the secret S-boxes described in [7]. We call the attack in [7] as *Borghoff's linear attack* for short. First, we introduce some notations and definitions used in this paper.

3.1 Linear Cryptanalysis

We follow the notations used in [7]. The canonical inner product on F_2^n is denoted by $\langle \cdot, \cdot \rangle$, that is

$$\langle (a_0, a_1, \cdots, a_{n-1}), (b_0, b_1, \cdots, b_{n-1}) \rangle = \sum_{i=0}^{n-1} a_i b_i$$

Given a function $H : F_2^n \to F_2^m$, for an m -bit output mask β and an n-bit input mask α, the bias ε of the linear approximation $\langle \beta, H(x) \rangle + \langle \alpha, x \rangle$ is defined by

$$p(\langle \beta, H(x) \rangle + \langle \alpha, x \rangle = 0) = \frac{1}{2} + \varepsilon$$

where the probability p is taken over all choices of inputs x. The Walsh or Fourier-transform of H at the pair $(\alpha, \beta) \in F_2^n \times F_2^m$ is defined by

$$\hat{H}(\alpha, \beta) = \sum_{x \in F_2^n} (-1)^{\langle \beta, H(x) \rangle + \langle \alpha, x \rangle}$$

The relation between the Fourier transform of H and the bias of a linear approximation is given by

$$\varepsilon = \frac{\hat{H}(\alpha, \beta)}{2^{n+1}}$$

Thus to study the bias $\langle \beta, H(x) \rangle + \langle \alpha, x \rangle$ is equivalent to study the Fourier-transform of H at the pair (α, β).

3.2 Brief Description of Borghoff's Linear Attack

In this section, we introduce the basic principle of Borghoff's linear Attack. In [7], Borghoff *et al.* started with a differential-style attack on PRESENT-like cipher. They supposed that the characteristics with single-bit differences at the output of the S-box layer in the first round have a stronger correlation with single-bit differences at the input to the S-box layer in the last round. They named this differential attack as the slender-set differential cryptanalysis. Using a similar hypothesis for linear characteristics, they could get useful information about the secret S-boxes with the single-bit mask at the output of the S-box layer in the first round and the low-weight mask at the input of the S-box layer in the last round. Thus we can name this linear attack as slender-set linear cryptanalysis. In the following, we outline Borghoff's linear attack.

Without loss of generality, Borghoff's linear attack focused on the leftmost S-box, which is denoted simply by S. All other S-boxes can be processed similarly. We denote that

$$F : F_2^4 \times F_2^{60} \to F_2^{64} \quad and \quad F(x, y) = c$$

where the function F is the encryption function that starts after the first layer of S-boxes, x is the output of the leftmost S-box of the first S-box layer, and y is concatenation of the outputs of the remaining S-boxes of the first S-box layer. Thus the Fourier-transform of F at the pair $((\alpha_1, \alpha_2), \beta) \in F_2^{64} \times F_2^{64}$ is denoted by

$$\hat{F}(\alpha, \beta) = \hat{F}((\alpha_1, \alpha_2), \beta) = \sum_{x \in F_2^4} \sum_{y \in F_2^{60}} (-1)^{\langle \beta, F(x,y) \rangle + \langle \alpha_1, x \rangle + \langle \alpha_2, y \rangle}$$

In [12], Borghoff *et al.* used of the bias of the value $\langle \beta, F(x, y) \rangle$ for a fixed values of x. They denote the corresponding function by

$$T_x : F_2^{60} \to F_2^{64} \quad and \quad T_x(y) = F(x, y)$$

and we look at

$$\hat{T}_x(0, \beta) = \sum_{y \in F_2^{60}} (-1)^{\langle \beta, T_x(y) \rangle} = \sum_{y \in F_2^{60}} (-1)^{\langle \beta, F(x,y) \rangle}$$

We list two useful lemmas in [7] as follows.

Lemma 1. *[7] With the notation from above, it holds that*

$$2^4\hat{T}_\lambda(0,\beta) = \sum_{\alpha_1 \in F_2^4} (-1)^{\langle \alpha_1, \lambda \rangle} \hat{F}((\alpha_1, 0), \beta)$$

Denote the whole encryption function by E. They split its input as before and consider

$$E : F_2^4 \times F_2^{60} \to F_2^{64} \quad \textit{and} \quad E(x,y) = c$$

where x is now the input to the first S-box, and y is the input to all other S-boxes. They define the function corresponding to fixing x as T'_x, that is

$$T'_x : F_2^{60} \to F_2^{64} \quad \textit{and} \quad T'_x(y) = E(x,y)$$

For a selection of masks β and each possible value of x, they estimate the value of $T'_x(0,\beta)$. This is simply done by encrypting a number of known plaintexts with the first four bits fixed to x and counting how often the value of

$$\langle \beta, T'_x(y) \rangle = \langle \beta, E(x,y) \rangle$$

is zero (resp., one). The next lemma is the key for deducing information about the secret S-box.

Lemma 2. *[7] With the notation from above, the bias of $\langle \beta, T'_x(y) \rangle$ is equal to the bias of $\langle \beta, T_{S(x)}(y) \rangle$. That is*

$$\hat{T}'_x(0,\beta) = \hat{T}_{S(x)}(0,\beta)$$

In [7], they assume that, for a given mask β, there is exactly one mask α such that $\hat{F}((\alpha,0),\beta)$ is high while for any $\xi \neq \alpha$ the value $\hat{F}((\xi,0),\beta)$ is close to zero. According to Lemma 1 and Lemma 2, we have

$$\begin{aligned} \hat{T}'_x(0,\beta) = \hat{T}_{S(x)}(0,\beta) &= 2^{-4} \sum_{\xi \in F_2^4} (-1)^{\langle \xi, S(x) \rangle} \hat{F}((\xi,0),\beta) \\ &\approx 2^{-4}(-1)^{\langle \alpha, S(x) \rangle} \hat{F}((\alpha,0),\beta) \end{aligned} \tag{1}$$

In this way, we can partition the values of x into two equally-sized sets V_0 and V_1 depending on the sign of $\hat{T}'_x(0,\beta)$, where $V_\gamma = \{x | \langle \alpha, S(x) \rangle = \gamma\}, \gamma = 0, 1$. If we get all four linearly independent coordinate functions of secret S-box, such as $(\langle 2^i, S(0) \rangle, \langle 2^i, S(1) \rangle, \cdots, \langle 2^i, S(15) \rangle), 0 \leq i \leq 3$, we can recover the secret S-box. Borghoff's linear attack is summarized as the following steps:

Step1. Let the output mask $\beta = 0^{4j}||b||0^{60-4j}$, $0 \leq j \leq 15$. For every leftmost input $0 \leq x \leq 15$ and for every $1 \leq b \leq 15$, we estimate the value of the counter $\hat{T}'_x(0,\beta)$ by Eq. (1) after encrypting enough plaintexts.

Step2. After 15 vectors $W_\beta = (\hat{T}'_0(0,\beta), \hat{T}'_1(0,\beta), \cdots, \hat{T}'_{15}(0,\beta))$ being retrieved, we identify the three longest vectors using the Euclidean norm as a metric, as Borghoff *et al.* assume that these vectors contain the most reliable information. We transform each of these vectors into a binary vector such that the eight highest counter values correspond to '1'-bits and the remaining correspond to '0'-bits.
Step3. We take a majority vote among these three binary vectors to find the correct coordinate functions of secret S-box.
Step4. We recover the 4-bit secret S-boxes based on four linearly independent coordinate functions of secret S-boxes.

Through the above steps, we may partition the value of x into two sets $V_\gamma = \{x|\langle\alpha, S(x)\rangle = \gamma\}, \gamma = 0, 1$. However, the value of α and γ are unknown. Borghoff *et al.* pointed out that they might reveal one or more of the sets $\{x|\langle 2^i, S(x)\rangle = 0\}, 0 \leq i \leq 3$ by repeating the steps described above for other value of $\beta = 0^{4j}||b||0^{60-4j}$ with different $0 \leq j \leq 15$.

However, the linear cryptanalysis proposed in [7] was in less details and has something to be improved.

First, according to Eq. (1), we can only partition the value of x into two sets $V_\gamma = \{x|\langle\alpha, S(x)\rangle = \gamma\}, \gamma = 0, 1$ by Borghoff's linear attack. However, for a fixed α, the value of γ is unknown, which would cause problems: for a fixed α, we only know which values of x belonging to the same set, but we still do not know the values of x belonging to which set. In other words, if one coordinate of the binary vector is equal to "0", it dose not mean the corresponding value of coordinate function is essentially equal to "0". Furthermore, the sign of $\hat{F}((\alpha, 0), \beta)$ might be opposite for the same α and the different β. According to Eq. (1), the opposite sign of $\hat{F}((\alpha, 0), \beta)$ will cause opposite sign of $\hat{T}'_x(0, \beta)$. This makes it possible to partition the x into different set with the same α and the different β. We did the same work as Borghoff *et al.* did. As an example, we carried out this attack using 2^{25} known plaintexts on the 10 rounds PRESENT-like cipher. The three longest vectors were these:

$(-3138, -2218, -3156, 3146, -2486, 1784, -2974, -3452, 1392, 1602, 2850, 3198, -3100, 2796, -3458, 1708)$

$(-2558, -1768, -2022, 2798, -1754, 2538, -1808, -2440, 2784, 2694, 2424, 3378, -2576, 2378, -2658, 2424)$

$(3046, 1842, 1730, -2982, 1952, -1600, 2116, 2930, -2426, -2742, -2036, -2440, 2918, -1764, 3112, -1670)$

After transforming these vectors into binary vectors as described, one gets

$$(0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 1)$$

$$(0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 1)$$

$$(1, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0)$$

We can see that the partition of x in the third binary vector is opposite comparing with that in the first and second binary vectors. This problem will bring some noise when we recover the correct coordinate functions of secret S-box with majority vote. To overcome this, we develop a new idea to support consistency of the partition of x.

Second, due to that Borghoff *et al.* just used the information from the three longest vectors, which will lose the information from the candidate binary vectors. Furthermore, the majority vote for getting "true" vector in [7] is not reasonable and to be perfected. In order to improve the information collection phase, we present a new method of making full use of information from the 240 output low-weight masks $\beta = 0^{4j}||b||0^{60-4j}, 0 \leq j \leq 15, 1 \leq b \leq 15$ of the S-box layer in the last round instead of three longest vectors. And we introduce an improved voting method for constructing the correct candidate coordinate functions.

Third, for PRESENT-like cipher, the experiment shows that we can only get one correct set $\{x|\langle 2^i, S(x)\rangle = 0\}, 0 \leq i \leq 3$ in most case by using Borghoff's linear attack. As an example, we carried out Borghoff's attack by using 2^{22} to 2^{27} known plaintexts on the 10 rounds PRESENT-like cipher. However, we can only get one or two correct coordinate functions of secret S-box (see Table 3). This makes it possible to recover 1 or 2 out of 4 bit information of secret S-box in most case, but not the 4-bit information about secret S-box. The question is: How to get more correct coordinate functions of secret S-box as many as possible with lower data complexity? To overcome this problem, we proposed method of constructing all correct coordinate functions of secret S-box by using pruning search algorithm.

4 Improved Slender-Set Linear Cryptanalysis

In this section, we explain the approach of our improved slender-set linear cryptanalysis of recovering the secret S-box. Comparing with the Borghoff's linear attack, there are three main improvements in this paper. First, we present a new technique to avoid the complementary vectors in the candidate coordinate functions of secret S-boxes. Second, we propose an efficiently method of getting full use of all the information coming from the active S-box synthetically. In other words, we consider all the $15 \times 16 = 240$ vectors together instead of the three longest vectors. Third, we introduce an effective filter to construct correct coordinate functions of secret S-boxes by using pruning search algorithm. The pruning search algorithm can help us to search the correct coordinate functions as many as possible with lower complexity. Finally, we focus on the settings where the S-boxes are key-dependent and are repeated for the first and last rounds. We investigate an efficient method of recovering the secret S-box uniquely with a low time complexity.

Notation. Throughout this section, assuming that α is a binary vectors, the Hamming weight of α is denoted by $wt(\alpha)$. The complementary vector of α is

denoted by $\overline{\alpha}$. The vector $W_\beta = (\hat{T}'_0(0,\beta), \hat{T}'_1(0,\beta), \cdots, \hat{T}'_{15}(0,\beta))$ described in Sect. 3.2 is called as the *original vector*.

4.1 Recovering the Correct Coordinate Functions

We start with the first new technique of our improved slender-set linear cryptanalysis. In Borghoff's linear attack, the value of corresponding coordinate in different binary vectors may be different. But the partition of x is valid in one binary vector. In this case, for a fixed k, we consider to partition x by the distance between $\hat{T}'_i(0,\beta)$ and $\hat{T}'_k(0,\beta)$, where $0 \le i \le 15$. Without loss of generality, we let $k = 0$ in our paper. We propose the notion of *relative distance* to support consistency of the partition of x as follows.

Specifically, for every $0 \le j \le 15, 1 \le b \le 15$ we consider the output masks $\beta = 0^{4j}||b||0^{60-4j}$, and we get $15 \times 16 = 240$ *original vectors* $W_\beta = (\hat{T}'_0(0,\beta), \hat{T}'_1(0,\beta), \cdots, \hat{T}'_{15}(0,\beta))$ after encrypting enough plaintexts. For every β and every $0 \le i \le 15$, we compute the *relative distance*

$$\mathbb{R}^{(i)}_\beta = \hat{T}'_0(0,\beta) - \hat{T}'_i(0,\beta)$$

between $\hat{T}'_0(0,\beta)$ and $\hat{T}'_i(0,\beta)$. We transform each of these *relative distance* vectors $(\mathbb{R}^{(0)}_\beta, \mathbb{R}^{(1)}_\beta, \cdots, \mathbb{R}^{(15)}_\beta)$ into binary vectors $(\mathbb{B}^{(0)}_\beta, \mathbb{B}^{(1)}_\beta, \cdots, \mathbb{B}^{(15)}_\beta)$ such that the eight highest values correspond to "1"-bits and the remaining values correspond to "0"-bits. If $\mathbb{B}^{(0)}_\beta$ is equal to "1", then we transform the vector $(\mathbb{B}^{(0)}_\beta, \mathbb{B}^{(1)}_\beta, \cdots, \mathbb{B}^{(15)}_\beta)$ into the complementary vector $B_i = (\overline{\mathbb{B}^{(0)}_\beta}, \overline{\mathbb{B}^{(1)}_\beta}, \cdots, \overline{\mathbb{B}^{(15)}_\beta})$. If $\mathbb{B}^{(0)}_\beta$ is equal to "0", then we let $B_i = (\mathbb{B}^{(0)}_\beta, \mathbb{B}^{(1)}_\beta, \cdots, \mathbb{B}^{(15)}_\beta)$.

As an example, we obtained the vectors described in Sect. 3.2. The *relative distance* vector are these for the three longest vectors:

$$(0, 920, -18, 6284, 652, 4922, 164, -314, 4530, 4740, 5988, 6336, 38, 5934, -320, 4846)$$

$$(0, 790, 536, 5356, 804, 5096, 750, 118, 5342, 5252, 4982, 5936, -18, 4936, -100, 4982)$$

$$(0, -1204, -1316, -6028, -1094, -4646, -930, -116, -5472, -5788, -5082, -5486, -128, -4810, 66, -4716)$$

We transform the *relative distance* vector into binary vectors as follows:

$$(0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 1)$$

$$(0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 1)$$

$$(0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 1)$$

The benefit of this technique is that the value of the first coordinate of binary vector B_i is identically equal to "0". By using this method, we can support consistency of the partition of input x based on the *original vector* $W_\beta = (\hat{T}'_0(0,\beta), \hat{T}'_1(0,\beta) \cdots, \hat{T}'_{15}(0,\beta))$.

After we get the 240 binary vectors $B_i, 1 \leq i \leq 240$, we introduce the second technique of our improved attack as follows.

We define the distances between two binary vectors B_i and B_j by

$$\mathbb{D}_{B_i,B_j} = wt(\overline{B_i \oplus B_j})$$

where $1 \leq i, j \leq 240$ and $wt(\overline{B_i \oplus B_j})$ is the *Hamming weight* of $\overline{B_i \oplus B_j}$. Due to the first coordinate of the binary vectors being identically equal to "0", it holds that $\mathbb{D}_{B_i,B_j} \neq 0$. Thus we can see that $\mathbb{D}_{B_i,B_j} = 2, 4, 6, 8, 10, 12, 14, 16$ and the number of $\mathbb{D}_{B_i,B_j}$ is equal to $C_{240}^2 = 28680$. For two random vectors α and β, we present the probability distribution of $wt(\overline{\alpha \oplus \beta})$ as following lemma.

Lemma 3. *With the notation above, let α, β be random vectors and $wt(\alpha) = wt(\beta) = 8$ with $\alpha = (0, a_1, a_2, \cdots, a_{15}), \beta = (0, b_1, b_2, \cdots, b_{15}), a_i, b_i \in \{0, 1\}, 1 \leq i \leq 15$. Then the probability of $wt(\overline{\alpha \oplus \beta}) = k$ is equal to*

$$p(wt(\overline{\alpha \oplus \beta}) = k) = \frac{C_8^{(16-k)/2} C_7^{(16-k)/2}}{C_{15}^7}$$

where $k = 2, 4, 6, 8, 10, 12, 14, 16$.

We now compute the probability distribution of $wt(\overline{\alpha \oplus \beta})$ for two vectors α, β based on the notation in Lemma 3 (see Table 1).

Table 1. The probability distribution of $wt(\overline{\alpha \oplus \beta})$ for randomly chosen vectors α, β

k	Probability	k	Probability	k	Probability	k	Probability
2	0.1243 %	6	18.2751 %	10	30.4584 %	14	0.8702 %
4	3.0458 %	8	38.0730 %	12	9.1375 %	16	0.0155 %

Due to the method of *relative distance*, there have no complementary vectors in the binary vectors. According to Table 1, the expectation of $wt(\overline{\alpha \oplus \beta})$ is equal to 8.533 for two random vectors. For two binary vectors, they will be similar to each other when $wt(\overline{B_i \oplus B_j})$ approximate to 16. Thus we believe the vectors B_i and B_j would be closer to each other when $wt(\overline{B_i \oplus B_j}) > 8$.

Now we explain our approach for getting the information about secret S-box from all 240 binary vectors B_β. We assume that the binary vectors corresponding the same α are similar to each other. We start with the method of partition B_β by using the notion of *similarity degree* as following definition.

Definition 1. Given 240 binary vectors $B_i, 1 \leq i \leq 240$. Let $\xi, \tau > 0$ and $\mathbb{D}_{B_i,B_j} = wt(\overline{B_i \oplus B_j})$ be the distances between binary vectors B_i and B_j, where $1 \leq j \leq 240$. The *similarity degree* of B_i and B_j is defined by

$$\mathbb{S}_{B_i,B_j} = g(B_i, B_j) + \sum_{\substack{1 \leq k \leq 240 \\ k \neq i, k \neq j}} (f(B_i, B_k) + f(B_j, B_k))$$

where the function $g(B_i, B_j) = \begin{cases} \xi, & if\ wt(\overline{B_i \oplus B_j}) \geq t; \\ 0, & others. \end{cases}$ and $f(B_i, B_j) = \begin{cases} \tau, & if\ wt(\overline{B_i \oplus B_j}) \geq t; \\ 0, & others. \end{cases}$

According to Definition 1, the *similarity degree* of B_i and B_j considers not only the relationship between B_i and B_j but also the relationship from other B_k, which can help us to collect all the correlation between 240 candidate binary vectors synthetically with suitable value of t, ξ, τ. Figure 1 shows the form of *similarity degree*:

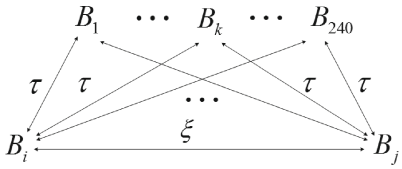


Fig. 1. The form of similarity degree

For 240 candidate binary vectors, we can get $C_{240}^2 = 28680$ *similarity degrees* between the binary vectors B_i and B_j. The higher value of *similarity degree*, the stronger correlation between two binary vectors.

In order to recover four coordinate functions of secret S-box, we start with approach of how to partition the 240 binary vectors into four parts as follows.

Given 240 binary vectors $B_i, 1 \leq i \leq 240$. As can be seen in Table 1, the probability of $\mathbb{D}_{B_i,B_j} = 16$ is equal to 0.0155 %. Such a small probability means that: If $\mathbb{D}_{B_i,B_j} = 16$, there must be a very strong correlation between two binary vectors B_i and B_j. In other words, the binary vectors B_i and B_j should be very close to the same coordinate function of secret S-box in case of $\mathbb{D}_{B_i,B_j} = 16$. Therefore, we treat the binary vectors which hold $\mathbb{D}_{B_i,B_j} = 16$ as the priority vectors. Assuming that we construct r priority sets $\Omega_k, 1 \leq k \leq r$, for every $k_i, k_j \in \Omega_k$, it must hold that $\mathbb{D}_{B_{k_i},B_{k_j}} = 16$. According to the definition of priority sets, the higher value of $|\Omega_k|$, the more information about one coordinate function of secret S-box. Therefore, we start with the priority set which the value of $|\Omega_k|$ is maximal. Let $k_i \in \Omega_k$. We choose a vector B_{k_i} from the 240 binary vectors and mark B_{k_i}. Then we sort 240 binary vectors in descending order up to the value of $\mathbb{S}_{B_{k_i},B_j}$ from $j = 1$ to $j = 240$. We check if the vector B_j is marked for every $1 \leq j \leq 240$. If B_j is unmarked, We add B_j into the first set Φ_1 until $|\Phi_1| = 60$. We stop repeating this method when we have identified four parts $\Phi_1, \Phi_2, \Phi_3, \Phi_4$.

Next, we propose the improved voting method of constructing the candidate coordinate functions of secret S-boxes based on these four sets $\Phi_1, \Phi_2, \Phi_3, \Phi_4$. In [7], Borghoff *et al.* proposed a majority voting method to get "true" vector from three longest vectors. It means that the weighting factor of each vector is considered equal. Compared to Borghoff's attack, our attack treats the similarity degree as the weighting factor in voting method as follows.

Let $\beta = (\beta_0, \beta_1, \cdots, \beta_{15})$. For a given set Φ_l, $1 \le l \le 4$, for each $0 \le x \le 15$, we compute

$$v_{l,x} = \sum_{\alpha \in \Phi_l} \left(\sum_{\beta \in \Phi_l, \beta_x = 0} \mathbb{S}_{\alpha,\beta} - \sum_{\beta \in \Phi_l, \beta_x = 1} \mathbb{S}_{\alpha,\beta} \right)$$

and transform the vectors $(v_{l,0}, v_{l,1}, \cdots, v_{l,15})$ into a binary vector which is denoted by V_l, such that the eight highest value correspond to "0"-bits and the remaining correspond to "1"-bits. Our method of finding four candidate coordinate functions of secret S-boxes is described in Algorithm 2.

Algorithm 2. Finding four candidate coordinate functions of secret S-boxes

Require:
The 240 binary vectors $B_i, 1 \le i \le 240$;
The value of t, ξ, τ;

Ensure: Four candidate coordinate functions V_1, V_2, V_3, V_4

1: According to the value of t, ξ, τ, for every $1 \le i, j \le 240$, we compute the similarity degrees $\mathbb{S}_{B_i,B_j}$.
2: Construct r priority sets $\Omega_k, 1 \le k \le r$.
3: Choose binary vector $w \in \Omega_k$, where the value of $|\Omega_k|$ is maximal. We mark w and sort 240 binary vectors in descending order up to the value of $\mathbb{S}_{w,B_j}$, $1 \le j \le 240$.
4: $l = 1$
5: **while** $l \le 4$ **do**
6: $n = 1$.
7: **for** $j = 1$ to 240 **do**
8: **if** B_j is unmarked **then**
9: Add B_j into the set Φ_l and $n \leftarrow n + 1$.
10: **else if** $n \le 60$ **then**
11: Continue.
12: **else**
13: Break.
14: **end if**
15: **end for**
16: For each $0 \le x \le 15$, we compute

$$v_{l,x} = \sum_{\alpha \in \Phi_l} \left(\sum_{\beta \in \Phi_l, \beta_x = 0} \mathbb{S}_{\alpha,\beta} - \sum_{\beta \in \Phi_l, \beta_x = 1} \mathbb{S}_{\alpha,\beta} \right)$$

where $\beta = (\beta_0, \beta_1, \cdots, \beta_{15})$. Then we transform the vectors $(v_{l,0}, v_{l,1}, \cdots, v_{l,15})$ into a binary vector V_l such that the eight highest value correspond to "0"-bits and the remaining correspond to "1"-bits.
17: $l \leftarrow l + 1$
18: **end while**
19: **return** Four candidate coordinate functions V_1, V_2, V_3, V_4.

However, in our experiment, the candidate vectors found by Algorithm 2 might not be complete the correct coordinate functions of secret S-box. In order to get all information about secret S-box, we propose a method of constructing all

correct coordinate functions of secret S-box by using pruning search algorithm. The efficient filtering method in pruning search algorithm is presented as the following proposition.

Proposition 1. *Let $r \leq m$, S-box $S(x) = (s_0(x), s_1(x), \cdots, s_{m-1}(x)) : \{0,1\}^m \rightarrow \{0,1\}^m$ be a bijective mapping, and $t_0, t_1, \cdots, t_{m-1}$ be a permutation of $0, 1, \cdots, m-1$. Then it holds that the function $h(x) = (s_{t_0}(x), s_{t_1}(x), \cdots, s_{t_{r-1}}(x)) : \{0,1\}^m \rightarrow \{0,1\}^r$ is balanced.*

In [7], the authors pointed out that the intersections of two correct sets (with different masks) contain exactly four values. our Proposition 1 extended the conclusion of their checking correctness.

For every correct vectors $V_i = (v_{i,0}, v_{i,1}, \cdots, v_{i,15}), 1 \leq i \leq r \leq 4$, it holds that

$$h(x) = \begin{pmatrix} 0 & 1 & \cdots & 15 \\ v_{1,0}||\cdots||v_{r,0} & v_{1,1}||\cdots||v_{r,1} & \cdots & v_{1,15}||\cdots||v_{r,15} \end{pmatrix}$$

is balanced. We call this filter the *balance filter*. In order to measure the effectiveness of *balance filter*, we present the probability of random vectors passing this filter.

Without loss of generality, let $a_i, b_i, c_i, d_i \in \{0,1\}, 1 \leq i \leq 15$, $A = (0, a_1, a_2, \cdots, a_{15})$, $B = (0, b_1, b_2 \cdots, b_{15})$, $C = (0, c_1, c_2, \cdots, c_{15})$, $D = (0, d_1, d_2, \cdots, d_{15})$ and $wt(A) = wt(B) = wt(C) = wt(D) = 8$. Then the probability of A, B, C, D passing the *balance filter* is equal to

$$\frac{15!}{(C_{15}^7)^4} \approx 2^{-10.36}$$

Such a small probability means the *balance filter* is a strong filter, and many wrong candidate coordinate functions may be found by this filter. The pruning search algorithm of constructing correct coordinate functions of secret S-boxes by using *balance filter* is described in Algorithm 3.

In conclusion, our attack for recovering the correct coordinate functions of secret S-boxes can be described as following steps:

Step1. We get 240 *original vectors* $W_\beta = (\hat{T}'_0(0,\beta), \hat{T}'_1(0,\beta), \cdots, \hat{T}'_{15}(0,\beta))$ after encrypting enough plaintexts. For every β, we compute the *relative distance* $\mathbb{R}^i_\beta$ between $\hat{T}'_0(0,\beta)$ and $\hat{T}'_i(0,\beta)$.
Step2. Since we have obtained 240 *relative distance* vectors $(\mathbb{R}^{(0)}_\beta, \mathbb{R}^{(1)}_\beta, \cdots, \mathbb{R}^{(15)}_\beta)$ in step 1, we transform these vectors into binary vectors B_β.
Step3. In this step we find four candidate coordinate functions by using Algorithm 2 based on 240 binary vectors B_β.
Step4. We search all four correct candidate coordinate functions of secret S-boxes by using Algorithm 3.

After the steps above, though we get four correct sets $V_{\gamma_i} = \{x | \langle \alpha_i, S(x) \rangle = \gamma_i\}, \gamma_i = 0, 1, 0 \leq i \leq 3$, we still do not know the value of α_i, γ_i. Therefore, we just recover the equivalent S-boxes of the secret S-boxes. In [7], Borghoff *et al.* pointed out that the single-bit mask at the output of the S-box layer in the

Algorithm 3. Constructing the sets of four candidate correct coordinate functions

Require:
Four candidate coordinate functions V_1, V_2, V_3, V_4 searched by Algorithm 2;
The sets $\Theta_t^{(k)}$, $t = 1, 2, 3, 4$, $k = 2, 4, 6, 8, 10, 12, 14, 16$. The vector $V_{t,r}^{(k)} \in \Theta_t^{(k)}$, is the one of the possible vectors with changing k bits compared with the vector V_t and with $wt(V_{t,r}^{(k)}) = 8$, where $1 \leq r \leq |\Theta_t^{(k)}|$;
$j = 0$;

Ensure: The set Ψ of four correct candidate coordinate functions

The Function $F(\Theta, j, V)$

$j \leftarrow j + 1$
for k is even, $k = 2$ to 16 **do**
  for $i = 1$ to $|\Theta_t^{(k)}|$ **do**
    $U_j = V_{j,i}^{(k)}$.
    if $2 \leq j < 4$ **then**
      Check if the vectors $U_1, \cdots, U_j$ pass the *balance filter*.
      if pass the *balance filter* **then**
        do $F(\Theta, j, V)$.
      else
        Continue.
      end if
    end if
    if $j = 4$ **then**
      Add $\{U_1, \cdots, U_j\}$ into the set Ψ.
    end if
  end for
end for
return The set Ψ of four correct candidate coordinate functions

first round has a stronger correlation with the low-weight mask at the input of the S-box layer in the last round. It was reasonable to assume that α_i was of weight one. According to this assumption, in order to determine the correct secret S-box, we should consider $2^4 \times 4! \approx 2^{8.6}$ possible equivalent S-boxes for each 4-bit S-box in the cipher Maya. Therefore, we need $(2^{8.6})^{16} \approx 2^{137.6}$ time complexity to recover all 16 secret S-boxes by exhaustive search. It is so large that the complexity is infeasible. In this paper, we focus on the settings of PRESENT-like cipher where the S-boxes are key-dependent and are repeated for the first and last rounds. We present a method of determining all 16 secret S-boxes with lower time complexity.

4.2 Determining the Secret S-box

In this section, we propose a technique to determine the secret S-boxes from the equivalent ones. Let $S_1, S_2, \cdots, S_N$ be secret S-boxes and $S_1^{-1}, S_2^{-1}, \cdots, S_N^{-1}$ be the inverse of these S-boxes. Without the loss of generality, we explain how to determine the leftmost S-box S_1. Assuming that we get m correct coordinate functions of secret S-box $U_j = (u_{j,0}, u_{j,1}, \cdots, u_{j,2^m-1}), V_j = (v_{j,0}, v_{j,1}, \cdots, v_{j,2^m-1})$,

$1 \le j \le m$, for m-bit secret S-boxes S_1 and S_1^{-1}, we denote $S_1(x) = P_1(f_1(x) \oplus k_1)$ and $S_1^{-1}(x) = Q_1^{-1}(g_1^{-1}(x) \oplus t_1)$, where $0 \le k_1, t_1 \le 2^m - 1$, P_1, Q_1 are m-bit permutations and f_1, g_1 are known functions based on U_j and V_j:

$$f_1(x) = \begin{pmatrix} 0 & 1 & \cdots & 2^m - 1 \\ u_{1,0}||\cdots||u_{m,0} & u_{1,1}||\cdots||u_{m,1} & \cdots & u_{1,2^m-1}||\cdots||u_{m,2^m-1} \end{pmatrix}$$

$$g_1(x) = \begin{pmatrix} 0 & 1 & \cdots & 2^m - 1 \\ v_{1,0}||\cdots||v_{m,0} & v_{1,1}||\cdots||v_{m,1} & \cdots & v_{1,2^m-1}||\cdots||v_{m,2^m-1} \end{pmatrix}$$

With the notations above, if we determine the value of k_1, P_1 or t_1, Q_1, we can recover the secret S-box S_1 (or S_1^{-1}) uniquely. We propose an efficient filtering method of determining the value of t_1, P_1, Q_1 as follows.

For every $0 \le x \le 2^m - 1$, it must hold that $P_1(f_1(x) \oplus k_1) = g_1(Q_1(x) \oplus t_1)$. Then we have

$$P_1 f_1(x) \oplus P_1 k_1 = g_1(Q_1(x) \oplus t_1) \quad and \quad P_1 f_1(0) \oplus P_1 k_1 = g_1(Q_1(0) \oplus t_1)$$

then we have

$$P_1 f_1(0) \oplus P_1 f_1(x) \oplus g_1(Q_1(0) \oplus t_1) \oplus g_1(Q_1(x) \oplus t_1) = 0$$

and then

$$P_1(f_1(0) \oplus f_1(x)) = g_1(Q_1(0) \oplus t_1) \oplus g_1(Q_1(x) \oplus t_1)$$

where f_1, g_1 are known functions, t_1 is an unknown m-bit constant and P_1, Q_1 are unknown m-bit permutation. In order to determine t_1, P_1, Q_1, we guess t_1, Q_1 by exhaustive search first. If the value of t_1 and Q_1 are correct, it must hold that P_1 is an m-bit permutation. By using this guess and determine method, the time complexity of determining one secret S-box can be reduced to $2^m \times m!$ from $(2^m)^2 \times (m!)^2$. To the cipher Maya, the complexity to determine one correct S-box is equal to $2^4 \times 4! \approx 2^{8.58}$.

In order to check the correctness of t_1 and Q_1, we introduce two effective filters as following definition.

Definition 2. Let $0 \le e_1, e_2 \le 2^m - 1$, $e_1 \ne e_2$. Assuming that $P_1(f_1(0) \oplus f_1(x)) = g_1(Q_1(0) \oplus t_1) \oplus g_1(Q_1(x) \oplus t_1)$, where f_1, g_1 are known functions, t_1 is an unknown m-bit constant and Q_1, P_1 are unknown m-bit permutation. If the value of t_1, Q_1 are correct, it holds that

$$\begin{aligned} &g_1(Q_1(f_1^{-1}(e_1 \oplus e_2 \oplus f_1(0))) \oplus t_1) \oplus g_1(Q_1(0) \oplus t_1) \\ &= g_1(Q_1(f_1^{-1}(e_1 \oplus f_1(0))) \oplus t_1) \oplus g_1(Q_1(f_1^{-1}(e_2 \oplus f_1(0))) \oplus t_1) \end{aligned}$$

we name this filter *linear filter*. Let $0 \le e_i \le 2^m - 1$ and $wt(e_i) = 1, 1 \le i \le r$, it holds that

$$wt\left(g_1(Q_1(f_1^{-1}(e_i \oplus f_1(0))) \oplus t_1) \oplus g_1(Q_1(0) \oplus t_1)\right) = 1$$

we name this filter *weight filter*.

The linear and weight filter can generalize to more than two elements e_1, e_2. In general, given r linearly independent $e_1, e_2, \cdots, e_r$, where $r \leq m$. For *linear filter*, assuming that $a_i \in \{0, 1\}$ and at least two of $a_1, a_2, \cdots, a_r$ are nonzero, then the linear combination $a_1e_1 \oplus a_2e_2 \oplus \cdots \oplus a_re_r$ can obtain one equation for checking the correctness of t_1, Q_1. There are $2^r - r - 1$ linear combinations based on $e_1, e_2, \cdots, e_r$, thus we have $2^r - r - 1$ equations for checking the correctness of t_1, Q_1. For *weight filter*, each $e_i, wt(e_i) = 1, 1 \leq i \leq r$ obtains one equation for checking the correctness of t_1, Q_1. Thus we have r equations for checking the correctness of t_1, Q_1. In fact, the *linear filter* is equivalent to P being linear, and *weight filter* is equivalent to $wt(P(e_i)) = 1$ with $wt(e_i) = 1$. In order to measure the effectiveness of *linear filter* and *weight filter*, we start with the necessary and sufficient condition of t_1, Q_1 passing these two filters as following theorem.

Theorem 1. *Let P be an m-bit bijective mapping. The necessary and sufficient condition of P being a m-bit permutation is that P is linear and for every e_i with $wt(e_i) = 1$, we have $wt(P(e_i)) = 1$.*

According to Theorem 1, the case of P being a m-bit permutation is equivalent to that of t_1, Q_1 passing the *linear filter* and *weight filter*. For an m-bit bijective mapping P, the probability of P being a bit-wise permutation is equal to $4!/16! \approx 2^{-39.67}$. Thus the probability of t_1, Q_1 passing the *linear filter* and *weight filter* is equal to $2^{-39.67}$. Moreover, the number of candidate t_1, Q_1 is equal to $2^4 \times 4! \approx 2^{8.58}$. It means that we can determine the correct t_1, Q_1 uniquely. Since we have known the correct t_1, Q_1, we can calculate P_1 uniquely. By using this method, we can determine the correct S-boxes one by one. The time complexity for recovering all 16 S-boxes can be reduced to $16 \times 2^4 \times 4! \approx 2^{12.58}$ from $2^{137.6}$.

5 Experiments

In this section, we apply our attack on PRESENT-like cipher in practice and estimate the success rate and complexity. Our experiment is based on 200 independent trials. In our experiment, let $\tau = 1, \xi = 2, t = 10$ (see Definition 1 and Algorithm 2).

In the sequel, we measure the data complexity in units which are equivalent to a known plaintext. The dominative time consumption of our attack is the time cost in the pruning search algorithm (Algorithm 3). Therefore, we measure the time complexity of our attack in units which are equivalent to an operation of checking if the vectors passing the *balance filter*.

In [7], Borghoff *et al.* pointed out that they carried out the attack using 2^{25} data complexity to 10 rounds Maya. However, the number of correct coordinate functions obtained and success rate were not presented in [7]. We did the same experiment as Borghoff *et al.* did on the 10 rounds Maya with 2^{22} to 2^{27} data complexity. Table 2 shows the success rate of finding at least one correct coordinate function of the secret S-box by using Algorithm 2 in this paper and using the method in [7]. Table 3 shows more details about the number of correct coordinate functions and the success rate by Borghoff's method.

Table 2. The success rate of finding at least one correct coordinate function of the secret S-box on 10 rounds Maya with different data complexity by using Algorithm 2 and Borghoff's method

Data complexity	2^{27}	2^{26}	2^{25}	2^{24}	2^{23}	2^{22}
Algorithm 2	100.0 %	100.0 %	99.5 %	94.0 %	55.0 %	18.5 %
Borghoff's method	91.5 %	88.0 %	71.0 %	40.5 %	23.0 %	3.5 %

Table 3. The number of correct coordinate functions and success rate with different data complexity on 10 rounds Maya in [7]

Number of correct coordinate functions	2^{27}	2^{26}	2^{25}	2^{24}	2^{23}	2^{22}
1	37.0 %	55.0 %	63.0 %	36.0 %	22.0 %	3.5 %
2	49.5 %	29.0 %	6.5 %	4.0 %	1.0 %	0.0 %
3	5.0 %	4.0 %	1.5 %	0.5 %	0.0 %	0.0 %
4	0.0 %	0.0 %	0.0 %	0.0 %	0.0 %	0.0 %

According to Table 2, the success rate is higher in our improved slender-set linear attack. It means that we can get more information about the secret S-boxes than Borghoff's attack. As can be seen from Table 3, more than two correct coordinate functions would be recovered by Borghoff's linear attack in very rare cases with 2^{22} data complexity and even though with 2^{27} data complexity. In order to recover a 4-bit secret S-box, it requires four linearly independent coordinate functions. It means that Borghoff's attack can not get enough information to recover the secret S-box.

However, after getting one or more correct coordinate functions, being different from Borghoff's attack, our attack can construct all four correct coordinate functions of secret S-box by using Algorithm 3. Assuming that we get one correct coordinate function of the secret S-box, the maximal time complexity of Algorithm 3 is equal to $(C_{15}^7)^3 \approx 2^{37.95}$. Table 4 shows the time consumption of Algorithm 3 based on 1 to 3 correct coordinate functions. Our experiment is based on 200 independent trials.

Table 4. The average time complexity of finding all four correct coordinate functions of secret S-box by using Algorithm 3 on 10 rounds Maya

Number of known correct coordinate functions	Time complexity	Cost on stander PC
1	$2^{17.78}$	≈7 min 12 s
2	$2^{14.74}$	≈53 s
3	$2^{9.35}$	≤1 s

From Table 4, we can find all four correct coordinate functions of secret S-box in less than a few minutes on a standard PC (at AMD Athlon 7750 Processor 2.7 GHz) after getting one or more correct coordinate functions. To 10 rounds Maya, we can recover all four correct coordinate functions of secret S-box with 2^{24} data and $2^{17.78}$ time complexity at success rate 94 % (see Tables 2 and 4). We apply our attack to 6 to 16 rounds Maya and estimate the data and time complexity and success rate (see Table 5).

Table 5. The data and time complexity and success rate of recovering all four correct coordinate functions on 6 to 16 rounds Maya cipher in this paper

Rounds	6	7	8	9	10	11	12	13	14	15	16
Data complexity	$2^{16.6}$	$2^{18.1}$	$2^{20.0}$	$2^{22.1}$	$2^{24.0}$	$2^{26.3}$	$2^{27.9}$	$2^{29.5}$	2^{31}	$2^{34.2}$	2^{36}
Time complexity	$2^{18.4}$	$2^{18.9}$	$2^{17.7}$	$2^{18.7}$	$2^{17.8}$	$2^{18.7}$	$2^{15.2}$	$2^{16.7}$	$2^{18.9}$	$2^{18.7}$	$2^{18.9}$
Success rate	90.5 %	87.5 %	88.0 %	91.5 %	94.0 %	89.5 %	90.0 %	93.0 %	91.5 %	88.5 %	87.5 %

From Table 5, we can see that our attack can break 16 rounds Maya with 2^{36} known plaintexts and $2^{18.9}$ time complexity at success rate 87.5 %. Our experiments suggest that 30-rounds Maya cipher can be break with approximately 2^{64} known plaintexts by using our attack.

6 Conclusion

In this paper, we present an improved slender-set linear attack to PRESENT-like cipher with secret S-boxes. We present three improvements comparing with Borghoff's attack. First, we use a new technique to support consistency of partitions of the input x to the secret S-boxes of the first S-box layer. Second, a new technique is proposed by making full use of the information from all the 240 original vectors together instead of the three longest vectors. Third, an effective filter for constructing correct coordinate functions of secret S-boxes by using pruning search algorithm is presented. These three techniques can help us to get the all four correct coordinate functions more efficiently. Finally, we focus on the settings where the secret S-boxes are key-dependent and are repeated for the first and last rounds. We propose a filter to determine the correct S-box from equivalent S-boxes with lower time complexity. We describe the practical attack to full round Maya, as detailed in Table 5. The experiments show that the correct S-box can be recovered with 2^{36} known plaintexts, $2^{18.9}$ time complexity and negligible memory complexity at a success rate of 87.5 % based on 200 independent trials. Our attack is the improvement and sequel of Borghoff's work, which is the best linear attack on PRESENT-like cipher with secret S-boxes up to now.

An interesting open question is to find a more efficient method to recover the correct coordinate functions of secret S-boxes. In fact, the approximation by Eq. (1) in Sect. 3.2 could bring too much noise. Furthermore, the theoretical model for complexity of recovering coordinate functions would be a possible direction of future work.

Acknowledgments. I wish to thank the anonymous reviewers for very useful comments that help to improve the paper. This work was supported by National Natural Science Foundation of China (Grant No. 61272488, No. 61272465).

References

1. Biham, E., Biryukov, A.: How to strengthen DES using existing hardware. In: Pieprzyk, J., Safavi-Naini, R. (eds.) Advances in Cryptology - ASIACRYPT 1994. LNCS, vol. 917, pp. 398–412. Springer, Berlin Heidelberg (1995)
2. Biryukov, A., Shamir, A.: Structural cryptanalysis of SASAS. In: Pfitzmann, B. (ed.) EUROCRYPT 2001. LNCS, vol. 2045, pp. 395–405. Springer, Heidelberg (2001)
3. Biryukov, A., Shamir, A.: Structural cryptanalysis of SASAS. J. Cryptol. **23**(4), 505–518 (2010). doi:10.1007/s00145-010-9062-1
4. Bogdanov, A.A., Knudsen, L.R., Leander, G., Paar, C., Poschmann, A., Robshaw, M., Seurin, Y., Vikkelsoe, C.: PRESENT: an ultra-lightweight block cipher. In: Paillier, P., Verbauwhede, I. (eds.) CHES 2007. LNCS, vol. 4727, pp. 450–466. Springer, Heidelberg (2007)
5. Borghoff, J., Knudsen, L.R., Leander, G., Matusiewicz, K.: Cryptanalysis of C2. In: Halevi, S. (ed.) CRYPTO 2009. LNCS, vol. 5677, pp. 250–266. Springer, Heidelberg (2009)
6. Borghoff, J., Knudsen, L.R., Leander, G., Thomsen, S.S.: Cryptanalysis of PRESENT-like ciphers with secret S-boxes. In: Joux, A. (ed.) FSE 2011. LNCS, vol. 6733, pp. 270–289. Springer, Heidelberg (2011)
7. Borghoff, J., Knudsen, L., Leander, G., Thomsen, S.: Slender-set differential cryptanalysis. J. Cryptol. **26**(1), 11–38 (2013). doi:10.1007/s00145-011-9111-4
8. De Cannière, C., Dunkelman, O., Knežević, M.: KATAN and KTANTAN — a family of small and efficient hardware-oriented block ciphers. In: Clavier, C., Gaj, K. (eds.) CHES 2009. LNCS, vol. 5747, pp. 272–288. Springer, Heidelberg (2009)
9. Cho, J.Y.: Linear cryptanalysis of reduced-round PRESENT. In: Pieprzyk, J. (ed.) CT-RSA 2010. LNCS, vol. 5985, pp. 302–317. Springer, Heidelberg (2010)
10. Gilbert, H., Chauvaud, P.: A chosen plaintext attack of the 16-round Khufu cryptosystem. In: Desmedt, Y.G. (ed.) CRYPTO 1994. LNCS, vol. 839, pp. 359–368. Springer, Heidelberg (1994)
11. Hong, D., Sung, J., Hong, S.H., Lim, J.-I., Lee, S.-J., Koo, B.-S., Lee, C.-H., Chang, D., Lee, J., Jeong, K., Kim, H., Kim, J.-S., Chee, S.: HIGHT: a new block cipher suitable for low-resource device. In: Goubin, L., Matsui, M. (eds.) CHES 2006. LNCS, vol. 4249, pp. 46–59. Springer, Heidelberg (2006)
12. Izadi, M., Sadeghiyan, B., Sadeghian, S., Khanooki, H.: MIBS: a new lightweight block cipher. In: Garay, J., Miyaji, A., Otsuka, A. (eds.) CANS 2009. LNCS, vol. 5888, pp. 334–348. Springer, Heidelberg (2009)
13. Leander, G., Paar, C., Poschmann, A., Schramm, K.: New lightweight DES variants. In: Biryukov, A. (ed.) FSE 2007. LNCS, vol. 4593, pp. 196–210. Springer, Heidelberg (2007)
14. Lim, C.H., Korkishko, T.: mCrypton – a lightweight block cipher for security of low-cost RFID tags and sensors. In: Song, J.-S., Kwon, T., Yung, M. (eds.) WISA 2005. LNCS, vol. 3786, pp. 243–258. Springer, Heidelberg (2006)
15. Mahadevan, G., Ruby, B.L.: Maya: a novel block encryption function. In: International Workshop on Coding and Cryptography. http://palms.princeton.edu/system/files/maya.pdf

16. 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)
17. Matsui, M.: Linear cryptanalysis method for DES cipher. In: Helleseth, T. (ed.) EUROCRYPT 1993. LNCS, vol. 765, pp. 386–397. Springer, Heidelberg (1994)
18. Peng, J., Jin, S.: Designing key-dependent S-boxes using hyperchaotic Chen system. In: Zhong, Z. (ed.) Proceedings of the International Conference on Information Engineering and Applications (IEA) 2012. LNEE, vol. 216, pp. 733–740. Springer, London (2013)
19. Schneier, B., Kelsey, J., Whiting, D., Wagner, D., Hall, C., Ferguson, N.: Twofish: a 128-bit block cipher. In: AES proposal (1998)
20. Standaert, F.-X., Piret, G., Gershenfeld, N., Quisquater, J.-J.: SEA: a scalable encryption algorithm for small embedded applications. In: Domingo-Ferrer, J., Posegga, J., Schreckling, D. (eds.) CARDIS 2006. LNCS, vol. 3928, pp. 222–236. Springer, Heidelberg (2006)
21. Stoianov, N.: One approach of using key-dependent S-BOXes in AES. In: Dziech, A., Czyżewski, A. (eds.) MCSS 2011. CCIS, vol. 149, pp. 317–323. Springer, Heidelberg (2011)
22. Szaban, M., Seredynski, F.: Dynamic cellular automata-based S-boxes. In: Moreno-Díaz, R., Pichler, F., Quesada-Arencibia, A. (eds.) EUROCAST 2011, Part I. LNCS, vol. 6927, pp. 184–191. Springer, Heidelberg (2012)
23. Vaudenay, S.: On the weak keys of blowfish. In: Gollmann, D. (ed.) Fast Software Encryption. LNCS, vol. 1039, pp. 27–32. Springer, Heidelberg (1996)

Cryptanalysis of KLEIN

Virginie Lallemand(✉) and María Naya-Plasencia

Inria, Rocquencourt, France
{virginie.lallemand,maria.naya_plasencia}@inria.fr

Abstract. Due to the recent emergence of resource-constrained devices, cryptographers are facing the problem of designing dedicated lightweight ciphers. KLEIN is one of the resulting primitives, proposed at RFIDSec in 2011 by Gong et al. This family of software-oriented block ciphers has an innovative structure, as it combines 4-bit Sboxes with the AES MixColumn transformation, and has woken up the attention of cryptanalysts. Several security analyses have been published, in particular on the 64-bit key version. The best of these results could attack up to 10 rounds out of the total number of 12. In this paper we propose a new family of attacks that can cryptanalyze for the first time all the 12 rounds of the complete version of KLEIN-64. Our attacks use truncated differential paths and are based both on some of the notions developed in previous attacks and on our new ideas that allow to considerably improve the performance. To prove the validity of our attacks, we have implemented reduced-round versions of them. In particular we were able to reproduce a practical attack that recovers the whole key on 10 rounds, which also corresponds to the best practical attack against KLEIN-64.

Keywords: KLEIN · Lightweight block cipher · Truncated differential cryptanalysis · MixColumn · Key-recovery

1 Introduction

Design of lightweight and secure primitives has become one of the major interests of the cryptographic community in order to answer the requirements of a large number of applications, like RFID and wireless sensor networks. Through these last years an enormous amount of promising such primitives has been proposed, like PRESENT [7], LED [12], Spongent [6], ARMADILLO [5], PRINCE [8], PRINTcipher [14], KLEIN [11], LBlock [23] and Twine [22]. Correctly evaluating the security of these proposals has become a primordial task that merits all the attention from the community. This has been proved by the big number of security analyses of the previous primitives that has appeared (to cite a few: [1,9,10,16,17,19,20]).

On the other hand, most of now-a-days cryptanalysis results are analyses of round-reduced versions that do not apply to, nor break the full primitive

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

C. Cid and C. Rechberger (Eds.): FSE 2014, LNCS 8540, pp. 451–470, 2015.
DOI: 10.1007/978-3-662-46706-0_23

studied. This trend could be explained by the recent improvements made by cryptographers in the last few years - partially resulting from competitions like AES, SHA-3 and eStream - that gave them solid bases and criteria to build secure designs. Nevertheless, it is still crucial to make security analyses in order to filter out the candidates that are not secure enough, and narrow down the list of available lightweight primitives to recommend the use of the secure ones.

KLEIN [11] is a lightweight block cipher proposed at RFIDSec2011 by Gong et al. Three versions were proposed for different key sizes: 64 bits, 80 bits and 96 bits, with 12, 16 and 20 rounds respectively. KLEIN is combining the AES `MixColumn` operations with 4-bit Sboxes. Since its publication, several cryptanalysts have been interested in its analysis and some results on round-reduced versions have been published [2–4,13,21,24]. So far, the highest number of attacked rounds was 10 in the 64-bit version (out of 12), 11 out of 16 in the 80-bit version and 13 out of 20 in the 96-bit version. Recently biclique analyses ([2,3]) appeared, but we can remark that this analyses require to perform an exhaustive search on the whole key and that the acceleration factors are very small.

We propose here a family of attacks that successfully exploits the slow diffusion between higher and lower nibbles in the cipher. They can be applied to the full 12-round KLEIN-64 with a time and data complexities of $2^{57.07}$ and $2^{54.5}$ respectively (other trade-offs are also possible). They apply to 13 and 14 rounds of KLEIN-80 and KLEIN-96 respectively, improving the previous attacks. We have also been able to implement some of our attacks, obtaining a practical key-recovery for 10 rounds of KLEIN-64. We recall that previous practical attacks reached at most 8 rounds. We have also been able to implement 8-round attacks with a considerably lower data complexity than previous ones and a faster execution.

The paper is organized as follows: the next two sections introduce KLEIN family of block ciphers (Sect. 2) and summarize the results of the security analyses that have been done so far and also recall some of their key ideas that are useful for our analysis (Sect. 3). Section 4 gives a generic description of our family of attacks and details the technical improvements we found to reduce the complexity. Section 5 is dedicated to the possible time-memory-data complexity trade-offs, and gives 4 examples of it while detailing the best time complexity attack we found on the 12-round version. Section 6 discusses and provides the results of the implementations we made to verify our attacks, and presents the 10-round practical attack on KLEIN-64. The paper ends with a conclusion.

2 Description of KLEIN

KLEIN is a family of lightweight block ciphers presented by Gong, Nikova and Law at RFIDSec2011. By design choice, it is implementation compact and has low-memory needs both in software and hardware, so it is a suitable family for resource-limited devices such as RFID tags and wireless sensors.

KLEIN encryption routine is a Substitution-Permutation Network that operates on 64-bit blocks. Three versions are proposed, denoted KLEIN-64, KLEIN-80 and KLEIN-96 with key-sizes of 64, 80 and 96 bits and 12, 16 and 20 rounds

Table 1. KLEIN 4×4 Sbox

x	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
S[x]	7	4	a	9	1	f	b	0	c	3	2	6	8	e	d	5

respectively. Each round is composed of 4 layers: `AddRoundKey`, `SubNibbles`, `RotateNibbles` and `MixNibbles`.

More precisely, the state entering a round is first xored with the round-key through `AddRoundKey`. The result is then divided in 16 4-bit parts or *nibbles* that are all transformed by the same involutive 4×4 Sbox represented in Table 1. KLEIN designers chose this Sbox instead of a byte-wise one to minimize implementation costs and memory needs.

Then, `RotateNibbles` rotates the state two bytes to the left and finally `MixNibbles` applies Rijndael `MixColumn` transformation to each half of the state. Note that this last step is byte-wise while the others can be seen as nibble-wise. Contrary to `Rijndael`, the step `MixNibbles` is not omitted in the last round. A final whitening key is added at the end of the process so the encryption routine requires one more key than the number of rounds. The round-keys are computed from the MasterKey with the KeySchedule algorithm that follows a Feistel scheme. A complete description of it can be found in [11].

3 Previous Cryptanalysis

Since KLEIN proposal in 2011, several cryptanalysis have been published, mostly on the KLEIN-64 version: this block cipher has been attacked with differential cryptanalysis methods up to 8 rounds [4,24] and an integral cryptanalysis was also proposed in [24] up to 7 rounds. All of these attacks reached the maximal number of rounds possible using their techniques. In [13] a study of the security of KLEIN is performed, and some ideas are proposed for trying to reach 9 rounds without success. The best previous attack on KLEIN-64 was made by Nikolić et al. in [21] and reached 10 rounds. This attack uses a new technique named the Parallel-Cut Meet-In-The-Middle (PC MITM) that can also apply to KLEIN-80 (up to 11 rounds) and to KLEIN-96 (up to 13 rounds). A summary of these results and of our best new ones is done in Table 2, where their complexities, number of rounds reached and version of the cipher are specified.

Most of the previous attacks took advantage of the almost full independence between higher and lower nibbles: either to build truncated differential paths ([4,24]) or to split the cipher in two independent subciphers ([21]). More precisely [4,21,24] pointed out the fact that all encryption layers, with the exception of `MixNibbles`, are nibble-wise and do not mix higher nibbles with lower nibbles. The two analyses [4,24] also gave an interesting property of the `MixColumn` structure, namely:

Proposition 1. *[4, 24] If the eight nibbles entering* `MixColumn` *are of the form* $0X0X0X0X$*, where the wild-card* X *represents any 4-bit value, then the output*

Table 2. Previous results and some of our new results on KLEIN.

Version of KLEIN	Source	Rounds	Data	Time	Memory	Attacks
KLEIN-64	[24]	7	$2^{34.3}$	$2^{45.5}$	2^{32}	Integral
	[24]	8	2^{32}	$2^{46.8}$	2^{16}	Truncated
	[4]	8	2^{35}	2^{35}	-	Differential
	[21]	10	1	2^{62}	2^{60}	PC MITM
	[3]	12	2^{39}	$2^{62.84}$	$2^{4.5}$	Biclique
	Sect. 5.1	12	$2^{54.5}$	$2^{57.07}$	2^{16}	Differential
KLEIN-80	[24]	8	$2^{34.3}$	$2^{77.5}$	2^{32}	Integral
	[21]	11	2	2^{74}	2^{74}	PC MITM
	Sect. 5.6	13	2^{52}	2^{76}	2^{16}	Differential
	[2]	16	2^{48}	2^{79}	2^{60}	Biclique
KLEIN-96	[21]	13	2	2^{94}	2^{82}	PC MITM
	Sect. 5.6	14	$2^{58.4}$	$2^{89.2}$	2^{16}	Differential
	[2]	20	2^{32}	$2^{95.18}$	2^{60}	Biclique

is of the same form if and only if the MSB of the 4 lower nibbles have all the same value. This case occurs with probability 2^{-3}.

Indeed, the higher nibble from the output of `MixColumn` belonging to the ith byte depends on: (a) the 4 higher nibbles from the input and on (b) the xor of the MSB of the input lower nibble from the ith byte and the MSB of the input lower nibble from the $i+1$th byte (mod 4). This information can be expressed as three quantities of one bit computed with the MSB of the input lower nibbles. This proposition allows to construct truncated differential paths with important probabilities leading to efficient distinguishers and key-recovery attacks up to 8 rounds, as the ones used in [4,13,24].

Also, as the pattern of the difference in the input of `MixColumn` is exactly the same as the one in the output, this truncated difference allows the attacker to build an iterative path. In the following, we denote by *iterative round* a round of KLEIN that goes from a difference with only lower nibbles active ($0X0X0X0X\ 0X0X0X0X$) to the same type of difference in the output. In [4] it was claimed that a difference with only lower nibbles active passes through an iterative round with probability $2^{-5.82}$, while in [24] it was computed as 2^{-6}. We will discuss the value of this probability in the next section.

An attentive study of the `KeySchedule` led to the following proposition:

Proposition 2. *[4,13,21,24] In the* `KeySchedule` *algorithm, lower nibbles and higher nibbles are not mixed: the lower nibbles of any round-key can be computed directly from the lower nibbles of the master key. The same property holds for higher nibbles.*

Note that in KLEIN-64 case, since each round key is as long as the master key, the lower nibbles of any round-key can be computed directly from the lower nibbles of any other round-key.

This proposition means that the keySchedule can be seen as two independent and parallel subroutines involving each half of the key bits.

Propositions 1 and 2 are the two main ideas used in [4,13,21,24], that we recycle and improve in our attacks.

4 Generic Description of Our Attacks

The best previous differential cryptanalyses of KLEIN basically exploited the iterative truncated differential path over R rounds for building attacks on $R+1$ rounds. No condition was imposed in the last round, making the whole output active.

Despite this, if we are given two ciphertexts (C, C') with a certain difference Δ_{out}, we can easily check if they verify the difference conditions at the output of round R (i.e., if they form a pair of values that might satisfy the differential path) without needing to guess any key-bits: we just have to apply to the output difference the linear transformation $\texttt{MixNibbles}^{-1}$, and check if the higher nibbles obtained are null, which will occur with a probability of 2^{-32}.

In previous cryptanalysis, since the number of rounds considered was relatively small, this big sieving of probability 2^{-32} appearing in the last round was sufficient to select only the pairs that conform the differential path. The set of possible key-bits involved in the differential path of the last round was then reduced to the ones that verify the conditions of round R, by only keeping the keys that generated higher nibble inputs with zero difference on each `MixColumn` from round R. Since this imposes $3 + 3 = 6$ bit-conditions as seen in Proposition 1, the number of possible key-bits involved is basically reduced by 2^{-6}.

In previous attacks several conforming pairs were produced (for example 6 in the 7-round attack) in order to iterate the filtering step of 2^{-6} and then recover the involved round-key bits. Contrary to those attacks which require to keep only the pairs that follow the differential path in the first step, we allow ourselves to conserve pairs that do not follow the differential path after the first filtering step. The pairs that verify the 32-bit conditions at round $R+1$ but do not satisfy the differential path are called *false alarms*. The idea is then to consider separately each *candidate* set made of one of these pairs and an associated possible part of the key (only the lower nibbles), i.e. a candidate triplet (C, C', k_{low}), and apply to it several filters to decide if the set is valid, that is if the pair is conforming the differential path while the considered part of the key is the correct one. Those several filters consist in using not only the conditions from the `MixNibbles` of round R but also the ones from the other rounds. Our sieves will then consist in checking the conformity of the candidates' differences going through all the successive `MixNibbles` operations, starting from the ciphertexts and inverting each round or starting from the plaintexts and considering the rounds in the forward direction. As we show in Sect. 4.3, reaching the next sieve requires to make a guess of 6 information bits. Since the sieve is of 2^{-6} in the case of iterative rounds, the number of remaining candidates is unchanged after the iterative filters, i.e., on average, for each candidate triplet (C, C', k_{low}),

only one candidate pair $(S, S')_r$ can be associated per iterative round. We will show how we can reduce the number of candidate triplets by first choosing a differential path with specific non-iterative first rounds, that provide a more powerfull filter, and second by comparing the informations obtained from the plaintexts/ciphertexts and from the filtering process.

The step for recovering the remaining key-bits is usually much cheaper.

In the following, we will explain in detail this attack in a generic way, including some technical improvements that have allowed us to reduce the complexities of the attack, and provide a result on the whole 12 rounds of KLEIN-64, the 64-bit key version, as well as much improved results on the other versions. We will start by studying in detail the properties and equations derived from `MixNibble`, showing how to correctly compute the probabilities of the path, the cost of inverting each round and of guessing the necessary key-bits. We will also provide the generic attack procedure with the corresponding complexities. Further, in Sect. 5, we present different possible trade-offs of this attack and the concrete results on the KLEIN ciphers, with one detailed example on the full 64-bit version that will also help the comprehension of the attacks.

4.1 `MixNibbles` Properties and Detailed Equations

Let us denote the binary decomposition of the byte a by $(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7)$, where a_0 is the MSB and a_7 the LSB.

Proposition 3. *The values of the lower nibbles outputting* `MixColumn` *depend on the values of the lower nibbles at the input and on 3 quantities computed from the higher nibbles that we will call 3 information bits. More precisely, to compute the lower nibbles resulting of the operation* `MixColumn(a,b,c,d)`*, the 3 information bits are:*

$$\begin{cases} a_0 + b_0 \\ b_0 + c_0 \\ c_0 + d_0 \end{cases}$$

In particular, $a_0 + b_0$ *is needed for computing the lower nibble at the first position,* $b_0 + c_0$ *the one in the second,* $c_0 + d_0$ *the one in the third and* $d_0 + a_0$*, which is the sum of the three of them, is needed for computing the lower nibble in the fourth position.*

When considering $\texttt{MixColumn}^{-1}\texttt{(a,b,c,d)}$*, a similar property holds for the computation of the output lower nibbles, which requires the following 3 information bits in addition to the input lower nibbles:*

$$\begin{cases} a_1 + a_2 + b_2 + c_0 + c_1 + c_2 + d_0 + d_2 \\ a_1 + b_0 + b_1 + c_1 + d_0 + d_1 \\ a_0 + a_1 + a_2 + b_0 + b_2 + c_1 + c_2 + d_2 \end{cases}$$

Similarly and consequently with Proposition 1, the values of the higher nibbles outputting `MixColumn` *depend on the values of the higher nibbles at the input and*

on 3 quantities computed from the lower nibbles that we will call 3 information bits. More precisely, to compute the higher nibbles resulting of the operation `MixColumn(a,b,c,d)`*, the 3 information bits are:*

$$\begin{cases} a_4 + b_4 \\ b_4 + c_4 \\ c_4 + d_4 \end{cases}$$

When considering $\texttt{MixColumn}^{-1}\texttt{(a,b,c,d)}$*, a similar property holds for the computation of the output higher nibbles, which requires the following 3 information bits in addition to the input higher nibbles:*

$$\begin{cases} a_5 + a_6 + b_6 + c_4 + c_5 + c_6 + d_4 + d_6 \\ a_5 + b_4 + b_5 + c_5 + d_4 + d_5 \\ a_4 + a_5 + a_6 + b_4 + b_6 + c_5 + c_6 + d_6 \end{cases} \quad (1)$$

The proof can be found in the full version of this article [15]. This proposition implies that an attacker that knows the values of the input lower (respectively higher) nibbles, has to guess 6 bits only to compute `MixNibbles` (or $\texttt{MixNibbles}^{-1}$) in its lower (respectively higher) nibbles. Since `MixColumn` is linear, Proposition 3 applies both to values and to differences. This proposition will be used in our attacks for two different things: computing the correct probabilities of each round, as well as showing how to invert the rounds.

4.2 Probability of One Iterative Round and Other Initial Rounds

As previously said, the probability of one iterative round was correctly pointed out in [24] as 2^{-6}. In [4] the given probability was $2^{-5.82}$. In fact, if no condition is imposed on the input of one iterative round, the probability of verifying it is indeed 2^{-6}. We have been able to implement and verify this: we have found a probability extremely close to 2^{-6} per iterative round with no special condition on the input. As 2^{-6} can be seen as a close lower bound, we will consider this probability for the iterative rounds without conditions as it is the worst scenario for the attacker.

The difference in the probability computation can be introduced when, because of a specific form of the differential path and the branch number of `MixColumn` from previous rounds, some input nibbles are active or inactive for sure. This will change the probability due to the Sbox: as already pointed out in [4], when a nibble is active with probability one, its probability of outputting a pair of values with a difference having its MSB to 0 is 7/15, but the probability of having its MSB to 1 is 8/15. We have correctly taken this into account and conducted new computations and experiments. All the configurations that we will use in our analysis are represented in Fig. 2, so we use this figure as reference. We have obtained the following results for the rounds 1,2,3 and 4 represented in the figure:

– For round 1, as pointed out in [24], there exists only one possible difference with the 4 active lower nibbles that outputs after `MixColumn` a difference

with only one lower nibble active and no difference in the higher nibbles: $(0d0b0e09)$. The probability of this round is then 2^{-16}, as we want to obtain a fixed difference after `SubNibbles`.
- For round 2 we know for sure that we have only one active `MixColumn`, and in its input, we have only one active nibble, that will be active with probability one. This means that the probability of verifying this round is $(\frac{7}{15}) = 2^{-1.1}$.
- For round 3 we have both `MixColumn` active, each with exactly two active nibbles in its input. The probability for this round is $(\frac{7}{15})^4 = 2^{-4.4}$, as the MSB of all the 4 active nibbles need to be 0.
- For round 4 the track of the influence of the branch number starts to be hard to follow. We have then performed experiments, and we could verify that the probability of this step is extremely close to 2^{-6}. Though a bit higher, for the sake of simplicity and for considering the worst case for the attacker, we consider it to be 2^{-6} (so the same case as iterative rounds).

4.3 Cost of Inverting One Round

By *inverting one iterative round* we consider the situation where a pair of values for the lower nibbles of the output is given and we want to obtain the possible pairs of values for the lower nibble inputs of that round. The lower nibbles of the corresponding round-key are supposed known.

To invert round r we are given a candidate (C, C', k_{low}), and its associated candidate pair at round r: $(S, S')_r$. At the end, we obtain $(S, S')_{r-1}$. As the lower nibbles of the key are known, we will omit `AddRoundKey` from the explanation, as we only work on the lower nibbles of the state and it can consequently be seen as transparent. The cost of inverting one round by using the naive way of guessing the 6 additional information bits (3 for each $\texttt{MixColumn}^{-1}$ as explained in Proposition 3) all at once, and next discarding the ones that do not verify the conditions of the previous round is of 2^6. We propose here a procedure for inverting one round with a cost of 2^4 instead of 2^6.

Improved Cost of Inverting One Round. We first consider the case of inverting one iterative round. We will later see what happens when the round is not iterative. First, we make the 3-bit guess from the right part of Fig. 1 to compute $\texttt{MixColumn}^{-1}$, $\texttt{RotateNibbles}^{-1}$ and $\texttt{SubNibbles}^{-1}$ from steps 1, 2 and 3 on half of the known state of $(S, S')_r$. Then we have obtained the pair of values of the two most-left and of the two most-right lower nibbles from $(S, S')_{r-1}$, and with them we can compute the nibble differences a, b, c', d' at the output of `MixColumn` from step 4 in the figure. With the notations from Sect. 4.1, for each of the 2^3 values of the information bits that we have tried, we compute the following six bits in this exact order: $(a_5 + a_6 + b_6, a_5 + b_4 + b_5, a_4 + a_5 + a_6 + b_4 + b_6, c'_4 + c'_5 + c'_6 + d'_4 + d'_6, c'_5 + d'_4 + d'_5, c'_5 + c'_6 + d'_6)$. It is easy to verify that these quantities correspond each to one half of the last three equations from Proposition 3, equations that need to be verified in order to have the higher nibbles inactive in the input of `MixColumn`. The first three quantities correspond

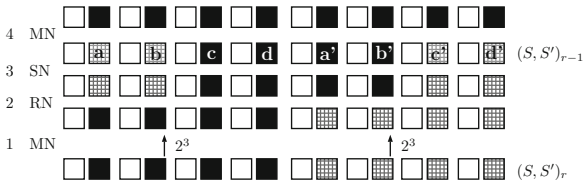


Fig. 1. Detail of inverting one round.

to the part of the equations associated to the left `MixColumn` from Fig. 1, and the next three, to the right one.

We can store all of these 6-bit values in a list of size 2^3. Next, we perform a guess of the 3 bits needed for inverting the other half of $(S, S')_r$, and we can compute the nibble differences a', b', c, d from the figure. For each one of the 2^3 values that we try, we compute the following 6 bits, in this order: $(c_4 + c_5 + c_6 + d_4 + d_6, c_5 + d_4 + d_5, c_5 + c_6 + d_6, a'_5 + a'_6 + b'_6, a'_5 + b'_4 + b'_5, a'_4 + a'_5 + a'_6 + b'_4 + b'_6)$ and check if there is a match in the precomputed list, as we know that, for any $\texttt{MixColumn}^{-1}$ with no higher nibbles active in the input nor the output, the three expressions from Eq. (1) must be equal to 0. The associated values of the match will represent a value for a pair candidate at round $r - 1$, where $\Delta(S, S')_{r-1} = (a, b, c, d, a', b', c', d')$. A match occurs with probability of 2^{-6}, as we want to collide on 6 bits. As we have $2^3 \times 2^3$ pairs, on average we can expect only one value to stay, so only one value on average for $(S, S')_{r-1}$ per candidate triplet as previously announced, so inverting one iterative round does not increase the number of kept candidates. The cost of this step, given by applying the instant matching algorithm from [18] is about $2^3 + 2^3 = 2^4$.

If the round we want to invert is not iterative, there usually exist more difference conditions to be verified on the input, as in addition to the previous six bit conditions, some more differences might have been fixed, like for example non-active lower input Sboxes (in the non-iterative rounds we have considered in our study, this is always the case). The cost of inverting is exactly the same (2^4), but since there are more equations to verify, the probability that a candidate pass will be smaller than 2^{-6}: fewer candidates will be kept. The number of candidates kept is even smaller when we invert the last round since we have to match both differences and values of the computed half-states.

4.4 Guessing the Necessary Key-Bits

To perform the attack, we consider the pairs of ciphertexts (C, C') that verify the last round condition on 32 bits and we need to guess the values of the lower nibbles of the key (k_{low}) that verify as well the first round conditions. Then, we are able to compute the values and differences before the first `MixNibbles`, place where we will stop inverting and where we will do the match of values and differences recovered in the other direction. Proposition 2 shows us that a $|k|/2$ bit guess (where $|k|$ is the length of the key) of the lower nibbles of the key is sufficient to obtain all the lower nibbles of all the round-keys.

We use a divide-and-conquer approach to guess the bits from k_{low} and we are then able to build the candidates (C, C', k_{low}). This technique is similar to the one proposed for inverting one round. Instead of guessing all the lower nibbles of one key at the same time, we can perform this guess in an ingenious way that allows to reduce the computational cost.

We consider the first round of the path (as conditions before `MixNibbles` can be tested here without needing to guess any extra bits). From the plaintexts (from which we obtained the ciphertexts C, C') we first consider 4 lower nibbles that will be the inputs to the same `MixColumn`, and we guess the sixteen bits from k_{low} that are xored to this 4 nibbles in the first round. We next compute the inputs of the corresponding `MixColumn` and we only keep the keys that fulfill the needed conditions to verify round 1. These conditions will depend on the specific path considered. For example, in Fig. 2, the conditions of the right `MixColumn` from the first round are verified with probability 2^{-16}, as only one difference can verify them (as explained previously), which means that among the 2^{16} keybits tried for this half, only one value will verify this part of the path. We repeat the procedure with the other 4 input lower nibbles and the other 16 key-bits. We combine both partial solutions and we obtain all the 2^{32+p_1} possible candidate triplets derived from (C, C'), where 2^{p_1} is the probability of verifying the first round of the path.

The cost in number of encryptions of recovering the candidate triplets for each pair of ciphertexts considered is then $2^{16} + 2^{16} + 2^{32+p_1} \times 2^{|k_{low}|-32}$, as in the first round only 32 bits from the lower nibbles intervene, and for realizing the whole attack we have to guess the remaining ones (for KLEIN-64 the last factor is 1 as $|k_{low}| = 32$).

4.5 Generic Description of the Attack Procedure and Complexity

First, we choose a truncated differential path over the desired number of rounds: several choices are possible, varying with the considered differences in the first three rounds of the path and leading to different time-memory-data trade-offs, as we will see in Sect. 5. Note that the size of the truncated difference entering the first round determines the size of the structures that we can build with the input plaintexts: if the size is Δ_{in} bits, we can build about $2^{2\Delta_{in}-1}$ pairs with $2^{\Delta_{in}}$ inputs. In the following 2^p represents the probability of the whole path, R is the number of rounds of the differential path and $R+1$ is the number of rounds that we will attack. We denote by 2^{p_1}, 2^{p_2} and 2^{p_3} the probabilities of the first 3 rounds respectively, $p_1, p_2, p_3 \leq 0$. Since one iterative round has a probability of 2^{-6} of being verified and because of the forms of the considered paths, we have $p = p_1 + p_2 + p_3 - 6 \times (R-3)$.

1. Obtaining enough data: With the use of structures, we generate a certain number of ciphertexts such that we obtain enough pairs to be ensured to get one that verify our differential path: to obtain the required 2^{-p} pairs, we encrypt $\frac{2^{-p}}{2^{2\Delta_{in}-1}} 2^{\Delta_{in}}$ plaintexts so we perform $\frac{2^{-p}}{2^{2\Delta_{in}-1}} 2^{\Delta_{in}}$ encryptions.

2. Last-round filter: At this point, we can discard some pairs that for sure do not verify the differential path. As detailed in [4,24], by inverting the output difference through the last `MixColumn` we can observe the value of the difference entering this transformation and then discard the ones that do not have the higher nibbles inactive. In practice, we construct a sorted list of all the higher nibbles values obtained by inverting `MixColumn` from the ciphertexts of a same structure (without considering the key addition) and look for collisions. Such a collision occurs with probability 2^{-32} so there are 2^{-p-32} remaining pairs of plaintexts.
3. Guess the involved key-bits: For each pair of plaintexts and their associated ciphertexts that collide at the previous step, we make two 16-bit guesses as explained in Sect. 4.4 and obtain possible values of the first 8 lower nibbles of the key. Since the conformity with round 1 is of probability 2^{p_1} it gives us $2^{-p-32+32+p_1}$ candidates formed by a pair of plaintexts and the 8 first lower nibbles of the master key. If the version attacked is KLEIN-64, the 8 lower nibbles correspond to the lower nibbles of the whole key but for KLEIN-80 and KLEIN-96, we have to make additional guesses to obtain all the possible lower nibble values. For KLEIN-64, KLEIN-80 and KLEIN-96 we obtain respectively $2^{-p-32+32+p_1}$, $2^{-p-32+32+p_1+8}$ and $2^{-p-32+32+p_1+16}$ possible candidates (C, C', k_{low}). This step requires $2 \times \frac{1}{12} \times 2^{16}$ encryptions, and allows us to compute the associated pair of half-states (associated to each candidate) at the input of the first `MixNibbles` that already satisfies the conditions from round 1. We will denote this pair of half-states by $(S, S')_1^*$.
4. Inverting the rounds: At this point we start inverting the rounds from the candidates that we have obtained, generating possible pairs $(S, S')_r$ for r from R to 1. That step requires 2^4 round encryptions per inversion and per triplet, as detailed in Sect. 4.3. During the iterative rounds, the number of possible triplets stays the same, contrary to what happens during the non-iterative rounds where the number of candidates is reduced (see Sect. 4.3). The attack is performed one triplet at a time. Once we have computed $(S, S')_1$, we have to guess the 6 bits needed to invert the first `MixNibbles`, and next we have to match values and active differences with the already computed values $(S, S')_1^*$.

   In the differential paths that we will consider, like the one represented in Fig. 2, if we denote by 2^q the filtering probability obtained when inverting rounds 2, 3 and 4 (in some cases round 4 or even 3 adds just a filter of 2^{-6}, but to be general we include it in the special case), the total number of remaining candidates in the end is $2^{-p-32+32+p_1+6\times4+q}$, $2^{-p-32+32+p_1+8+6\times4+q}$ and $2^{-p-32+32+p_1+16+6\times4+q}$ respectively for KLEIN-64, KLEIN-80 and KLEIN-96. If the number of remaining candidates is smaller than $2^{|k_{low}|}$, as there is one possible value for k_{low} per candidate, the cost of recovering the key is smaller than the one of exhaustive search. In practice, after inverting all the rounds, the number of remaining candidates is currently very small.

   The cost of this step is given by the initial candidate triplets $2^{-p-32+32+p_1}$ multiplied by 2^4 (cost of inverting), multiplied by the number of inverted

rounds and by the relative cost to one encryption of each inverted round. In the next section we will see an illustrative and detailed example of this computation.

5. Recovering the whole key: Finally, we have to recover the higher nibbles of the key, which can be done by an exhaustive search or better (as it is explained in Sect. 4.6), and we have recovered the whole key.

4.6 Higher Nibbles Recovery

To recover the complete key, we can perform a more efficient attack than the exhaustive search for the remaining key-bits, by deducing information from the 6-bit guesses associated to the candidates that remain after the sieving process. One more time, each candidate that passes the sieving process will be studied separately; we make the hypothesis that the candidate is valid so the 6-bit guesses give us the values that must take certain combinations of intermediate states.

For instance, at the first round, we know the value that must be taken by the sum of the MSB of the higher nibbles of the state entering the two first `MixColumn`. For each possible value of the higher nibbles of the key, we will compute that state from the plaintext and check this 6-bit condition. The keys that pass the test will undergo the second sieve resulting from the second round information bits and so on.

To limit the number of encryptions required to complete this step, we can one more time use the independences between the 2 half-states during the `MixNibbles` operation. We first make a guess on the 16 higher nibbles of the master key that are added to the 32 middle bits of the state at the first round and that impact the value of the 32 bits at the output of the left `MixColumn`. We then check the 3 corresponding information bits and realise the same operations for the 16 key bits required to compute the right `MixColumn`. Since 2^{16} half-state encryptions are required for each half, the first round filter requires $2^{16} \times \frac{1}{2} \times 2$ round encryptions.

Next, as previously said, each round gives a 6-bit filter for the higher nibbles of the key. For example, for KLEIN-64 the total complexity of the higher nibbles recovery is of $(2^{16} + 2^{26} + 2^{20} + 2^{14} + 2^8 + 2^2) \times \frac{1}{12} = 2^{22.4}$ complete encryptions. This procedure needs to be repeated for each candidate recovered during the k_{low} search.

5 Different Time-Memory-Data Trade-Offs

Various differential truncated paths are possible, each one leading to different time-memory-data trade-offs. We have studied many different possibilities and we present here the 4 cases that have provided better results. The only difference between them is the shape of the wanted differences in the first three rounds of the path. We will explain one of them applied to the full 12 rounds of KLEIN-64 in a detailed way as an illustration of the attacks, and next present the other cases considered, also with results given on the 64-bit key version. In the next

section we will provide the results obtained with some considered cases in all the versions of the cipher, for several number of rounds up to the highest number that could be reached (which is 12, 13 and 14 for the 64-, 80-, and 96-bit versions respectively).

We recall here that the probability values that we will use are well detailed and explained in Sect. 4.2.

5.1 Case I

In this case, we consider a truncated differential path that corresponds to the one in [24]. This path is depicted in Fig. 2, including the details of our attack. This is the application that will provide us the best time complexity attack on the full 12 rounds of KLEIN-64.

1. Obtaining enough data: Using Fig. 2 it is easy to verify that the differential path has a probability of $p = 2^{-16-1.1-4.4-6\times 8} = 2^{-69.5}$ and that $\Delta_{in} = 16$. Here, the probabilities correspond to the ones discussed in Sect. 4.2. We need to generate pairs such that we can expect with a good probability that one among them will verify the whole path. Since one structure allows us to build 2^{16} plaintexts, which lead to $\frac{2^{16}\times(2^{16}-1)}{2} \approx 2^{31}$ pairs, we have to build $2^{69.5-31} = 2^{38.5}$ structures to have among all the pairs one verifying the whole path. This step then requires $2^{38.5} \times 2^{16} = 2^{54.5}$ full encryptions, which correspond to the data complexity of the attack. As it is smaller than 2^{64}, the whole codebook, we can obtain such an amount of plaintexts. Since the true conforming pair is necessarily composed of 2 plaintexts from the same structure, one structure can be treated after the other so the required memory is of 2^{16} plaintexts.
2. Last-round filter: we compare the values obtained when inverting the last `MixNibbles` on each ciphertext and keep the pairs that have a difference with inactive higher nibbles at this point. There remain $2^{69.5-32} = 2^{37.5}$ candidate pairs of plaintext. The cost of this step is negligible.
3. Guess the involved key-bits: At this point, we perform the optimized guess of the lower nibbles of the key, k_{low}, as seen in Sect. 4.4, so we obtain $2^{32-16} \times 2^{37.5} = 2^{53.5}$ candidates (C, C', k_{low}) with $(2^{16} \times \frac{1}{2} \times \frac{1}{12}) \times 2$ encryptions. So far, we have 2^{16} possible keys k_{low} for each candidate pair (C, C'). For each candidate set, we know the values and differences at the input of `MixNibbles` of the first round, $(S, S')^*_1$, and these values already verify the conditions imposed through the first `MixNibbles`.
4. Inverting the rounds: Each time that we obtain one of the $2^{53.5}$ candidates, we start inverting rounds and generating the candidate pairs from $(S, S')_{11}$ to $(S, S')_4$ (so we invert 8 rounds). We can see in Fig. 2 that for all of these rounds, the amount of bits guessed compensates the probability of verifying the path, and we expect to obtain one pair candidate per round and per candidate set. As we can see in the figure, the remaining probabilities for inverting rounds 4, 3, 2 and the `MixNibbles` transform from round 1 (so to arrive to the previously computed values $(S, S')^*_1$) is $2^q = 2^{-20-13-36} = 2^{-69}$.

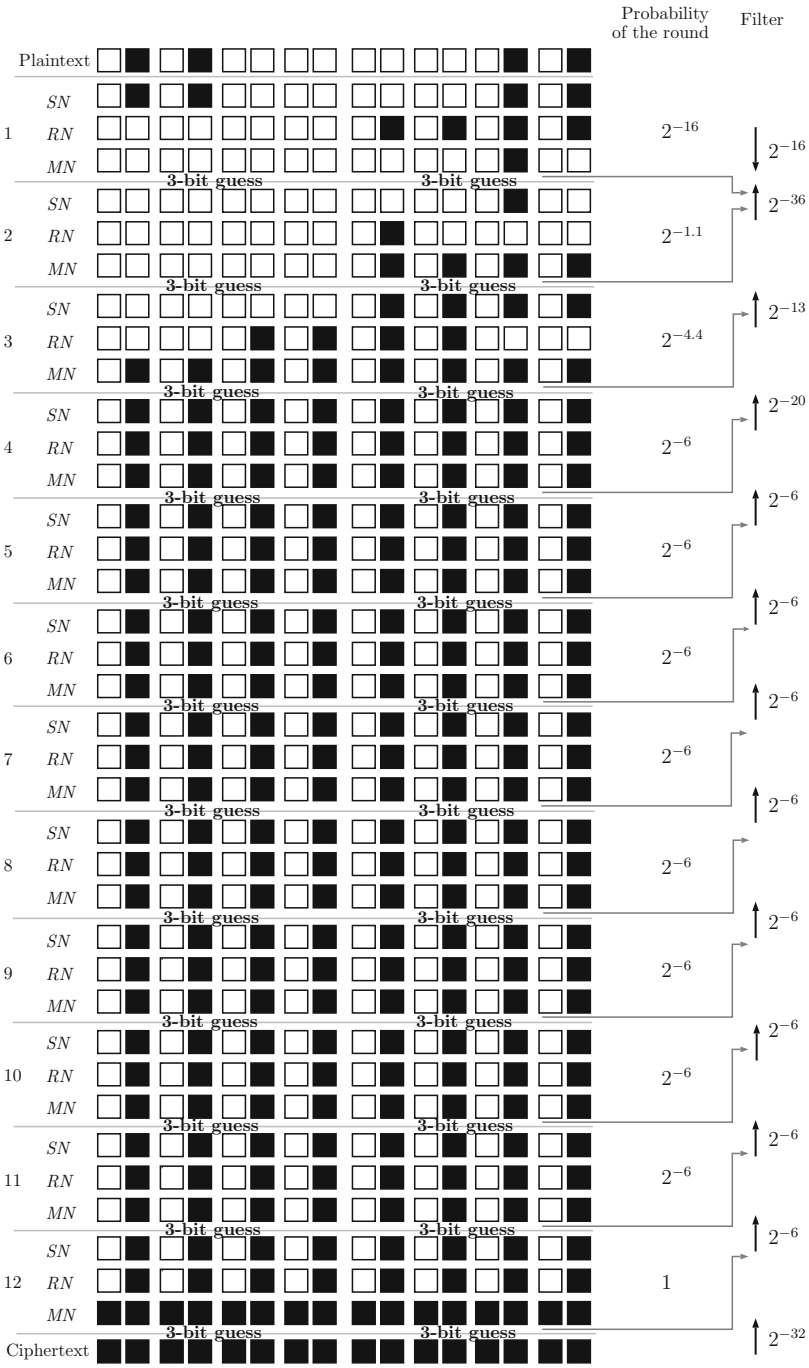


Fig. 2. Attack on 12 rounds of KLEIN-64 using case I

Indeed, the filter probability at the end of round 3 is of $2^{-(4\times4)-4} = 2^{-20}$, since we need to have 4 lower nibbles inactive and two MSB of the 4 active lower nibbles to 0 and the input of `MixNibbles` as depicted in Fig. 2. The probability of the filter at the end of round two is of $2^{-13} = 2^{-(3\times4)-1}$ because of the three inactive lower nibbles at the input of the active `MixColumn` and the MSB of the active one, and the filter probability at the end of the first round is of $2^{-36} = 2^{-32-4}$ as we have to collide with the previously forward computed 32 bits of values and 4 bits of differences.

If we apply the formula from Sect. 4.5, we can compute the number of remaining candidate triplets as $2^{69.5-32+32-16+6\times4-69} = 2^{8.5}$, meaning also that we recover only $2^{8.5}$ possibilities for the 32 bits in k_{low}. The term $2^{6\times4}$ comes from the fact that we have to guess the 6 bits for inverting 4 rounds, namely rounds 4, 3, 2 and also 1, as we want to match the values in $(S, S')^*_1$.

The cost of this step is $2^{69.5-32+32-16} \times 2 \times \frac{1}{2} \times \frac{8}{12} \times (2^3 + 2^3) \simeq 2^{56.9}$ encryptions for the inversion of the first 8 rounds. And for inverting the remaining 4 rounds, we have a complexity of $2^{53.5} \times \frac{1}{12} \times (2^3+2^3) + 2^{53.5+6-20} \times \frac{1}{12} \times (2^3 + 2^3) + 2^{39.5+6-13} \times \frac{1}{12} \times (2^3 + 2^3) + 2^{32.5+6} \times \frac{1}{12} \times (2^3 + 2^3) \simeq 2^{53.9}$.

This part of the attack will be the bottleneck of the total time complexity: $2^{56.9} + 2^{53.9} = 2^{57.07}$.

5. Recovering the whole key: Finally, we recover the higher nibbles with the process explained in Sect. 4.6. The cost of this step is $2^{8.5} \times 2^{22.4}$, so the bottleneck in terms of time complexity for recovering the whole key is the one of the previous step.

This version requires a total of $2^{57.07}$ encryptions, $2^{54.5}$ data and 2^{16} memory.

5.2 Case II

We use a truncated differential path associated to the path given in [4] (the value of the input difference is not fixed). As there is only one active nibble at the beginning of the path, the structures will be the smallest ones that we will use (size of $\Delta_{in} = 4$), and the memory will be very small, while the data complexity will be bigger than in other cases.

For the case of 12 rounds of KLEIN-64, we have a probability for the path of $2^{-1.1-4.4-6\times9} = 2^{-59.5}$. The amount of data needed is $2^{59.5-7+4} = 2^{56.5}$, and the memory needed is 2^4. The bottleneck in the time complexity is given by $2^{59.5-32+32-1.1} \times 2 \times \frac{1}{2} \times \frac{9}{12} \times (2^3 + 2^3) \simeq 2^{61.98}$. The number of remaining candidates is $2^{58.39+6-16-4+6-12-1+6-4-32} = 2^{7.39}$. The time for recovering the remaining bits of the key is $2^{7.39} \times 2^{22.4} = 2^{29.79}$.

5.3 Case III

This attack uses a path with an iterative round at every round, and Δ_{in} is consequently 32. This attack has low data complexity, but the highest memory, and not very good time complexity in the case of an attack on 12-round KLEIN-64.

For the scenario on 12 rounds of KLEIN-64, the time complexity is very close to the one of the exhaustive search. We have a probability for the path of $2^{-6\times 11} = 2^{-66}$. The amount of data needed is $2^{3+32} = 2^{35}$, and the memory needed is 2^{32}. The cost of the time bottleneck is given by $2^{66-32+32-6} \times 2 \times \frac{1}{2} \times \frac{11}{12} \times (2^3 + 2^3) = 2^{63.87}$. The number of remaining candidates is $2^{60+6-64} = 2^2$. The time for recovering the remaining bits of the key is $2^2 \times 2^{22.4} = 2^{24.4}$.

5.4 Case IV

In this case, we use a truncated path that starts with a difference of the form $(0X0X000000000X0X)$, when represented as 16 nibbles. This one has the same time complexity than the second one, better data and worse memory.

For the case of 12 rounds of KLEIN-64, we have a probability for the path of $2^{-3-4-6\times 9} = 2^{-61}$. The amount of data needed is $2^{16+30} = 2^{46}$, and the memory needed is 2^{16}. The bottleneck in the time complexity is given by $2^{61-32+32-3} \times 2 \times \frac{1}{2} \times \frac{9}{12} \times (2^3+2^3) = 2^{61.57}$. The number of remaining candidates is $2^{61-3+6\times 3-20-48} = 2^8$. The time for recovering the remaining bits of the key is $2^8 \times 2^{22.4} = 2^{30.4}$.

5.5 Results on KLEIN-64

The complexity of the 4 different trade-offs presented here are summarized in Table 3 and depicted on Fig. 3. Table 4 provides the results obtained on KLEIN-64 using case I for different numbers of rounds.

5.6 Results on KLEIN-80 and KLEIN-96

For KLEIN-80 and 96 we provide in Table 5 the obtained results for several numbers of rounds using different cases. Case I does not reach a lot of rounds, as

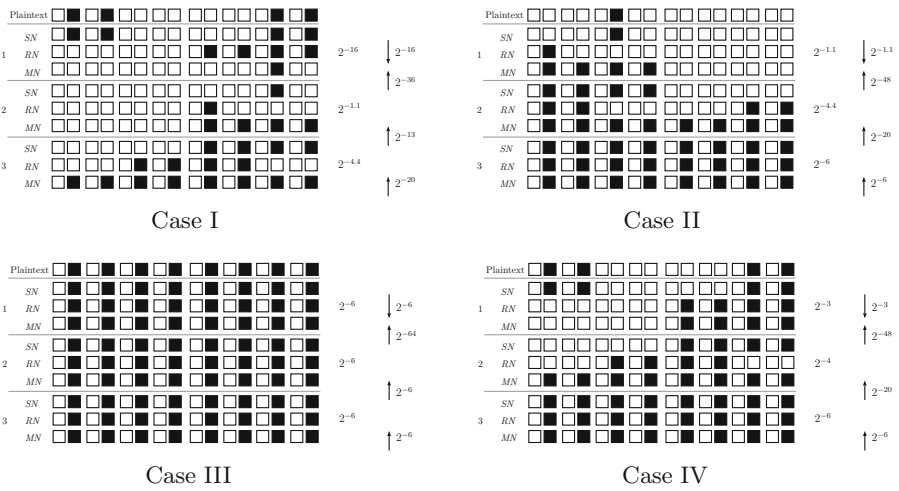


Fig. 3. Beginnings of the truncated differential paths of our 4 trade-offs

Table 3. Summary of the probabilities and complexities of our 4 trade-offs

Case	p1	p2	p3	Data	Time	Memory
I	-16	-1.1	-4.4	$2^{54.5}$	2^{57}	2^{16}
II	-1.1	-4.4	-6	$2^{56.5}$	2^{62}	2^{4}
III	-6	-6	-6	2^{35}	$2^{63.9}$	2^{32}
IV	-3	-4	-6	2^{46}	$2^{61.6}$	2^{16}

Table 4. Best time complexities for recovering the whole key for several round-reduced variants of KLEIN-64

Rounds	Data	Time	Memory
8	$2^{30.5}$	$2^{31.7}$	2^{16}
9	$2^{36.5}$	2^{38}	2^{16}
10	$2^{42.5}$	$2^{44.4}$	2^{16}
11	$2^{48.5}$	$2^{50.6}$	2^{16}
12	$2^{54.5}$	$2^{57.07}$	2^{16}

the data complexity exceeds 2^{64}, which is the maximal amount of data available, after 13 rounds.

6 Implementation and Verification

We have experimentally verified the efficiency of the proposed attacks by implementing some variants in C language. We wrote our own implementation of KLEIN-64 with look-up tables for `MixColumn` and its inverse and we verified it with the test vectors given in KLEIN specifications [11].

We then implemented the attack exactly as described in Sect. 5: the complete key is recovered in 2 steps with first the search for the lower nibbles with a truncated differential and then the search for the higher nibbles with an improved exhaustive search.

In particular, we have been able to implement the first successful practical attack on KLEIN-64 up to 10 rounds. For this, we have considered case I, the one having the smallest time complexity and average data and memory needs. We used several speed-optimization flags and a computer with an Intel(R) Xeon(R) CPU W3670 at 3.20 GHz (12 MB cache), and with 8 GB of RAM. Our program shows that the proposed attack works and recovers the correct key.

Below we report the outputs of our program for an attack on 10 rounds. The field `structure` refers to the randomly chosen values at the beginning, i.e. the 12 nibbles common to all the plaintexts so that the differences between 2 plaintexts are only in the first and last 2 lower nibbles. *Plaintext 1* and *Plaintext 2* form the conforming pair found that enabled us to determine the lower nibbles of the key. Once the lower nibbles are found, the higher ones are recovered in a few seconds.

Table 5. Best complexities for recovering the whole key for several round-reduced variants of KLEIN-80 and −96

Version	Case	Rounds	Data	Time	Memory
80	I	12	$2^{54.5}$	2^{65}	2^{16}
80	I	13	$2^{60.5}$	$2^{71.1}$	2^{16}
80	II	13	$2^{62.5}$	2^{76}	2^{4}
80	III	13	2^{41}	2^{78}	2^{32}
80	IV	13	2^{52}	2^{76}	2^{16}
96	III	14	2^{47}	2^{92}	2^{32}
96	IV	14	$2^{58.4}$	$2^{89.2}$	2^{16}

```
NB rounds: 10
MasterKeyToFind:         66 a2 fa 17  23 19 39 bd

structure:               c0 70 03 39  de 72 30 e0
Plaintext 1:             c3 79 03 39  de 72 34 e4
Plaintext 2:             c2 77 03 39  de 72 30 e1
lower nibbles found:     06 02 0a 07  03 09 09 0d
Complete Key:            66 a2 fa 17  23 19 39 bd
number of pltxt:         624712959124  i.e.: 2^39.184403
number of false alarms:  4764629       i.e.: 2^22.183932
number of structures:    9532364
time elapsed:            1254407.310000 sec
```

We checked manually that the 2 returned plaintexts conform the differential path and compared our theoretical results with the practical ones. First, we notice that this result required by chance little less data than theoretically predicted ($2^{42.5}$) and second we notice that the ratio of false alarms meets the theory: since we encrypted $2^{39.184}$ plaintexts, we were able to create $2^{54.184}$ pairs so we expected a total of $2^{54.184-32} = 2^{22.184}$ pairs to pass the first test which is really close to the observed number of false alarms. This experiment takes near to 15 days.

The experiments on 9-round versions took us an average of 2 days to recover the complete key. Some of them are detailed in the full version of this paper [15]. Since that experiments were quicker, we were able to do several ones to compute average values. As for the previous result, some experiments needed less data than expected theoretically (by chance) but in average on 4 tests $2^{36,483}$ plaintexts were encrypted (which is really close to the $2^{36.5}$ expected), $2^{19,483}$ false alarms appeared and 47 h were required.

7 Conclusion

In this paper we propose the first attack on the full version of KLEIN-64. It improves the previous results from 10 to 12 rounds. For the 80-bit and 96-bit key versions we have provided several attacks on 13 and 14 rounds respectively.

We have implemented round-reduced versions of our attacks, and have been able to verify the theoretical complexities that we have predicted and to validate our assumptions. In particular, we have successfully implemented an attack on 10 rounds of KLEIN-64. This is the practical attack realized on the highest number of rounds, as previous results could not reach more than 8 rounds.

The main weakness of the cipher might be the fact that the `MixColumn` transformation does not correctly mix higher and lower nibbles, as it is the only transform that does so. Maybe considering other matrices instead could lead to a more solid construction. Also, the fact that the KeySchedule does not mix higher and lower nibbles helps the cryptanalyst to perform a reduced partial key search, so a stronger KeySchedule could help to prevent the attacks.

We believe that the family of attacks presented, though clearly dedicated to the cryptanalysis of KLEIN, might apply to other ciphers with big independences between two parts of the state.

References

1. Abdelraheem, M.A., Blondeau, C., Naya-Plasencia, M., Videau, M., Zenner, E.: Cryptanalysis of ARMADILLO2. In: Lee, D.H., Wang, X. (eds.) ASIACRYPT 2011. LNCS, vol. 7073, pp. 308–326. Springer, Heidelberg (2011)
2. Abed, F., Forler, C., List, E., Lucks, S., Wenzel, J.: Biclique Cryptanalysis of PRESENT, LED, and KLEIN. Cryptology ePrint Archive, Report 2012/591 (2012). http://eprint.iacr.org/
3. Ahmadian, Z., Salmasizadeh, M., Aref, M.R.: Biclique Cryptanalysis of the Full-Round KLEIN Block Cipher. Cryptology ePrint Archive, Report 2013/097 (2013). http://eprint.iacr.org/
4. Aumasson, J.-P., Naya-Plasencia, M., Saarinen, M.-J.O.: Practical Attack on 8 Rounds of the Lightweight Block Cipher KLEIN. In: Bernstein, D.J., Chatterjee, S. (eds.) INDOCRYPT 2011. LNCS, vol. 7107, pp. 134–145. Springer, Heidelberg (2011)
5. Badel, S., Dağtekin, N., Nakahara Jr, J., Ouafi, K., Reffé, N., Sepehrdad, P., Sušil, P., Vaudenay, S.: ARMADILLO: A Multi-Purpose Cryptographic Primitive Dedicated to Hardware. In: Mangard, S., Standaert, F.-X. (eds.) CHES 2010. LNCS, vol. 6225, pp. 398–412. Springer, Heidelberg (2010)
6. Bogdanov, A., Knežević, M., Leander, G., Toz, D., Varıcı, K., Verbauwhede, I.: SPONGENT: a Lightweight Hash Function. In: Preneel, B., Takagi, T. (eds.) CHES 2011. LNCS, vol. 6917, pp. 312–325. Springer, Heidelberg (2011)
7. Bogdanov, A.A., Knudsen, L.R., Leander, G., Paar, C., Poschmann, A., Robshaw, M., Seurin, Y., Vikkelsoe, C.: PRESENT: An Ultra-Lightweight Block Cipher. In: Paillier, P., Verbauwhede, I. (eds.) CHES 2007. LNCS, vol. 4727, pp. 450–466. Springer, Heidelberg (2007)
8. Borghoff, J., et al.: PRINCE – A Low-Latency Block Cipher for Pervasive Computing Applications. In: Wang, X., Sako, K. (eds.) ASIACRYPT 2012. LNCS, vol. 7658, pp. 208–225. Springer, Heidelberg (2012)
9. Borghoff, J., Knudsen, L.R., Leander, G., Thomsen, S.S.: Cryptanalysis of PRESENT-like Ciphers with Secret S-Boxes. In: Joux, A. (ed.) FSE 2011. LNCS, vol. 6733, pp. 270–289. Springer, Heidelberg (2011)

10. Collard, B., Standaert, F.-X.: A Statistical Saturation Attack Against the Block Cipher PRESENT. In: Fischlin, M. (ed.) CT-RSA 2009. LNCS, vol. 5473, pp. 195–210. Springer, Heidelberg (2009)
11. Gong, Z., Nikova, S., Law, Y.W.: KLEIN: A New Family of Lightweight Block Ciphers. In: Juels, A., Paar, C. (eds.) RFIDSec 2011. LNCS, vol. 7055, pp. 1–18. Springer, Heidelberg (2012)
12. Guo, J., Peyrin, T., Poschmann, A., Robshaw, M.: The LED Block Cipher. In: Preneel, B., Takagi, T. (eds.) CHES 2011. LNCS, vol. 6917, pp. 326–341. Springer, Heidelberg (2011)
13. Jha, V.K.: Cryptanalysis Of Lightweight Block Ciphers. Aalto University Master's thesis (2011). https://into.aalto.fi/download/attachments/9382995/VikashKumarJha_thesis.pdf
14. Knudsen, L., Leander, G., Poschmann, A., Robshaw, M.J.B.: PRINTcipher: A Block Cipher for IC-Printing. In: Mangard, S., Standaert, F.-X. (eds.) CHES 2010. LNCS, vol. 6225, pp. 16–32. Springer, Heidelberg (2010)
15. Lallemand, V., Naya-Plasencia, M.: Cryptanalysis of KLEIN (full version). Cryptology ePrint Archive, Report 2014/090 (2014). http://eprint.iacr.org/
16. Leander, G., Abdelraheem, M.A., AlKhzaimi, H., Zenner, E.: A Cryptanalysis of PRINTcipher: The Invariant Subspace Attack. In: Rogaway, P. (ed.) CRYPTO 2011. LNCS, vol. 6841, pp. 206–221. Springer, Heidelberg (2011)
17. Mendel, F., Rijmen, V., Toz, D., Varıcı, K.: DifferENtial Analysis of the LED Block Cipher. In: Wang, X., Sako, K. (eds.) ASIACRYPT 2012. LNCS, vol. 7658, pp. 190–207. Springer, Heidelberg (2012)
18. Naya-Plasencia, M.: How to Improve Rebound Attacks. In: Rogaway, P. (ed.) CRYPTO 2011. LNCS, vol. 6841, pp. 188–205. Springer, Heidelberg (2011)
19. Naya-Plasencia, M., Peyrin, T.: Practical Cryptanalysis of ARMADILLO2. In: Canteaut, A. (ed.) FSE 2012. LNCS, vol. 7549, pp. 146–162. Springer, Heidelberg (2012)
20. Nikolić, I., Wang, L., Wu, S.: Cryptanalysis of Round-Reduced LED. In: Moriai, S. (ed.) FSE 2013. LNCS, vol. 8424, pp. 112–130. springer, Heidelberg (2014)
21. Nikolic, I., Wang, L., Wu, S.: The Parallel-Cut Meet-in-the-Middle Attack. Cryptology ePrint Archive, Report 2013/530 (2013). http://eprint.iacr.org/
22. Suzaki, T., Minematsu, K., Morioka, S., Kobayashi, E.: *TWINE*: A Lightweight Block Cipher for Multiple Platforms. In: Knudsen, L.R., Wu, H. (eds.) SAC 2012. LNCS, vol. 7707, pp. 339–354. Springer, Heidelberg (2013)
23. Wu, W., Zhang, L.: LBlock: A Lightweight Block Cipher. In: Lopez, J., Tsudik, G. (eds.) ACNS 2011. LNCS, vol. 6715, pp. 327–344. Springer, Heidelberg (2011)
24. Yu, X., Wu, W., Li, Y., Zhang, L.: Cryptanalysis of Reduced-Round KLEIN Block Cipher. In: Wu, C.-K., Yung, M., Lin, D. (eds.) Inscrypt 2011. LNCS, vol. 7537, pp. 237–250. Springer, Heidelberg (2012)

Hash Functions

Branching Heuristics in Differential Collision Search with Applications to SHA-512

Maria Eichlseder(✉), Florian Mendel, and Martin Schläffer

IAIK, Graz University of Technology, Graz, Austria
maria.eichlseder@iaik.tugraz.at

Abstract. In this work, we present practical semi-free-start collisions for SHA-512 on up to 38 (out of 80) steps with complexity $2^{40.5}$. The best previously published result was on 24 steps. The attack is based on extending local collisions as proposed by Mendel et al. in their Eurocrypt 2013 attack on SHA-256. However, for SHA-512, the search space is too large for direct application of these techniques. We achieve our result by improving the branching heuristic of the guess-and-determine approach to find differential characteristics and conforming message pairs. Experiments show that for smaller problems like 27 steps of SHA-512, the heuristic can also speed up the collision search by a factor of 2^{20}.

Keywords: Hash functions · Cryptanalysis · SHA-512 · Collision attack · Guess-and-determine attack · Branching heuristic

1 Introduction

Since 2005, many collision attacks have been shown for commonly used and standardized hash functions. In particular, the collision attacks of Wang et al. [41,42] on MD5 and SHA-1 have convinced many cryptographers that these widely deployed hash functions can no longer be considered secure. As a consequence, NIST has proposed the transition from SHA-1 to the SHA-2 family. Many companies and organization follow this advice and have already migrated to SHA-2. Even more might do so, since Keccak [33] has not been standardized as SHA-3 yet and SHA-2 is faster on several platforms. In particular, SHA-512 is much faster than both SHA-256 and Keccak on most 64-bit platforms [2]. For this reason, it has been suggested to use a truncated version of SHA-512 even for 256-bit hash values [38]. NIST also defines this variant, called SHA-512/256, in FIPS 180-4 [32].

Nevertheless, not many cryptanalytic results on SHA-512 have been published in the last few years. The security of SHA-512 against preimage attacks was first studied by Aoki et al. in [1]. They presented a preimage attack on 46 out of 80 steps. This was later extended to 50 steps by Khovratovich et al. in [19]. Recently, Li et al. showed that particular preimage attacks can also be used to construct a free-start collision attack for up to 57 steps of SHA-512 in [24]. However, all attacks are only slightly faster than the respective generic attack complexities.

C. Cid and C. Rechberger (Eds.): FSE 2014, LNCS 8540, pp. 473–488, 2015.
DOI: 10.1007/978-3-662-46706-0_24

The currently best known practical collision attack on both the SHA-512 hash and compression function is for 24 steps. It has been published independently by Indesteege et al. [16] and by Sanadhya and Sarkar [36]. Both attacks are trivial extensions of the attack strategy of Nikolić and Biryukov [34] which applies to both SHA-256 and SHA-512. Recently, Mendel et al. [27,29] demonstrated how to extend these attacks to get collisions for the SHA-256 compression function on up to 38 steps with practical complexity.

The attacks by Mendel et al. use a guess-and-determine based automatic search tool to find differential characteristics and conforming message pairs for reduced SHA-256. Since the first publication by De Cannière and Rechberger on SHA-1 in [7], such tools have been constantly improved [21,22,27,29]. Nevertheless, the increased search space of SHA-512 (due to larger word sizes) prevented successful attacks without the application of new ideas.

To handle the larger search space of SHA-512, we propose a new branching heuristic for the guess-and-determine strategy used in these attacks. Our approach is inspired from related ideas in SAT solvers [15,23]. The heuristic performs a randomized look-ahead selection of candidates which should be guessed first. Using this approach, we can detect contradictions earlier and reduce the search space faster. More specifically, we are able to speed up the search on SHA-512 by a factor of about 2^{20} (for 27 steps), which allows us to construct practical collisions for 38 steps with a complexity of $2^{40.5}$.

The remainder of this paper is structured as follows. We first give a high-level overview of our attack strategy and related work in Sect. 2. In Sect. 3, we discuss branching heuristics used in SAT solvers and propose our new look-ahead branching heuristic for differential cryptanalysis tools. In Sect. 4, we demonstrate the application of the heuristic to SHA-512 and present a practical semi-free-start collision for 38 steps. Finally, we conclude in Sect. 5.

2 Motivation

In this section, we give a brief overview on the differential cryptanalysis of hash functions and how the guess-and-determine approach is used to search for differential characteristics. Furthermore, we provide a high-level view on optimization options to improve this search.

2.1 The Search for Differential Characteristics

A differential attack consists of two main parts: constructing a differential characteristic and finding a confirming message pair. Since the attacks by Wang et al. [40–42], these parts are further divided to improve the overall attack complexity as follows:

- **Find a differential characteristic**
  1. Construct the high-probability part of a characteristic.
  2. Determine the low-probability part of a characteristic.

– **Find a conforming message pair**
  3. Use message modification in low-probability part.
  4. Perform random trials in high-probability part.

We provide significant improvements in finding dense low-probability differential characteristics. To motivate our work, we first provide an overview of previously published methods and show how we improve upon these methods using improvements in the guess-and-determine approach.

Constructing the differential characteristic for the low-probability part is one of the most difficult tasks in a differential attack. The main reason is that such low-probability characteristics are usually very dense and have many (hidden) relations which need to be taken into account. Wang et al. found the dense low-probability characteristics for the attacks on MD4, MD5, RIPEMD, SHA-0 and SHA-1 mostly by hand [40–42]. However, for more complex hash functions, such an approach is infeasible. Therefore, (semi-)automatic approaches have been published soon afterwards [7,37]. These approaches have then been refined in a number of publications. Recently, more sophisticated approaches have been proposed that enable attacks on more complex hash functions such as SHA-256 [27,29] among many others [20,22,26,28]. All these approaches (including the search by hand) follow the guess-and-determine strategy.

2.2 The Guess-and-Determine Approach

The basic idea of the search algorithm is to pick and guess previously unrestricted bits. After each guess, the information gained from these restrictions is propagated to other bits. If an inconsistency occurs, the algorithm backtracks to an earlier state of the search and tries to correct it. Similar to [27], we denote these three parts of the search by decision (guessing), deduction (propagation), and backtracking (correction). Then, the search algorithm proceeds as in Algorithm 1 given below.

Algorithm 1. Guess-and-Determine Search Algorithm

Let U be a set of undetermined bits
while U contains undetermined bits **do**
  Decision (Guessing)
    1. pick an undetermined bit (randomly or heuristically)
    2. impose new constraints on this bit
  Deduction (Propagation)
    3. propagate the new information to other variables and equations
    4. **if** an inconsistency is detected, start backtracking,
       else continue with step 1
  Backtracking (Correction)
    5. try a different choice for the decision bit and continue with step 3.
    6. **if** all choices result in inconsistencies,
         undo guesses until this critical bit can be resolved

This procedure can also be visualized by a search tree, which is traversed by depth first search. The branching strategy decides on which variable to split the tree next and thus defines the tree's shape. Typically, the complete tree is much too large for complete traversal, so it is crucial that more promising branches are visited first. In addition, the backtracking algorithm can skip parts of the tree in favor of exploring more distant parts. This makes the search incomplete, but in practice greatly improves the performance.

The challenge in finding a long differential characteristic lies in the fine-tuning of the search algorithm. There are many possible variations, and details can determine whether the search succeeds or fails.

2.3 Improving the Guess-and-Determine Approach

Basically, a guess-and-determine is just a repetition of two steps: first, guess the value of some unknowns and second, determine the value of as many unknowns as possible. However, in practice more details need to be considered to mount successful guess-and-determine attacks on complex hash functions. The most important points to consider are given as follows:

1. **Problem Description:** The complexity of a guess-and-determine attack can be significantly improved if we first optimize the problem description. For example, first constructing a characteristic and then searching for a message pair is already such an optimization. Additionally, the choice of intermediate variables and a good starting point are crucial for a guess-and-determine attack to succeed.
2. **Guessing Strategy:** Instead of randomly guessing variables, using high-level information can lead to much better guesses. For example, by preferring bits (or even words) with no differences, characteristics tend to get sparser, have a higher probability, and conforming message pairs are more likely to exist.
3. **Branching Rules:** In every iteration, the guess-and-determine algorithm needs to decide which branch of the search tree to follow. Using a good branching heuristic, contradictions can be found faster and the search space can be reduced more quickly.
4. **Propagation:** Every time a variable is guessed, we need to check whether the guess is invalid, or new information on other variables can be determined. There is a trade-off between the effort we spend in this step and simply guessing more bits. Different propagation methods for ARX-based hash functions are covered in detail in [9,21,22,27].
5. **Backtracking:** To recover from bad search spaces which do not contain many solutions anymore, we need to backtrack. Two extreme options are performing a complete restart or examining the complete search space. A successful backtracking strategy for SHA-2 has been published in [27].

The first two points are very specific to a given problem and cannot be solved in general. In our attacks on SHA-512, a good starting point is constructed using improved local collisions, similar as in the attack on SHA-256 in [29]. The last two

points have already been covered in a number of publications. Additional efforts in these points did not improve the guess-and-determine attack on SHA-512. This leaves the branching rules which have not been optimized yet. In the following, we show that a good branching heuristic can significantly improve the efficiency of a guess-and-determine attack.

3 Branching Heuristics

Branching rules are one of the essential ingredients for guess-and-determine attacks. They define how the search algorithm selects the next variable to guess, and which guess values to try first for this variable. The branching rule aims to keep the search runtime as short as possible. Depending on whether the current partial assignment is correct (satisfiable) or contradictory, this means either that a satisfying solution is found as soon as possible, or that the contradiction is detected quickly. In the latter case, this corresponds to identifying a conflicting subset of unassigned variables and branching on these first in order to prune the search tree. The search trees traversed by different branching rules can vary drastically in size, from constant (for unsatisfiable problems) or linear (for satisfiable problems) to exponential in the number of variables [35].

This section first discusses existing branching rules used in general-purpose SAT solvers and for the cryptanalysis of hash functions. Afterwards, we introduce our randomized look-ahead heuristic.

3.1 Branching Heuristics in SAT Solvers

Most general-purpose SAT solvers are based on the Davis-Putnam-Logemann-Loveland (DPLL) algorithm [6], a guess-and-determine approach for satisfiability problems given in conjunctive normal form (CNF). The problem of choosing optimal branching variables and corresponding assignments for DPLL algorithms has been proven to be both NP-hard and coNP-hard [25]. However, there is a variety of commonly implemented branching rules based on different heuristics to evaluate the urgency or relevance of potential branching variables. In addition, meta-rules to select different branching rules depending on the situation and search history have been proposed [13].

Commonly used SAT branching rules can be categorized according to their target heuristic (current properties, look-ahead or history analysis), their output (a single branching variable/literal or a preselection of candidate variables) and their randomness (deterministic or randomized). Popularly used heuristics include the following:

– **Uniformly random**. A random unassigned variable is picked with uniform distribution. This approach is computationally cheapest. Many modern SAT solvers apply this rule with a small probability and otherwise use a more informed choice. In differential cryptanalysis, this is the most typical rule.

- **Small clauses**. The earliest heuristics greedily favor variables that appear in many small clauses. The rationale for this choice is twofold. First, smaller clauses need to be fulfilled "more urgently" since there are fewer options left that avoid contradictions. Second, even if the guessed literal evaluates to false in binary clauses, unit propagation ensues and curtails the search tree.

  Example heuristics of this category include Böhm's rule [3], MOM (maximum occurrences on clauses of minimum size) [10], and the Jeroslaw-Wang rules [17]. The latter, for example, assign weights $w(c)$ to clauses c that decrease exponentially with the clause length $|c|$. Each literal (OS-JW) or variable (TW-JW) scores according to the weight sum of all clauses it appears in, and the best literal or variable is picked for guessing.

  More recently, small clauses have been used as a preselection heuristic for more expensive look-ahead rules. In differential guess-and-determine attacks, two-bit conditions [27] play a related role. This preselection heuristic also favors variables with a higher number of closely coupled undetermined variables.
- **Literal count**. These heuristics ignore the clause size and simply count unresolved clauses linked to a variable. Examples include DLCS and DLIS as introduced by the GRASP solver [39]. It makes sense in CNF problems, where satisfying one literal resolves the complete clause. This does not apply, for example, for the xor-chains typically found in hash functions. Instead, this heuristic would create a large amount of (hidden) dependencies and reduce the remaining freedom without the positive effect of immediate propagation.
- **Conflict driven**. A more popular variation of literal counting is VSIDS, first implemented in Chaff [30] and later included in MiniSAT [8] and others. Here, the initial literal score of each variable decays over time via multiplication with a constant $\delta < 1$. However, scores are refreshed (bumped) with occurrences in newly learned clauses from the CDCL process. Effectively, the score keeps track of recent contradictions involving the variable. Critical variables with many recent contradictions are guessed first.

  The BerkMin solver extends this concept to bump not only variables from learned clauses, but from any clauses involved in the resolution process [12].

  In differential attacks, the backtracking strategy [27] provides a similar behaviour.
- **Look-ahead**. Instead of judging current properties of the formula or the previous search history, look-ahead heuristics analyse the actual effects of branching in a candidate variable [15, 23]. For example, the Satz solver performs Unit Propagation Look-Ahead: both possible assigments for each free variable are tested for consequences of this decision and the caused unit propagations. If one of two assignments causes a contradiction, the other is fixed; if both are contradictory, backtracking is started; and if both seem valid, the variable v is assigned score

$$\mathcal{M}(v) = w(\neg v) \cdot w(v) \cdot 1024 + w(\neg v) + w(v),$$

where $w(\ell)$ is typically the number of new binary clauses caused by the propagation of literal $\ell \in \{v, \neg v\}$.
- **Locality**. To limit the candidates for expensive look-ahead calculations, the candidate variables can be limited to those occurring in recently changed (reduced) clauses, as implemented in the marchdl solver [14].

Not all of these rules are suitable for general Boolean satisfiability problems that are not given in CNF format, as already indicated in the list above. In particular, if the propagation and learning process differs from the standard SAT case, the above rules can be counterproductive. On the positive side, dedicated solvers for specific applications can apply domain-specific knowledge to guide the search process.

3.2 The Look-Ahead Branching Heuristic

The branching strategy is one of the most promising areas for optimization in differential cryptanalysis tools based on tree search. Ideally, the branching strategy quickly navigates towards a valid assignment of variables and avoids subtrees without solutions. For detecting invalid subtrees, the branching strategy relies on the propagation method to detect contradictions as soon as possible. However, the propagation procedure can not only be used to decide whether previous guesses were contradictory. In addition, we also want to apply it to guide the branching strategy. The goal of this interaction is to minimize the size of the search tree in order to find solutions faster.

The basic principles of our implementation of the look-ahead branching heuristic are given by the following two observations:

- **Productive propagation is good.** Guessing a variable where propagation of the value determines (many) other variables can have multiple advantages compared to variables with less propagation. The most immediate effect is that the remaining search space is reduced. If more variables are determined right now, they will not create unnecessary subtrees for guessing later. The overall tree size and thus the complexity of the remaining search is reduced.
- **Contradictions are even better.** Of course, the overall search aims to find non-contradictory assignments. Nevertheless, discovering contradictory value assignments in the current subtree is consistently helpful for the remaining search. If only one of two possible value assignments is contradictory, the variable certainly needs to be fixed to the other value. If both values are contradictory, we must already have made an error with a previous guess and need to backtrack immediately. In both cases, it is clearly better to address the conflicting bit sooner rather than later.

Note that the first criterion is not beyond controversy. In particular, limiting the search space at the same time reduces the remaining degrees of freedom.

If one value assigned to a specific bit propagates better than the second possible value, then, intuitively speaking, the probability for a solution in the remaining search space for the first option is lower than for the second value.

3.3 Implementation of the Look-Ahead Branching Heuristic

In order to implement the criteria above in a practical branching heuristic, we use a look-ahead approach related to the Unit-Propagation Look-Ahead (UPLA) used in some SAT solvers. When the branching rule needs to select the next variable to guess, each candidate is in turn evaluated.

In more detail, for each candidate, a value is tentatively assigned and the propagation method is applied to determine the consequences of this assignment. If a contradiction occurs, this candidate is selected immediately. Otherwise, the number of propagated variables is calculated. If it is better than the previously favorite candidate, this variable becomes the new favorite.

There are two performance-related problems with this basic approach. First, performing the look-ahead propagation for all free variables is very costly. Second, the basic UPLA approach includes no randomization. However, we need randomization since a complete search of the tree is typically computationally infeasible in differential cryptanalysis. Instead, large tree parts are skipped and the search is restarted regularly. To avoid becoming lost in the same search branches over and over again, it is essential that the branching strategy is sufficiently randomized.

We address both problems at once by selecting only a random subset of variables for closer evaluation. Our branching heuristic is summarized in Algorithm 2.

Algorithm 2. Look-ahead branching heuristic for differential cryptanalysis

Let U be a set of undetermined bits and $s_{\max}$ the limit of look-ahead candidates.
repeat
  Guessing
    1. pick a bit $v \in U$ randomly and increment s
    2. impose new constraints on this bit v
  Propagation
    3. propagate the new information to other variables and equations
    4. **if** an inconsistency is detected, **return** v as the decision bit
       else count the number m of additional variables that were assigned due to this guess and save the pair (v, m) in a list L.
  Update
    5. remove all variables that were assigned due to the guess v from the set U
    6. undo all changes to restore the original assignment
until U is empty or $s \geq s_{\max}$
return v^* from L with the highest score m as the decision bit

The size of the randomly selected subset is an essential parameter for the success of the heuristic. To limit the look-ahead costs, we limit the maximum

subset size by a constant number that is chosen in the beginning of the search procedure, depending on the specific problem instance. In order to also provide sufficient randomization, we additionally bound the size relative to the current number of unguessed variables.

Beside the subset size, the decision which individual variables to select for look-ahead plays a role. UPLA-based solvers use a pre-selection of interesting candidates, for example by locality criteria. In our case, the search performance can be greatly improved by only guessing bits of specific hash function words and favoring bits with more two-bit conditions or bits involved in recent conflicts. However, the selection must remain sufficiently randomized.

Additionally, we do not explicitly evaluate variables that were already determined by the propagation procedure of one of the previous candidates. We mark these as evaluated without calculating a separate look-ahead and without considering them as favorite candidates, since their score is at most as good as the bit that triggered their propagation (at least with respect to one of the assignment options).

4 Application to SHA-512

In this section, we discuss the application of our look-ahead branching heuristics to SHA-512. As a result, we are able to construct the first practical collision on the reduced SHA-512 compression function for 38 out of 80 steps. The best previously published result was on 24 steps.

4.1 Brief Description of SHA-512

SHA-512 is an iterated hash function that processes 1024-bit input message blocks and produces a 512-bit hash value. In the following, we briefly describe the hash function. It basically consists of two parts: the message expansion and the state update transformation. A detailed description of the hash function is given in [31].

Message Expansion. The message expansion of SHA-512 splits the 1024-bit message block into 16 64-bit words M_i and expands them into 80 expanded message words W_i as follows:

$$W_i = \begin{cases} M_i & 0 \leq i < 16 \\ \sigma_1(W_{i-2}) + W_{i-7} + \sigma_0(W_{i-15}) + W_{i-16} & 16 \leq i < 80 \end{cases}$$

The functions $\sigma_0(x)$ and $\sigma_1(x)$ are given by

$$\sigma_0(x) = (x \ggg 1) \oplus (x \ggg 8) \oplus (x \gg 7)$$
$$\sigma_1(x) = (x \ggg 19) \oplus (x \ggg 61) \oplus (x \gg 6).$$

State Update Transformation. The state update transformation starts from a (fixed) initial value IV of 8 64-bit words $A_{-4}, \ldots, A_{-1}, E_{-4}, \ldots, E_{-1}$ and updates them in 80 steps. In each step one expanded message word W_i is used to compute the two state variables E_i and A_i as follows:

$$E_i = A_{i-4} + E_{i-4} + \Sigma_1(E_{i-1}) + \mathrm{IF}(E_{i-1}, E_{i-2}, E_{i-3}) + K_i + W_i$$
$$A_i = E_i - A_{i-4} + \Sigma_0(A_{i-1}) + \mathrm{MAJ}(A_{i-1}, A_{i-2}, A_{i-3}).$$

For the definition of the step constants K_i we refer to [31]. The bitwise Boolean functions IF and MAJ used in each step are defined by

$$\mathrm{IF}(x, y, z) = x \wedge y \oplus x \wedge z \oplus z$$
$$\mathrm{MAJ}(x, y, z) = x \wedge y \oplus y \wedge z \oplus x \wedge z,$$

and the linear functions Σ_0 and Σ_1 are defined as follows:

$$\Sigma_0(x) = (x \ggg 28) \oplus (x \ggg 34) \oplus (x \ggg 39)$$
$$\Sigma_1(x) = (x \ggg 14) \oplus (x \ggg 18) \oplus (x \ggg 41).$$

After the last step of the state update transformation, the initial values are added to the output values of the last step (Davies-Meyer construction). The result is the final hash value or the initial value for the next message block.

4.2 Extending the Attacks on SHA-256 to SHA-512

For our collision attacks on SHA-512, we use the same strategy as in the attack on SHA-256 in [29]. Since the message expansion and state update transformation is the same (except for larger word sizes and different rotation values in Σ_i, σ_i), we can use similar local collisions (with differences in the same message words) to construct semi-free-start collisions for the compression function on up to 38 steps.

The starting point for 38 steps uses a local collision which spans 18 steps, with differences in 6 expanded message words $(W_7, W_8, W_{10}, W_{15}, W_{23}, W_{24})$. For more details on how to select the starting point, we refer to [29]. Once the starting point is fixed, the main task is to find a differential characteristic and confirming message pair for this 18-step local collision.

By using the same guessing, backtracking and propagation strategy, we did not find any results for 38 steps of SHA-512. Due to the large word size and thus, larger search space, contradictions are detected much later in SHA-512. We have tried different approaches on every level, but did not succeed in finding any valid differential characteristics. The solution was to optimize the branching strategy to detect on one hand contradictions earlier and on the other side to reduce the search space faster.

4.3 Improving the Search Using Look-Ahead Branching

To improve the search algorithm, we use the look-ahead branching heuristic proposed in Sect. 3.3. As discussed there, the choice of the subset size $s_{\max}$ is critical for the behaviour of the heuristic. We have evaluated different variants of the heuristic and get the best results for a limit of $s_{\max} = 16$. Larger values of $s_{\max}$ further reduce the tree depth, but due to the additional cost for evaluating more candidates, this does not improve the overall runtime.

Additionally, with larger subset sizes, the search tends to visit very similar subtrees again and again after each restart. This is particularly critical if the search space is limited to a few words, as in the focused search strategy described below. For other hash functions with larger states sizes or less focused search strategies, the optimal value for $s_{\max}$ may be very different.

Similar to [29], the guess-and-determine attack is separated into three stages. The rules of the guessing strategy are given in Table 1 and the three stages are summarized as follows:

Stage 1:
We first search for a consistent differential characteristic in the message expansion. Hence, we only add unconstrained bits ('?') and difference bits ('x') of W to the set U.

Stage 2:
We continue with the search for a differential characteristic in the state update. Hence, we add all unconstrained bits ('?') and difference bits ('x') of A and E to the set U. We pick decision bits more often from A, since this results in sparser characteristics for A. Experiments have shown that in this case, confirming message pairs are easier to find in the last stage.

Stage 3:
In the last stage, we search for confirming message pairs by guessing bits without difference ('-'). We only pick decision bits of A, E and W which are constrained by two-bit conditions, similar as in [27]. This serves as a preselection heuristic for the branching look-ahead.

Table 1. Decision rules in different search stages.

Stage	Decision bit	Decision rule		
		Probability	Choice 1	Choice 2
1–2	?	1	-	x
	x	1/2	u	n
		1/2	n	u
3	-	1/2	0	1
		1/2	1	0

4.4 Results

Using the improvements in the branching heuristic proposed in the previous section, we are able to find semi-free-start collisions for SHA-512 on up to 38 steps. Finding a differential characteristic together with a conforming message pair took 5441 s (≈1.5 h) on a cluster with 40 CPUs. This corresponds to a complexity of about $2^{40.5}$ evaluations of the SHA-512 compression function. The colliding message pair is given in Table 2 and the differential characteristic is shown in Table 3.

Table 2. Example of a semi-free-start collision for 38 steps of SHA-512.

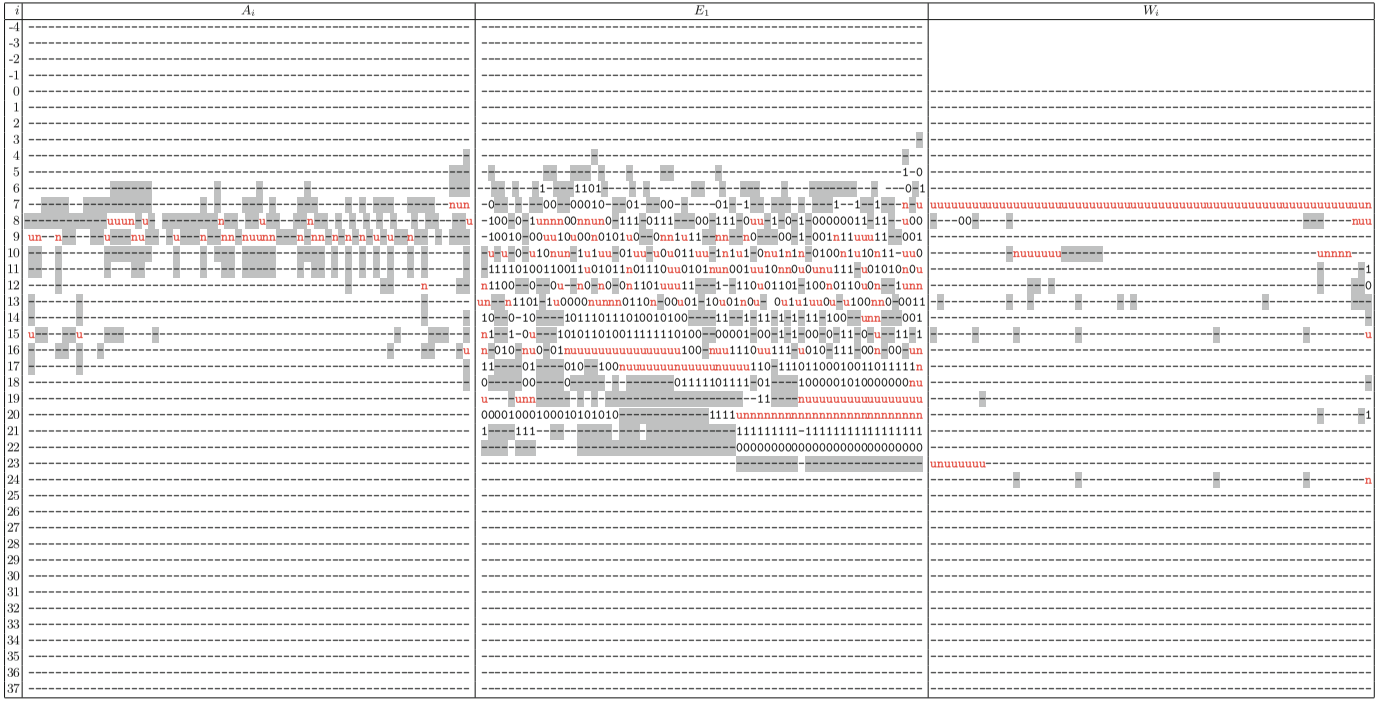
h_0	e8626f53a3771964	2ae427b8c5065790	c8fd5a1628fc3337	0f362d297f82f987
	89166a0c022ffc40	c2c49c30e629239f	d1fa8bd692843025	ad4bba64c797e6ec
m	610519a88f0d2809	3addc83f01c8b179	84afa7a2772c6141	ad539854e64c9cce
	85450b73549b2085	7296b5291f31c0d9	fc978d9624e2c2cc	fffffffffffffffe
	92114cb9d2f4cd9b	34a3198b79871212	cca7f43154e38081	ac0598a589168fe1
	f32ae6a0070a8d2e	755aa5cada87e894	4b9bd7df3c94b667	65291f2b80cc8c51
m^*	610519a88f0d2809	3addc83f01c8b179	84afa7a2772c6141	ad539854e64c9cce
	85450b73549b2085	7296b5291f31c0d9	fc978d9624e2c2cc	0000000000000001
	92114cb9d2f4cd9c	34a3198b79871212	cca8143154e38079	ac0598a589168fe1
	f32ae6a0070a8d2e	755aa5cada87e894	4b9bd7df3c94b667	65291f2b80cc8c50
Δm	0000000000000000	0000000000000000	0000000000000000	0000000000000000
	0000000000000000	0000000000000000	0000000000000000	ffffffffffffffff
	0000000000000007	0000000000000000	000fe0000000000f8	0000000000000000
	0000000000000000	0000000000000000	0000000000000000	0000000000000001
h_1	946a28eedc3b2ff6	c4573d0a13ea6268	11f07b04b06900dd	897c606e4053bbe4
	2406aae9d58504b4	89b237932b061ba8	663402cb4bb1972c	d99c062dce945423

To show the benefit of our new look-ahead branching heuristic, we have performed some comparisons. Without look-ahead branching, we were able to find a semi-free-start collision for 27 steps of SHA-512 using 4 days on a cluster with 40 nodes, which corresponds to a complexity of about $2^{46.5}$. Using look-ahead branching with $s_{\max} = 16$ we can find differential characteristics with conforming message pairs within seconds on a standard PC (complexity $2^{26.5}$).

The heuristic can also be used to improve the search complexity for primitives with a smaller state to a certain extent. For example, experiments show a speedup of more than an order of magnitude for attacks on 27 or 38 steps of SHA-256. However, due to the heuristic nature of the improvement and the general sensitivity of the search procedure to different parameters, the effects are hard to quantify.

Unfortunately, we were not able to extend the semi-free-start collisions to collision attacks on the hash function. The main reason is that the resulting differential characteristics are quite dense and we do not have enough freedom to match the IV with practical complexity.

Table 3. Differential characteristic for a semi-free-start-collision of SHA-512 reduced to 38 steps (bits with two-bit conditions highlighted).



5 Conclusions

In this work, we have improved the best semi-free-start collisions on SHA-512 from 24 to 38 steps. Our attack has a practical complexity of $2^{40.5}$ and we have shown a colliding message pair. We get this result by applying the semi-free-start collision attack on 38 steps of SHA-256 to SHA-512. However, due to the increased word size, and hence increased search space, a straight-forward extension was not possible.

To get these results we have analyzed possible improvements in the guess-and-determine approach to find differential characteristics and conforming message pairs. We got the best results by optimizing the branching heuristic using ideas from SAT solvers. Our heuristic performs a randomized look-ahead selection of candidates which should be guessed first.

Future work includes to apply the look-ahead heuristic to more complex designs. Also, other techniques from SAT solvers may improve guess-and-determine attacks in differential cryptanalysis. However, a direct application of SAT solver techniques without taking high-level information on differential cryptanalysis into account is usually not successful. Finally, an open question is how to use our new results to improve the collision attacks on the SHA-512 hash function.

Acknowledgments. The work has been supported in part by the Secure Information Technology Center-Austria (A-SIT), by the Austrian Government through the research program FIT-IT Trust in IT Systems (Project SePAG, Project Number 835919), and by the European Commission through the FP7 Joint Technology Initiatives (Call ARTEMIS-2012-1, Project Arrowhead, Grant Agreement Number 332987).

References

1. Aoki, K., Guo, J., Matusiewicz, K., Sasaki, Y., Wang, L.: Preimages for step-reduced SHA-2. In: Matsui, M. (ed.) ASIACRYPT 2009. LNCS, vol. 5912, pp. 578–597. Springer, Heidelberg (2009)
2. Bernstein, D.J., Lange, T.: eBASH: ECRYPT benchmarking of all submitted hashes, January 2011. http://bench.cr.yp.to/ebash.html
3. Buro, M., Kleine-Büning, H.: Report on a SAT competition. Bull. Eur. Assoc. Theor. Comput. Sci. **49**, 143–151 (1993)
4. Canteaut, A. (ed.): FSE 2012. LNCS, vol. 7549. Springer, Heidelberg (2012)
5. Cramer, R. (ed.): EUROCRYPT 2005. LNCS, vol. 3494. Springer, Heidelberg (2005)
6. Davis, M., Logemann, G., Loveland, D.W.: A machine program for theorem-proving. Commun. ACM **5**(7), 394–397 (1962)
7. De Cannière, C., Rechberger, C.: Finding SHA-1 characteristics: general results and applications. In: Lai, X., Chen, K. (eds.) ASIACRYPT 2006. LNCS, vol. 4284, pp. 1–20. Springer, Heidelberg (2006)
8. Eén, N., Sörensson, N.: An extensible SAT-solver. In: Giunchiglia and Tacchella [11], pp. 502–518
9. Eichlseder, M., Mendel, F., Nad, T., Rijmen, V., Schläffer, M.: Linear propagation in efficient guess-and-determine attacks. In: Budaghyan, L., Helleseth, T., Parker, M. G. (eds.) WCC (2013). http://www.selmer.uib.no/WCC2013/

10. Freeman, J.W.: Improvements to propositional satisfiability search algorithms. Ph.D. thesis, Departement of computer and Information science, University of Pennsylvania, Philadelphia (1995)
11. Giunchiglia, E., Tacchella, A. (eds.): SAT 2003. LNCS, vol. 2919. Springer, Heidelberg (2004)
12. Goldberg, E.I., Novikov, Y.: BerkMin: a fast and robust SAT-solver. In: DATE, pp. 142–149. IEEE Computer Society (2002)
13. Herbstritt, M., Becker, B.: Conflict-based selection of branching rules. In: Giunchiglia and Tacchella [11], pp. 441–451
14. Heule, M., van Maaren, H.: March_dl: adding adaptive heuristics and a new branching strategy. JSAT **2**(1–4), 47–59 (2006)
15. Heule, M., van Maaren, H.: Look-ahead based SAT solvers. In: Biere, A., van Heule, M., Maaren, H., Walsh, T. (eds.) Handbook of Satisfiability. Frontiers in Artificial Intelligence and Applications, vol. 185, pp. 155–184. IOS Press, Amsterdam (2009)
16. Indesteege, S., Mendel, F., Preneel, B., Rechberger, C.: Collisions and other non-random properties for step-reduced SHA-256. In: Avanzi, R.M., Keliher, L., Sica, F. (eds.) SAC 2008. LNCS, vol. 5381, pp. 276–293. Springer, Heidelberg (2009)
17. Jeroslow, R.G., Wang, J.: Solving propositional satisfiability problems. Ann. Math. Artif. Intell. **1**, 167–187 (1990)
18. Johansson, T., Nguyen, P.Q. (eds.): EUROCRYPT 2013. LNCS, vol. 7881. Springer, Heidelberg (2013)
19. Khovratovich, D., Rechberger, C., Savelieva, A.: Bicliques for preimages: attacks on Skein-512 and the SHA-2 family. In: Canteaut [4], pp. 244–263
20. Landelle, F., Peyrin, T.: Cryptanalysis of full RIPEMD-128. In: Johansson and Nguyen [18], pp. 228–244
21. Leurent, G.: Analysis of differential attacks in ARX constructions. In: Wang, X., Sako, K. (eds.) ASIACRYPT 2012. LNCS, vol. 7658, pp. 226–243. Springer, Heidelberg (2012)
22. Leurent, G.: Construction of differential characteristics in ARX designs application to skein. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013, Part I. LNCS, vol. 8042, pp. 241–258. Springer, Heidelberg (2013)
23. Li, C.M., Anbulagan: Heuristics based on unit propagation for satisfiability problems. In: IJCAI, vol. 1, pp. 366–371. Morgan Kaufmann, San Francisco (1997)
24. Li, J., Isobe, T., Shibutani, K.: Converting meet-in-the-middle preimage attack into pseudo collision attack: application to SHA-2. In: Canteaut [4], pp. 264–286
25. Liberatore, P.: On the complexity of choosing the branching literal in DPLL. Artif. Intell. **116**(1–2), 315–326 (2000)
26. Mendel, F., Nad, T., Scherz, S., Schläffer, M.: Differential attacks on reduced Ripemd-160. In: Gollmann, D., Freiling, F.C. (eds.) ISC 2012. LNCS, vol. 7483, pp. 23–38. Springer, Heidelberg (2012)
27. Mendel, F., Nad, T., Schläffer, M.: Finding SHA-2 characteristics: searching through a minefield of contradictions. In: Lee, D.H., Wang, X. (eds.) ASIACRYPT 2011. LNCS, vol. 7073, pp. 288–307. Springer, Heidelberg (2011)
28. Mendel, F., Nad, T., Schläffer, M.: Finding collisions for round-reduced SM3. In: Dawson, E. (ed.) CT-RSA 2013. LNCS, vol. 7779, pp. 174–188. Springer, Heidelberg (2013)
29. Mendel, F., Nad, T., Schläffer, M.: Improving local collisions: new attacks on reduced SHA-256. In: Johansson and Nguyen [18], pp. 262–278
30. Moskewicz, M.W., Madigan, C.F., Zhao, Y., Zhang, L., Malik, S.: Chaff: engineering an efficient SAT solver. In: DAC, pp. 530–535. ACM (2001)

31. National Institute of Standards and Technology. FIPS PUB 180–3: Secure Hash Standard. Federal Information Processing Standards Publication 180–3, U.S. Department of Commerce, October 2008. http://www.itl.nist.gov/fipspubs
32. National Institute of Standards and Technology. FIPS PUB 180–4: Secure Hash Standard. Federal Information Processing Standards Publication 180–4, U.S. Department of Commerce, March 2012. http://www.itl.nist.gov/fipspubs
33. National Institute of Standards and Technology. SHA-3 Selection Announcement, October 2012. http://csrc.nist.gov/groups/ST/hash/sha-3/sha-3_selection_announcement.pdf
34. Nikolić, I., Biryukov, A.: Collisions for step-reduced SHA-256. In: Nyberg, K. (ed.) FSE 2008. LNCS, vol. 5086, pp. 1–15. Springer, Heidelberg (2008)
35. Ouyang, M.: How good are branching rules in DPLL? Discrete Appl. Math. **89**(1–3), 281–286 (1998)
36. Sanadhya, S.K., Sarkar, P.: New collision attacks against up to 24-step SHA-2. In: Chowdhury, D.R., Rijmen, V., Das, A. (eds.) INDOCRYPT 2008. LNCS, vol. 5365, pp. 91–103. Springer, Heidelberg (2008)
37. Schläffer, M., Oswald, E.: Searching for differential paths in MD4. In: Robshaw, M. (ed.) FSE 2006. LNCS, vol. 4047, pp. 242–261. Springer, Heidelberg (2006)
38. Shay Gueron, J.W., Johnson, S.: SHA-512/256. Cryptology ePrint Archive, Report 2010/548 (2010). http://eprint.iacr.org/
39. Marques-Silva, J.: The impact of branching heuristics in propositional satisfiability algorithms. In: Barahona, P., Alferes, J.J. (eds.) EPIA 1999. LNCS (LNAI), vol. 1695, pp. 62–74. Springer, Heidelberg (1999)
40. Wang, X., Lai, X., Feng, D., Chen, H., Yu, X.: Cryptanalysis of the hash functions MD4 and RIPEMD. In: Cramer [5], pp. 1–18
41. 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)
42. Wang, X., Yu, H.: How to break MD5 and other hash functions. In: Cramer [5], pp. 19–35

On the Minimum Number of Multiplications Necessary for Universal Hash Functions

Mridul Nandi(✉)

Applied Statistics Unit, Indian Statistical Institute, Baranagar, India
mridul@isical.ac.in

Abstract. Let $d \geq 1$ be an integer and R_1 be a finite ring whose elements are called **block**. A d-block universal hash over R_1 is a vector of d multivariate polynomials in message and key block such that the maximum *differential probability* of the hash function is "low". Two such single block hashes are pseudo dot-product (PDP) hash and Bernstein-Rabin-Winograd (BRW) hash which require $\frac{n}{2}$ multiplications for n message blocks. The Toeplitz construction and d independent invocations of PDP are d-block hash outputs which require $d \times \frac{n}{2}$ multiplications. However, here we show that *at least* $(d-1)+\frac{n}{2}$ *multiplications are necessary* to compute a universal hash over n message blocks. We construct a *d-block universal hash, called* EHC*, which requires the matching* $(d-1)+\frac{n}{2}$ *multiplications for* $d \leq 4$. Hence it is optimum and our lower bound is tight when $d \leq 4$. It has similar parllelizibility, key size like Toeplitz and so it can be used as a light-weight universal hash.

Keywords: Universal hash · AXU hash · Multivariate polynomial · Error correcting code · Vandermonde matrix · Toeplitz hash

1 Introduction

Universal hash function and its close variants *ΔU hash* [10,13,40,42,43] are used as building blocks of several cryptographic constructions, e.g., *message authentication codes* [10,49], domain extension of *pseudorandom functions* [2,4], extractors [15,32] and quasi-randomness [44]. It also has close connection with *error correcting codes* and other combinatorial objects [13,43].

Informally, a universal hash function h takes two inputs, a key k and a *message m of arbitrary length*, and produces a *fixed-length* output $h_k(m) := h(k,m)$. For a universal (or ΔU) hash function h the following holds: for any two distinct messages m_1, m_2, the collision probability $\mathsf{Pr}[h_k(m_1) = h_k(m_2)]$ (or differential probability $\max_\delta \mathsf{Pr}[h_k(m_1) - h_k(m_2) = \delta]$) is small for an uniformly chosen key k. Formal definitions can be found in Sect. 2.

A very popular application of universal hash is to obtain a domain extension of pseudorandom function (or PRF) and message authentication code (or MAC). Let f be a PRF over fixed length input. When h has low collision probability, the composition function $f \circ h$ is a PRF [4] over arbitrary length. Thus h behaves

C. Cid and C. Rechberger (Eds.): FSE 2014, LNCS 8540, pp. 489–508, 2015.
DOI: 10.1007/978-3-662-46706-0_25

as a preprocessor to reduce the problem of designing arbitrary size input PRF to a fixed small size input PRF. Similarly, we can show that mapping (N, M) to $f(N) \oplus h(M)$ is a MAC [6] over arbitrary length messages when N is used as a nonce (i.e., not repeating) and h has low differential probability. These methods are only useful when we process long message and use a much faster h (than a PRF f). So our main question of this paper is that how fast a universal hash function could be in a reasonable computational model?

Multiplication Complexity. The above question has been answered [26,28] in terms of order in different circuit level computational models, e.g., branching model. In this paper, we consider "multiplication complexity" in "algebraic computation model" [24] in which a polynomial or a rational function is computed by using addition and multiplication (or division) over an underlying ring (or a field) R_1 in a sequence (see Definition 4 for more details). For example, to compute $x_1x_2 + x_1x_3$, we can compute it as $v_1 + v_2$ where $v_1 = x_1x_2$ and $v_2 = x_1x_3$. This computation requires two multiplications. However, the same polynomial can be computed as $x_1(x_2 + x_3)$ which requires only one multiplication. We define multiplication complexity of a multivariate polynomial as the minimum number of multiplications required for all possible computations of H. The multiplication complexities of some standard multivariate polynomials have been studied before and a brief survey is given in Appendix. Our target question of this paper is to **obtain a lower bound** of multiplication complexities among all ΔU hash functions and to show the **tightness of the bound by producing an example**.

1.1 Our Contribution and Outline of the Paper

In the following we assume a universal hash function hashes all messages from R_1^ℓ to R_1^d (usually $d = 1$) and hence multiplication complexity is measured in terms of ℓ and d.

Optimality of pseudo dot-product and BRW hash. In this paper **we prove that a hash function with low differential probability must have multiplication complexity at least** $\ell/2$ (see Theorem 3 in Sect. 5). We show it by proving contrapositive. If a function has multiplication complexity $c < \ell/2$ then there are $2c$ multiplicands. As we have ℓ message blocks and key blocks are linear in multiplicands we are able to solve for two distinct messages from R_1^ℓ which map to all $2c$ multiplicands identically for all keys. Hence differential probability is one. Even though the lower bound seems intuitive, to the best of our knowledge, it was not known before. The pseudo dot-product [46] based hash PDP (e.g. NMH hash [14], NMH* [14], NH [4] and others [8,21]) defined as (for even ℓ)

$$\mathsf{PDP}_{k_1,\ldots,k_\ell}(m_1,\ldots,m_\ell) = (m_1 + k_1)(m_2 + k_2) + \cdots + (m_{\ell-1} + k_{\ell-1})(m_\ell + k_\ell)$$

and Bernstein-Rabin-Winograd or BRW hash [7,36] are two known examples which achieve this bound ($\ell/2$ multiplications for ℓ message blocks).

Optimality of multiple block hash. We also extend this bound for multiple block hash outputs (such as Toeplitz construction [26] or independent applications of a single block hash function). To compute

$$(H_1 := x_1x_2 + x_3x_4, \quad H_2 := x_1x_3 + x_2x_4),$$

we can compute H_1 by two multiplications (it can be shown that H_1 or H_2 individually can not be computed in one multiplication only) and then compute $H_2 = (x_1 + x_4)(x_2 + x_3) - H_1$ by one multiplication. Similarly for d polynomials $H_1, \ldots, H_d$ (with individual multiplication complexity c) there is a scope of computing all d polynomials simultaneously in less than cd multiplications. In Theorem 5 (Sect. 5), **we prove that to obtain d block hash outputs on ℓ block messages, we need at least $(d-1) + \ell/2$ multiplications**.

Construction with Matching Complexity. So far, no construction is known achieving this lower bound for $d > 1$. Note that both Toeplitz and independent invocation applied to the PDP requires $\ell d/2$ multiplications.[1] So there is a possibility of scope of a hash construction having better multiplication complexity. In this paper, for $d \leq 4$ **we provide a d-block Δ-universal hash, called** xxx **EHC or encode-hash-combiner** (see Algorithm 1, in Sect. 4). The main ingredient of the construction was introduced in [17]. Here, we first encode the input using an efficiently computable linear error correcting code [27] with minimum distance d so that codewords for different inputs will differ in at least d places; then we feed the i^{th} **encoded symbol** (e.g., a pair of blocks for PDP) through its underlying universal hash function (e.g., PDP which requires one multiplication for a symbol or two blocks); and then apply another efficient linear combiner to the all hash outputs to obtain the final d block hash outputs. The optimization in [17] is entirely aimed at linear-time asymptotic encodings [41], which don't have much connection to concrete performance. Moreover, codewords can not be computed in online manner with small buffer and requires at least ℓ-blocks of memory (in addition to input and output size). This is the possible reason that community has not found any interest to implement it (even for smaller ℓ, i.e. for small messages).

Our choice of code and combiners (satisfying some desired property) are based on Vandermonde matrices which can be computed with small buffer. Number of multiplication will essentially depend on size of codewords and due to choice of MDS code we need exactly $(d-1) + \ell/2$ multiplications. Hence, the construction is optimum and our bound is tight. In terms of key size and parallelizibilty, both Toeplitz and EHC are similar. The idea trivially does not extend for $d > 4$ as we do not find any appropriate error correcting code with distance $d > 5$.

2 Definitions: Universal and Δ-universal Hash Function

ΔU Hash Function. A hash function h is a $(\mathcal{K}, D, R)$-family of functions $\{h_k := h(k, \cdot) : D \to R\}_{k \in \mathcal{K}}$ defined on its domain or message space D, taking

[1] Applying the Theorem 1 in [47], we can prove that these constructions have multiplication complexity $\ell d/2$.

values on a group R, called *output space* and indexed by the **key space** $\mathcal{K}$. Usual choices of R are (i) $\mathbb{Z}_p$ (the field of modulo a prime p), (ii) $\mathbb{Z}_{2^w}$ (the ring of modulo 2^w) (iii) $\mathbb{F}_{2^n}$ (Galois field of size 2^n) and (iv) R_1^d with **coordinate wise operation**, where R_1 is one of the previous choices. In the last example when $d > 1$, h is also called **multi** or d**-block** hash. An element of R (or R_1 for the multi-block) is called block. In general, we write R_1 even for $d = 1$. However, the output space is always denoted by $R = R_1^d$, $d \geq 1$. Except for $(\mathbb{Z}_p)^d$, R can be viewed as the set $\{0,1\}^N$ by using the canonical encodings and we say that hash size is N.

Definition 1 (ϵ-ΔU hash function). *A $(\mathcal{K}, D, R)$-family h is called* ϵ-Δ**U (universal) hash function** *if for any two distinct x and x' in D and a $\delta \in R$, the* δ-differential probability $\mathsf{diff}_{h,\delta}[x, x'] := \mathsf{Pr}_{\mathbf{K}}[h_{\mathbf{K}}(x) - h_{\mathbf{K}}(x') = \delta] \leq \epsilon$ *where the random variable* $\mathbf{K}$ *is uniformly distributed over the set $\mathcal{K}$.*

Unless mentioned explicitly, we always mean key $\mathbf{K}$ to be chosen uniformly from its key space. The *maximum δ-differential probability* over all possible of two distinct inputs x, x' is denoted by $\Delta_{h,\delta}$. The *maximum differential probability* $\Delta_h := \max_\delta \Delta_{h,\delta}$. If the addition is bit-wise xor "$\oplus$" on $R = \{0,1\}^N$, we call the hash family ϵ-**AXU** (almost-xor-universal) hash function [37].

Universal Hash Function. When $\delta = 0$, the 0-differential event is equivalent to collision. So we write $\mathsf{diff}_{h,0}[x, x']$ and $\Delta_{h,0}$ by $\mathsf{coll}_h[x, x']$ and coll_h respectively and we call them collision probabilities.

Definition 2 (ϵ-U hash function). *A hash family h is called* ϵ-**universal** *(or ϵ-U) if* $\mathsf{coll}_h := \max_{x \neq x'} \mathsf{Pr}_{\mathbf{K}}[h_{\mathbf{K}}(x) = h_{\mathbf{K}}(x')] \leq \epsilon$.

Balanced Hash Function. We call h ϵ-**balanced** [23,31] on a subset $D' \subseteq D$ if $\mathsf{Pr}[h_{\mathbf{K}}(x) = y] \leq \epsilon$ for all $x \in D'$, $y \in R$. If $D' = D$ then we call it ϵ-balanced. Note that ϵ is always at least $1/|R|$ for ϵ-ΔU (shown in [43]) and ϵ-balanced function (easy to check from definition) but not necessarily for an ϵ-U hash function [43]. An ϵ-balanced functions are useful to prove ϵ-ΔU property whenever h_K's are linear [23]. More precisely, for a linear hash, ϵ-ΔU is equivalent to ϵ-balanced function on $R \setminus \{0\}$.

3 Analysis Methods of Universal Hash Functions

In this section all messages (and possibly key) blocks are elements of the underlying field R_1 of size q.

3.1 Multi-linear Hash and Poly-Hash

The hash mapping $(m_1, \ldots, m_l) \mapsto m_1 \cdot \mathbf{K}_1 + \cdots + m_\ell \cdot \mathbf{K}_\ell$ can be shown to be q^{-1}-ΔU hash function.[2] It is known as *multi-linear hash* ML[13,49]. Later MMH

[2] One can also prove it by applying Lemma 2 as it is a sum hash.

was proposed [14] with a performance record. It is a multi-linear hash with a specific choice of $R_1 = \mathbb{Z}_{2^{64}}$ and a post-processor. All these constructions above requires (at least) ℓ multiplications and ℓ many independent key blocks.

By counting roots of a polynomial, one can show that $(m_1, \ldots, m_\ell) \mapsto m_1 \cdot \mathbf{K} + m_2 \cdot \mathbf{K}^2 + \cdots + m_\ell \cdot \mathbf{K}^\ell$ is an $\ell \times q^{-1}$-ΔU hash function. This is known as poly-hash [3,9,45]. Some examples are Ghash used in GCM [22], poly1305 [6], polyQ, polyR [25] (combination of two poly-hashes), and others [18,20] etc. The speed report of these constructions are given in [30,40].

Bernstein-Rabin-Winograd hash or BRW [7,36] hash is a multi-variate polynomial hash which is non-linear in message blocks. It requires $\ell/2$ multiplication and one key. As the algorithm is recursive and binary tree based, it requires $O(\log \ell)$ storage. This construction uses minimum number of keys (single block) and requires minimum number of multiplications (as we show in Theorem 3 of Sect. 5).

3.2 Composition of Universal Hashes

Given an ϵ_1-universal $(\mathcal{K}, D, D')$-hash function h and ϵ_2-ΔU $(\mathcal{K}', D', R_1)$-hash function h' the composition hash function defined below

$$(h' \circ h)_{k,k'}(m) = h'_{k'}(h_k(m)),\ \forall m \in D$$

is $(\epsilon_1 + \epsilon_2)$-$\Delta$U-hash function on D. Whenever h' is assumed to be only ϵ_2-U hash function, the composition is $(\epsilon_1+\epsilon_2)$-U-hash function [40]. This composition results are useful to play with domain and range for different choices and has been used in several constructions [4,25,39].

3.3 Pseudo Dot-Product

The notion of pseudo dot product hash is introduced for preprocessing some cost in matrix multiplications [46]. The construction NMH [14] uses this idea. NMH* and NH are variants of these construction. Later on NH has been modified to propose some more constructions [8,19,21,31]. A general form of pseudo dot-product PDP is $(m_1+\mathbf{K}_1)(m_2+\mathbf{K}_2)+\ldots+(m_{\ell-1}+\mathbf{K}_{\ell-1})(m_\ell+\mathbf{K}_\ell)$ which is same as multi-linear hash plus a function of messages and a function of keys separately. The main advantage of PDP is that, unlike multi-linear hash, it requires $\ell/2$ multiplications to hash ℓ message blocks. We first prove a general statement which is used to prove ΔU property of PDP.

Lemma 1. *Let h be an ϵ-ΔU $(\mathcal{K}, D, R_1)$-hash function where R_1 is an additive group. Then the following $(\mathcal{K}, D, R_1)$-hash function h'*

$$h'_k(m) = h_k(m) + f(k) + g(m). \tag{1}$$

is ϵ-ΔU hash function for any two functions f and g mapping to R_1.

Proof. For any $m \neq m'$ and δ, $h'_k(m) - h'_k(m') = \delta$ implies that $h_k(m) - h_k(m') = \delta' := \delta + g(m') - g(m)$. So for all $m \neq m'$ and δ,

$$\mathsf{Pr}[h'_k(m) - h'_k(m') = \delta] \leq \max_{\delta'} \mathsf{Pr}[h_k(m) - h_k(m') = \delta'] \leq \epsilon$$

and hence the result follows. □

Corollary 1 (Pseudo dot-product hash). *Let R_1 be a field of size q. The hash function $(m_1, m_2, \ldots, m_\ell) \mapsto (m_1 + \mathbf{K}_1)(m_2 + \mathbf{K}_2) + \ldots + (m_{\ell-1} + \mathbf{K}_{\ell-1})(m_\ell + \mathbf{K}_\ell)$ is q^{-1}-ΔU hash function for a fixed ℓ.*

Corollary 2 (Square hash [11]). *Let R_1 be a field of size q. The hash function $(m_1, m_2, \ldots, m_\ell) \mapsto (m_1 + \mathbf{K}_1)^2 + \cdots + (m_\ell + \mathbf{K}_\ell)^2$ is q^{-1}-ΔU hash function for a fixed ℓ.*

3.4 Message Space Extension: Sum Hash Construction

Now we provide some easy generic tools to hash larger message. Let h be an ϵ-ΔU hash function from D to R_1 with key space $\mathcal{K}$. A hash function is called **sum hash** (based on h), denoted h^{sum} if it is defined as

$$h^{\mathrm{sum}}_{k_1,\ldots,k_s}(m_1, \ldots, m_s) = \sum_{i=1}^{s} h_{k_i}(m_i), \tag{2}$$

The multi-linear hash ML and PDP are two examples of sum-hash.

Lemma 2. *If h is an ϵ-ΔU hash function from D to R_1 with key space $\mathcal{K}$ then h^{sum} is an ϵ-ΔU $(\mathcal{K}^s, D^s, R_1)$-hash function.*

Proof. One can verify it in a straightforward manner once we condition all keys $\mathbf{K}_j$'s except a key $\mathbf{K}_i$ for which $m_i \neq m'_i$ (the i^{th} elements of two distinct inputs m and m'). □

Universal Hash for Variable Length. The above sum-hash is defined for a fixed number of message blocks. Now we define a method which works for arbitrary domain $\overline{D} := \{0,1\}^{\leq t}$. To achieve this, we need a padding rule which maps $\overline{D}$ to $D^+ = \cup_{i \geq 1} D^i$. A padding rule $\mathsf{pad} : \overline{D} \to D^+$ is called D'-restricted if it is an injective function and for all $m \in D$ and $\mathsf{pad}(m) = (m_1, \ldots, m_s)$ we have $m_s \in D'$.

Lemma 3 (Extension for $\overline{D} := \{0,1\}^{\leq t}$). *Let h be an ϵ-ΔU $(\mathcal{K}, D, R_1)$-hash function and ϵ-balanced on $D' \subseteq D$ and $\mathsf{pad} : \overline{D} \to D^{\leq L}$ be a D'-restricted padding rule. The sum-hash $h^{\mathsf{pad},\mathrm{sum}}$, defined below, is an ϵ-ΔU $(\mathcal{K}^L, \{0,1\}^{\leq t}, R_1)$-hash function.*

$$h^{\mathsf{pad},\mathrm{sum}}_{K_1,\ldots,K_L}(m) = \sum_{i=1}^{s} h_{K_i}(m_i), \quad \mathsf{pad}(m) = (m_1, \ldots, m_s) \tag{3}$$

The proof is similar to fixed length sum-hash except that for two messages with different block numbers. In this case, the larger message uses an independent key for the last block which is not used for the shorter message and hence the result follows **by using balanced property of the hash**.

Remark 1. Note that ML is clearly not universal hash function for variable length messages. It is a sum hash applied on the hash $m \cdot \mathbf{K}$ which is not balanced on the field R_1 (the 0 message maps to 0 with probability one). However, it is q^{-1}-balanced for $R_1 \setminus \{0\}$. Hence for any padding rule pad which is injective and the last block is not zero will lead to an universal hash for ML construction. For example, the popular "10-padding" pads a bit 1 and then a sequence of zeros, if required, to make it a tuple of the binary field elements. This ensures that the last block has the bit 1 and hence it is non-zero.

The pseudo dot-product is $2q^{-1}$-balanced on R_1 and hence any injective padding rule for PDP will give a $2q^{-1}$-ΔU hash function.

An Alternative Method: Hashing Length. A generic way to handle arbitrary length is as follows: Let h be an ϵ-ΔU hash function on D_i, $1 \le i \le r$ and h' be an ϵ-ΔU hash function on $\{1, 2, \ldots, r\}$. Then the hash function $H_{k,k'}(m) = h_k(m) + h'_{k'}(i)$ where $m \in D_i$ is an ϵ-ΔU hash function on $D := \cup_i D_i$. We apply this approach in our construction to define over arbitrary messages.

3.5 Toeplitz Construction: A Method for Multi-block Hash

One straightforward method to have a d-block universal hash is to apply d independent invocation of universal hash h. More precisely, for d independent keys $\mathbf{K}_1, \ldots, \mathbf{K}_d$, we define a d-block hash as $h^{(d)} = (h_{\mathbf{K}_1}(m), \ldots, h_{\mathbf{K}_d}(m))$. We call it *block-wise hash*. It is easy to see that if h is ϵ-U (or ΔU) then $h^{(d)}$ is ϵ^d-U (or ΔU) hash function. The construction has d times larger key size. However, for a sum-hash h^{sum} we can apply **Toeplitz construction**, denoted $h^{T,d}$, which requires only d additional key blocks where h is an ϵ-ΔU $(\mathcal{K}, D, R)$-hash function.

$$h_i^{T,d}(m_1, \ldots, m_l) = h_{\mathbf{K}_i}(m_1) + h_{\mathbf{K}_{i+1}}(m_2) + \ldots + h_{\mathbf{K}_{l+i-1}}(m_l), \quad 1 \le i \le d. \tag{4}$$

We define $h^{T,d}_{\mathbf{K}_1,\ldots,\mathbf{K}_{l+d-1}}(m) = (h_1^{T,d}, \ldots, h_d^{T,d})$. Note that it requires $d-1$ additional keys than the sum construction for single-block hash. However the number of hash computations is multiplied by d times. Later we propose a better approach for a d-block construction which requires much less multiplications.

Lemma 4. *h is ϵ-ΔU $(\mathcal{K}, D, R_1)$-hash $\Rightarrow$ $h^{T,d}$ is ϵ^d-ΔU $(\mathcal{K}^{l+d-1}, D^l, R_1^d)$-hash.*

Proof. For two distinct messages $m \neq m'$ it must differ at some index. Let i be the first index where they differ i.e., $m_i \neq m'_i$ and $m_1 = m'_1, \ldots, m_{i-1} = m'_{i-1}$. Now condition all keys except $\mathbf{K}' := (\mathbf{K}_i, \ldots, \mathbf{K}_{i+d-1})$. Denote H_i and H'_i for the i^{th} block hash outputs for the messages m and m' respectively. Now, $H_d - H'_d = \delta_d$ leads a differential equation of h for the key $\mathbf{K}_{i+d-1}$ and so this

would contribute probability ϵ. Condition on any such $\mathbf{K}_{i+d-1}$, the previous equation $H_{d-1} - H'_{d-1} = \delta_{d-1}$ can be expressed as an differential equation of $\mathbf{K}_{i+d-2}$ and so on. The result follows once we multiply all these probabilities. □

The above proof argument has similarities in solving a system of upper triangular linear equations. So we start from solving the last equation and once we solve it we move to the previous one and so on until we solve the first one.

Toeplitz Construction Applied to an Arbitrary Length. Now we describe how Toeplitz construction can be used for arbitrary length inputs. If h is ϵ-balanced on a set $D' \subseteq D$ then we need a padding rule which maps a binary string to $(m_1, \ldots, m_s) \in D^s$ such that $m_s \in D'$. This condition is same as sum hash construction for arbitrary length.

Lemma 5 (Toeplitz construction for $\overline{D} := \{0,1\}^{\leq t}$). *Let h be an ϵ-ΔU $(\mathcal{K}, D, R_1)$-hash function and ϵ-regular on $D' \subseteq D$ and pad be D'-restricted. Then the Toeplitz hash $h^{T,d,\mathsf{pad}}(m) = h^{T,d}(\mathsf{pad}(m))$ is an ϵ^d-ΔU $(\mathcal{K}^{L+d-1}, \{0,1\}^{\leq t}, R_1^d)$ hash function.*

The proof is similar to the fixed length proof and hence we skip the proof. The 10-padding rule (as mentioned for ML hash) for Toeplitz construction in ML can be used [38]. Similarly, for PDP one can use any injective padding rule. Generalized linear hash [38], LFSR-based hash [23], CRC construction or Division hash [23,40], Generalized division hash [40], Bucket hash [37], a variant of Toeplitz construction [31] etc. are some other examples of multi-block hash.

4 Our Constructions

4.1 Error-Correcting Coding

Let A be an alphabet. Any injective function $e : D \to A^n$ is called an *encoding function* of length n. Any element in the image of the encoding function is called code word. For any two elements $x = (x_1, \ldots, x_n), y = (y_1, \ldots, y_n) \in A^n$ we define hamming distance $d_{\mathrm{ham}}(x, y) = |\{i : x_i \neq y_i\}|$, the number of places two n-tuples differ. We can extend the definition for arbitrary size. Let $x = (x_1, \ldots, x_n) \in A^n, y = (y_1, \ldots, y_m) \in A^m$ where $m \leq n$. We define $d^*_{\mathrm{ham}}(x, y) = (n - m) + d_{\mathrm{ham}}(x', y)$ where $x' = (x_1, \ldots, x_m)$.

Definition 3. *The minimum distance for an encoding function $e : D \to A^{\leq L} := \cup_{i \leq L} A^i$ is defined as $d(e) \triangleq \min_{M \neq M' \in D} d^*_{\mathrm{ham}}(e(M), e(M'))$.*

We know from coding theory that for any coding $e : A^k \to A^n$ we have $d(e) \leq n - k + 1$ (**singleton bound**). Moreover, there is a linear code[3] e, called MDS or **maximum distance separable** code, such that $d(e) = n - k + 1$. However, if we consider sequence of MDS codes applied to different length the combined coding

[3] There is a generator matrix $G_{k \times n}$ over the field $\mathbb{F}$ such that $e(x) = x \cdot G$.

may have minimum distance only one since we may find two distinct messages M, M' such that $e(M) = (x_1, \ldots, x_m)$ and $e(M') = (x_1, \ldots, x_m, x_{m+1})$. If the generator matrix of MDS code is of the systematic form $G = (I_k : S_{k\times(n-k)})$ with identity matrix I_k then S is called MDS matrix. A characterization of MDS matrix is that every square sub-matrix has full rank. There are some known systematic form of MDS code based on Vandermonde matrix [5]. We call a matrix $S_{k\times(n-k)}$ d-MDS if every square submatrix of size d has full rank. Thus, S is a MDS matrix if and only if it is d-MDS for all $1 \leq d \leq \min\{k, n-k\}$. Now we give examples of MDS and d-MDS matrix using a general form of Vandermonde matrix.

Vandermonde Matrix. We first define a general form of Vandermonde matrix $V_d := V_d(\alpha_1, \ldots, \alpha_n)$ over a finite field $\mathbb{F}_q$ where $d \leq n$ are positive integers and $\alpha_1, \ldots, \alpha_n$ are distinct elements of the field. It is an $d \times n$ matrix whose $(i, j)^{\text{th}}$ entry is α_j^{i-1}. If $n = d$ then the matrix is invertible and it is popularly known as Vandermonde matrix, denoted $V(\alpha_1, \ldots, \alpha_s)$. Moreover note that any r' columns of V_d are linearly independent where $r' \leq d$. In particular, V_d is a d-MDS matrix.

Lemma 6. *The matrix V_d defined above for n distinct elements $\alpha_1, \ldots, \alpha_n$ is d-MDS matrix.*

Proof. Let us take d columns $i_1, \ldots, i_d$ then the submatrix is the Vandermonde matrix of size d with d distinct elements $\alpha_{i_1}, \ldots, \alpha_{i_d}$. As the Vandermonde matrix is invertible the result follows. □

4.2 A General Construction

Let $d \geq 1$ be an integer. Our construction has three basic components.

(1) Let e be an error correcting code from a message space D to $A^{\leq L}$ with the minimum distance d.
(2) Let $h : \mathcal{K} \times A \to R_1$ be an ϵ-ΔU and ϵ-balanced hash function.
(3) For each $l \geq d$, let $V_{d,l}$ be a d-MDS matrix (any d columns are linearly independent) of dimension $d \times l$ whose entries are from R_1.

We define a hash on D which is a composition of three basic steps encoding or expansion, block-wise-Hash and a linear combination. We apply the encoding function e to expand the message $m \in D$ to an l-tuple $(m_1, \ldots, m_l) \in A^l$. In this process we ensure at least d places would differ for two distinct messages. Then we apply the hash h alphabet-wise to obtain $h := (h_1, \ldots, h_l)$. Finally, we apply a linear combiner on the hash blocks to obtain d-block hash output $V_{d,l} \cdot h$. We call this general construction method EHC or encode-hash-combiner. The description of it is given in Algorithm 1.

Theorem 1. *If e has minimum distance d (i.e. $d(e) = d$) and h is ϵ-ΔU and ϵ-balanced function then the extend function H is ϵ^d-Δ universal hash function.*

Input: $m \in D$
Output: $H \in R_1^d$
Key: $(k_1, \ldots, k_L) \in \mathcal{K}^L$

Algorithm $\mathsf{EHC}(m)$
1 $e(m) = (m_1, \ldots, m_l) \in A^l$. \\ Apply encoding function
2 For all $j = 1$ to l \\ Apply hash block-wise.
3 $h_j = h_{k_j}(m_j)$
4 $H = V_{d,l} \cdot (h_1, \ldots, h_l)^{\mathrm{tr}}$ \\ Apply d-MDS combiner .
5 Return H

Algorithm 1. A General Δ-universal hash construction. It uses an error correcting code $e : D \to R_1^+$ with a minimum distance d, a family of d-MDS matrix $V_{d\times l}$, $l \geq d$, and ΔU hash h from A to R_1 with key space $\mathcal{K}$.

Proof. Let $m \neq m'$ and $x = e(m) = (m_1, \ldots, m_l)$, $x' = e(m') = (x'_1, \ldots, x'_l)$. By definition of minimum distance of e, $d^*(e(m), e(m')) \geq d$. W.l.o.g we assume that $l \geq l'$ and $i_1 \leq \ldots \leq i_d \leq l$ are distinct indices at which the encoded messages differ. We condition all keys except $\mathbf{K}_{i_1}, \ldots, \mathbf{K}_{i_d}$. Now for any $\delta \in R_1^d$, $H(m) - H(m') = \delta$ implies that $V \cdot (a_{\mathbf{K}_{i_1}}, \ldots, a_{\mathbf{K}_{i_d}})^{tr} = \delta$ where $a_{\mathbf{K}_{i_j}} = h_{\mathbf{K}_{i_j}}(m_{i_j}) - h_{\mathbf{K}_{i_j}}(m'_{i_j})$ if $i_j \leq l'$ (we use ΔU property), otherwise, $a_{\mathbf{K}_{i_j}} = h_{\mathbf{K}_{i_j}}(m_{i_j})$ (we use balancedness property). Moreover, V is the sub-matrix of $V_{d,l}$ with the columns $i_1, \ldots, i_d$. Note that V is invertible and hence given differential event is equivalent to $(a_{\mathbf{K}_{i_1}}, \ldots, a_{\mathbf{K}_{i_d}})^{tr} = V^{-1} \cdot \delta = \delta'$. Since $\mathbf{K}_{i_j}$'s are independent $\mathsf{Pr}[a_{\mathbf{K}_{i_j}} = \delta'_j] \leq \epsilon$ (because h is ϵ-ΔU hash and ϵ balanced function). So the differential probability of H is at most ϵ^d. □

Remark 2. Note that the only non-linear part in key and message blocks appears in underlying hash computations. As the error correcting code and combiners are linear we only need to apply constant multiplications (which is also a linear function). For appropriate choices of constants, such as primitive element of the field R_1, the constant multiplication is much more efficient compare to non-constant multiplication.

4.3 Specific Instantiations

Specific Instantiations for Fixed Length. Let $d = 4$. Let R_1 be the Galois field of size 2^n and α be a primitive element. Note that R_1^2 can also be viewed as the Galois field of size 2^{2n} and let β be a its primitive element. The following coding function C_4 has minimum distance 4. $C_4(m_1, \ldots, m_t) = (m_1, \ldots, m_t, m_{t+1}, m_{t+2}, m_{t+3})$ where

- $m_{t+1} = \bigoplus_i m_i$,
- $m_{t+2} = \bigoplus_i m_i \beta^{i-1}$ and
- $m_{t+3} = \bigoplus_i m_i \beta^{2(i-1)}$.

Let $l = t+3$. The base hash function $h_{k,k'}(x, x') = (x \oplus k) \cdot (x' \oplus k')$ mapping $R_1^2 \to R_1$ with key space R_1^2. It is the pseudo-dot -product hash. Finally the d-MDS

matrix can be replaced by Vandermonde matrix $V_{d,l} := V_d(1, \alpha, \alpha^2, \ldots, \alpha^l)$ wher α.

Proposition 1. *The coding C_4 defined above has minimum distance 4 over the alphabet R_1^2 for a fixed length encoded message.*

Proof. This coding has systemic form $(I : S)$ where $S = V_l(1, \beta, \beta^2)$. It is known that S is 3-MDS matrix. Now we show that it is also 1-MDS and 2-MDS matrix. Showing 1-MDS matrix is obvious as every entry of the matrix is non-zero. To show that S is 2-MDS we need to choose two columns of the three columns. If we include the first column then the sub-matrix is again a Vandermonde matrix. If we choose the last two columns and i_1 and i_2^{th} rows then the determinant of the sub-matrix is $\beta^{i_1+i_2-2}(\beta^{i_2} - \beta^{i_1})$. □

It is easy to see that if we drop the last column or last two columns we have error correcting code with distance 3 and 2 respectively. Similarly, one can have a specific instantiation with $d = 2$. We do not know so far any coding function for $d > 4$ which can be efficiently computed for any arbitrary length input in an online manner without storing the whole message. However, for short messages one can apply some pre-specified MDS codes. Note that it is not necessary to apply MDS code. However, applying MDS-code make the key size and the number of multiplication as low as possible.

Variable Length ΔU Hash. The above construction works for fixed size input. Note that C_4 does not have minimum distance (with extended definition) four for arbitrary length blocks. However, with the extended definition of distance, we observe that C_4 has minimum distance over $D_0 := \cup_{i \equiv 0 \mod 4} R_1^i$. Similarly, it has minimum distance 4 over D_1, D_2 and D_3. Let $\mathbf{K}^{(1)}, \mathbf{K}^{(2)} \in \mathcal{K}$ be dedicated keys for length, i.e. not used to process message. Now we define our hash function ECH^* for arbitrary length message m as follows.

$$\mathsf{ECH}^*(m) = \mathsf{ECH}(m) + b_1 \cdot \mathbf{K}^{(1)} + b_2 \cdot \mathbf{K}^{(2)}, b_1, b_2 \in \{0, 1\}, l \equiv b_1 + 2b_2 \mod 4$$

where $e(m) = A^l$. Basically, we hash the length of codeword modulo 4. To analyze it works, we consider three cases for $e(m) \in D_j$ and $e(m') \in D_{j'}$.

1. If $j = j'$ then the previous theorem for fixed length works.
2. If $j \neq j'$ then the differential probability will be low due to the hash $b_1 \cdot \mathbf{K}^{(1)} + b_2 \cdot \mathbf{K}^{(2)}$ applied to two different two-bit string (b_1, b_2).

Theorem 2. *If h is an ϵ-ΔU hash function then the construction EHC^* is ϵ^d-ΔU hash function for variable length inputs.*

5 Lower Bound on Multiplications or Non-Linear Operations

A polynomial can be computed through a sequence of addition and multiplication. Moreover, we can combine all consecutive additions to a linear function.

When we multiply the multiplicands can be expressed as linear functions of all the variables and previously computed multiplication results. For example, poly hash $H := m_0 + km_1 + k^2 m_2$ can be computed through the following sequence of computations.

$$v_1 = k \cdot m_2, v_2 = (v_1 + m_1) \cdot k, H = v_2 + m_0.$$

Definition 4 *An* **algebraic computation.** *A is defined by the tuple of linear functions* $(L_1, \ldots, L_{2t}, L^1, \ldots, L^d)$ *where* L_{2i-1} *and* L_{2i} *are linear functions over variables* $m = (m_1, \ldots, m_l), k = (k_1, \ldots, k_s), v_1, \ldots, v_{i-1}$, $1 \le i \le t$ *and* L^i*'s are linear over* m, k *and* $v_1, \ldots, v_t$*. When we identify* v_i *by* $L_{2i-1} \cdot L_{2i}$ *recursively* $1 \le i \le t$, L^i *are multivariate polynomials (MVP). We call* t *the* multiplication complexity *of A.*

We also say that A computes the d-tuple of function $H := (L^1, \ldots, L^d)$. Multiplication complexity of H is defined as the minimum multiplication complexity of all algebraic computation which computes the d polynomials H. Note that while counting multiplication complexity, we ignore the constant multiplications which are required in computing L. This is fine when we are interested in providing lower bounds. However, for a concrete construction, one should clearly mention the constant multiplications also as it could be significant for a large number of such multiplications.

Let R be a ring. A linear function in the variables $x_1, \ldots, x_s$ over R is a function of the form $L(x_1, \ldots, x_s) = a_0 + a_1 x_1 + \ldots + a_s x_s$ where $a_i \in R$. We denote the constant term a_0 by c_L. We also simply write the linear function by $L(x)$ where $x = (x_1, \ldots, x_s)$ is the vector of variables. We add or subtract two vectors coordinate-wise. Note that if $c_L = 0$ then $L(x - x') = L(x) - L(x')$.

Notation. We denote the partial sum $a_1 x_1 + \ldots + a_i x_i$ by $L[x[1..i]]$ where $x[1..i]$ represents $x_1, \ldots, x_i$. If L is a linear function in the vectors of variables x and y then clearly, $L = a_L + L[x] + L[y]$. Now we state two useful lemmas which would be used to prove lower bounds of multiplication complexities of universal hashes.

Lemma 7 [43]. *Let* H *be a* ϵ*-*ΔU *hash function from* S *to* T *then* $\epsilon \ge \frac{1}{|T|}$.

Lemma 8. *Let* R *be a finite ring. Let* $V : \mathcal{K} \times \mathcal{M} \xrightarrow{*} R^t$ *be a hash function and* L *is a linear function on* R^t*. For any functions* f *and* g*, the following keyed function* H

$$H(K, x) = L(V(K, x)) + f(x) + g(K)$$

is ϵ*-*ΔU *hash function if* V *is* ϵ*-*ΔU *hash function. Moreover,* $\epsilon \ge \frac{1}{|R|^t}$.

Proof. By above lemma we have $x \ne x'$ and δ_1 such that $\mathsf{Pr}_K[V(K, x) - V(K, x') = \delta_1] \ge \frac{1}{|T|}$. Let $\delta = L(\delta_1) + (f(x) - f(x'))$ and hence $V(K, x) - V(K, x') = \delta_1 \Rightarrow H(K, x) - H(K, x') = \delta$. This proves the result. □

5.1 Minimum Number of Multiplications for ΔU Hash Function

Now we show our first lower bound on the number of multiplications for a ΔU hash function over a field $\mathbb{F}$ which is computable by addition and multiplication. Clearly, it must be a multivariate polynomial in key and message block and we call it multivariate polynomial or MVP hash function. The theorem shows that a ΔU MVP hash function requiring s multiplications can process at most $2s$ blocks of messages. In other words, any MVP hash function computable in s multiplications processing $2s+1$ message blocks has differential probability one. Intuitive reason is that if we multiply s times then there are $2s$ many linear functions of message m only. Thus, mapping $2s+1$ blocks to $2s$ linear functions would not be injective and hence we can find a collision. The detail follows.

Theorem 3. *Let $H(K, m_1, \ldots, m_l)$ be a MVP hash computable by using s multiplications with $2s+1 \leq l$. Then there are two distinct vectors $a, a' \in \mathbb{F}^l$ and $\delta \in \mathbb{F}$ such that $H(K,a) = H(K,a') + \delta$. for all keys K*

Proof. As H can be computed by s multiplications we have $2s+1$ linear functions $\ell_1, \ell_2, \ldots, \ell_{2s}$ and L such that ℓ_{2i-1} and ℓ_{2i} are linear functions over m, K and $v_1, \ldots, v_{i-1}$ where $v_i = \ell_{2i-1} \cdot \ell_{2i}$. Moreover, L is a linear function over m, K and $v = (v_1, \ldots, v_s)$ with $H = L$. Note that there are $2s$ many linear equations $\ell_i[m]$'s (the partial linear functions on x only) over at least $2s+1$ variables $m_1, \ldots, m_l$, we must have a non-zero solution $\Delta \in \mathbb{F}^l$ of $\ell_i[m]$'s. More precisely, there is non-zero $\Delta \in \mathbb{F}^l$ such that $\ell_i[\Delta] = 0$ for all $1 \leq i \leq 2s$. Let $a \in \mathbb{F}^l$ be any vector and $a' = a + \Delta$. Let us denote $v_i(K,a)$ and $v_i(K,a')$ by v_i and v'_i respectively.

Claim: $v_i = v'_i$ for all $1 \leq i \leq s$.

We prove the claim by induction on i. Note that

$$v_1 = (\ell_1[a] + \ell_1[K] + c_{\ell_1}) \cdot (\ell_2[a] + \ell_2[K] + c_{\ell_2})$$

and similarly for v'_1. We already know that $\ell_1[a] = \ell_1[a']$, $\ell_2[a] = \ell_2[a']$ and hence $v_1 = v'_1$. Suppose the result is true for all $j < i$. Then,

$$v_i = (\ell_{2i-1}[a] + \ell_{2i-1}[K] + \ell_{2i-1}[v_1, \ldots, v_{i-1}] + c_{\ell_i})$$

$$\times(\ell_{2i}[a] + \ell_{2i}[K] + \ell_{2i}[v_1, \ldots, v_{i-1}] + c_{\ell_2})$$

and similarly for v'_i. By using equality $\ell_{2i-1}[a] = \ell_{2i-1}[a']$ and $\ell_{2i}[a] = \ell_{2i}[a']$, and the induction hypothesis $v_1 = v'_1, \ldots, v_{i-1} = v'_{i-1}$ we have $v_i = v'_i$.

Thus, $V : \mathcal{K} \times \mathbb{F}^l \to \mathbb{F}^s$, mapping (K,x) to $(v_1(K,x), \ldots, v_s(K,x))$ has collision probability 1. The hash function $H(K,x)$ is defined as $L[V(K,x)] + L[K] + L[x] + c_L$. So by using Lemma 8 the result follows. □

Corollary 3. *The pseudo dot product hash* PDP *is optimum in number of multiplications.*

Remark 3.

1. The above result holds even if we ignore the cost involving key only, such as stretching the key by using pseudorandom bit generator or squaring the key (it does for BRW hash) etc. Hence the BRW hash is also optimum if key processing is allowed.
2. From the proof one can actually efficiently construct a, a' and δ. We only need to solve $2s$ equations $\ell_i[x]$. By previous remark, the result can be similarly extended if we ignore cost involving message only, e.g., we apply cryptographic hash to message blocks. More precisely, v_i is defined as product of f_i and g_i where $f_i = f_i^1(x) + f_i^2(K) + f_i^3(v_1, \ldots, v_{i-1})$ and similarly g_i. By using non-injectivity of $x \mapsto (f_1, g_1, \ldots, f_s, g_s)$ we can argue that there are distinct a and a' such that the f_i and g_i values for a and a' are same. However, this gives an existential proof of a and a' (which is sufficient to conclude the above theorem).
3. Our bound is applicable when we replace multiplication by any function. More precisely, we have the following result.

Theorem 4. *Let $H(x_1, \ldots, x_l, y_1, \ldots, y_k)$ be a function where x_1, …, x_l, y_1, …, y_k are variables. Let $f_i : \mathbb{F}^k \times \mathbb{F}^{r_i} \to \mathbb{F}$ be some functions, $1 \le i \le m$. Suppose $H(\cdot))$ can be computed by s_i invocations of f_i, $1 \le i \le m$. If $l \ge \sum_i s_i r_i + 1$ then there are two distinct vectors $a = (a_1, \ldots, a_l)$ and $a' = (a'_1, \ldots, a'_l)$ from $\mathbb{F}^l$ and $\delta \in \mathbb{F}$ such that*

$$H(a, y_1, \ldots, y_k) = H(a', y_1, \ldots, y_k) + \delta, \ \forall y_1, \ldots, y_k.$$

The proof is similar to the above theorem and hence we skip.

Now we extend our first theorem to a multi-block hash output, e.g. Toeplitz hash function. So we work in the field $\mathbb{F}$ however, the hash output is an element of $\mathbb{F}^d$ for some $d \ge 1$. Thus, it can be written as $(H_1, \ldots, H_d)$. Again we restrict to those hash functions which can be computed by adding and multiplying (like previous remark, we will allow any processing involving message or key only). So H_i is a MVP hash function and we call H to be d-MVP hash function.

Theorem 5. *Let $H = (H_1, \ldots, H_d)$ be a vector of d polynomials in $m = (m_1, \ldots, m_l)$ and K over a field $\mathbb{F}$ which can be computed by s multiplications. If $l \ge 2(s - r) + 1$ with $r \le d$, then there are $a \ne a'$, elements of $\mathbb{F}^r$ and $\delta \in \mathbb{F}$ such that*

$$\mathsf{Pr}_{\mathbf{K}}[H_{\mathbf{K}}(a) = H_{\mathbf{K}}(a') + \delta] \ge \frac{1}{|\mathbb{F}|^r}.$$

Proof. Suppose H can be computed by exactly s multiplications then we have $2s + d$ linear functions $\ell_1, \ell_2, \ldots, \ell_{2s}$ and $L_1, \ldots, L_d$ such that

(i) ℓ_{2i-1} and ℓ_{2i} are linear functions over m, K and $v_1, \ldots, v_{i-1}$
(ii) $v_i = \ell_{2i-1} \cdot \ell_{2i}$ and
(iii) L_i's are linear functions over x, y and $v = (v_1, \ldots, v_s)$.

Moreover, $H_i = L_i$ for all $1 \leq i \leq d$. The linear functions ℓ_i and L_i can be written as $\ell_i[m] + \ell_i[K] + \ell_i[v] + c_{\ell_i}$ and $L_i[m] + L_i[K] + L_i[v] + c_{L_i}$.

The first $2(s-r)$ many linear equations $\ell_i[m]$'s over at least $2(s-r)+1$ variables. Hence these will have a non-zero solution $\Delta \in \mathbb{F}^l$. Let a be any vector and $a' = a + \Delta$. It is easy to see that $v_i(a, K) = v_i(a', K)$ for all $i \leq s-r$ (similar to proof of Theorem 3). Now consider the mapping $f : \mathbb{F}^k \to \mathbb{F}^r$ mapping

$$K \mapsto (v_{s-r+1}(a, K) - v_{s-r+1}(a', K), \ldots, v_s(a, K) - v_s(a', K)).$$

There must exist $\delta_1 \in \mathbb{F}^r$ such that $\mathsf{Pr}_K[f(K) = \delta_1] \geq \frac{1}{|\mathbb{F}|^r}$. Now we define $\delta = (L_i[M] - L_i[M'] + L_i((0, \ldots, 0, \delta_1))_i$. For this choice of a, a' and δ the result holds. This completes the proof. □

Corollary 4. *The construction* EHC *is optimum when a MDS error correcting code is used. Thus the specific instantiations of* EHC*, given in Sect. 4.3, is optimum for d-block hash outputs,* $2 \leq d \leq 4$.

6 Conclusion and Research Problem

We already know that there is a close connection between error correcting code and universal hash. Here we apply error correcting code and Vandermonde matrix to construct a multi-block universal hash which require minimum number of multiplication. The minimum is guaranteed by showing a lower bound on the number of multiplication required. Previously in different context the lower bound on the number of multiplication has been considered. In this paper for the first time we study "concrete lower bound" (in terms of order a lower bound was known) for universal hash function. Similar lower bound was known for computations of polynomial of specific forms. See Appendix for a brief survey on it. However, we would like to note that those results can not be directly applicable as the contexts are differ ent.

To get a lower bound we take the popular algebraic computation model in which the time of multiplications are separated. We try to equate all the linear functions which are multiplied. Our construction performs better than Toeplitz construction in terms of number of multiplication.

This paper studies the relationship between complexity and security of universal hash. There are some relationship known for complexity and key-size however the picture is incomplete. Moreover, nothing is known involving these three important parameters: (i) security level, (ii) complexity, and (iii) key size. This could be possible future research direction in this topic. Our construction optimizes d block hash output for sum hash functions. It would be interesting to see how one adopts this for multi block polynomial hash using few keys.

In the view of the performance, the ongoing future research of us is to have a lightweight implementation of the universal hash function.

Acknowledgement. This work was supported by the Centre of Excellence in Cryptology, Indian Statistical Institute, Kolkata. The author would like to thank the anonymous reviewers for their useful comments. The author would like to thank Professor Palash Sarkar for his motivating discussion.

References

1. Belaga, G.E.: Some problems in the computation of polynomials. Doklady Akademii Nauk SSSR **123**, 775–777 (1958). ISSN 00023264. Citations in this document: §6
2. Bellare, M.: New proofs for NMAC and HMAC: security without collision-resistance. In: Dwork, C. (ed.) CRYPTO 2006. LNCS, vol. 4117, pp. 602–619. Springer, Heidelberg (2006)
3. Bierbrauer, J., Johansson, T., Kabatianskii, G.A., Smeets, B.J.M.: On families of hash functions via geometric codes and concatenation. In: Stinson, D.R. (ed.) CRYPTO 1993. LNCS, vol. 773, pp. 331–342. Springer, Heidelberg (1994)
4. Black, J., Halevi, S., Krawczyk, H., Krovetz, T., Rogaway, P.: UMAC: fast and secure message authentication. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 216–233. Springer, Heidelberg (1999). Citations in this document: §1, §1.1
5. Blaum, M., Bruck, J., Vardy, A.: MDS array codes with independent parity symbols. IEEE Trans. Inf. Theor. **42**(2), 529–542 (1996). Citations in this document: §4.1
6. Bernstein, D.J.: The Poly1305-AES message-authentication code. In: Gilbert, H., Handschuh, H. (eds.) FSE 2005. LNCS, vol. 3557, pp. 32–49. Springer, Heidelberg (2005). Citations in this document: §1, §3.1
7. Bernstein, D.J.: Polynomial evaluation and message authentication. URL: http://cr.yp.to/papers.html#pema. ID b1ef3f2d385a926123e 1517392e20f8c (2007)
8. Boesgaard, M., Christensen, T., Zenner, E.: Badger – a fast and provably secure MAC. In: Ioannidis, J., Keromytis, A.D., Yung, M. (eds.) ACNS 2005. LNCS, vol. 3531, pp. 176–191. Springer, Heidelberg (2005)
9. den Boer, B.: A simple and key-economical unconditional authentication scheme. J. Comput. Secur. **2**, 65–71 (1993)
10. Carter, L., Wegman, N.M.: Universal classes of hash functions. J. Comput. Syst. Sci. **18**(2), 143–154 (1979)
11. Etzel, M., Patel, S., Ramzan, Z.: Square hash: fast message authentication via optimized universal hash functions. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 234–251. Springer, Heidelberg (1999). Citations in this document: §2
12. Eve, J.: The evaluation of polynomials. Numer. Math. **6**, 17–21 (1964). Citations in this document: §6
13. Gilbert, N.E., MacWilliams, F.J., Neil, J.A.: Sloane, Codes which detect deception. Bell Syst. Tech. J. **53**, 405–424 (1974)
14. 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). Citations in this document: §1.1, §1.1, §3.1, §3.3
15. Hastad, J., Impagliazzo, R., Levin, L.A., Luby, M.: Construction of pseudorandom generators from any one-way function. SIAM J. Comput. **28**(4), 1364–1396 (1999)
16. Horner, W.G.: Philosophical transactions. Roy. Soc. Lond. **109**, 308–335 (1819). Citations in this document: §6

17. Ishai, Y., Kushilevitz, E., Ostrovsky, R., Sahai, A.: Cryptography with constant computational overhead. STOC **2008**, 433–442 (2008). Citations in this document: §1.1, §1.1
18. Johansson, T.: Bucket hashing with a small key size. In: Fumy, W. (ed.) EUROCRYPT 1997. LNCS, vol. 1233, pp. 149–162. Springer, Heidelberg (1997)
19. Kaps, J., Yüksel, K., Sunar, B.: Energy scalable universal hashing. IEEE Trans. Comput. **54**(12), 1484–1495 (2005)
20. Kohno, T., Viega, J., Whiting, D.: CWC: a high-performance conventional authenticated encryption mode. In: Roy, B., Meier, W. (eds.) FSE 2004. LNCS, vol. 3017, pp. 408–426. Springer, Heidelberg (2004)
21. Krovetz, T.: Message authentication on 64-bit architectures. In: Biham, E., Youssef, A.M. (eds.) SAC 2006. LNCS, vol. 4356, pp. 327–341. Springer, Heidelberg (2007)
22. McGrew, D.A., Viega, J.: The security and performance of the Galois/Counter mode (GCM) of operation. In: Canteaut, A., Viswanathan, K. (eds.) INDOCRYPT 2004. LNCS, vol. 3348, pp. 343–355. Springer, Heidelberg (2004). Citations in this document: §3.1
23. Krawczyk, H.: LFSR-based hashing and authentication. In: Desmedt, Y.G. (ed.) CRYPTO 1994. LNCS, vol. 839, pp. 129–139. Springer, Heidelberg (1994). Citations in this document: §2, §3.5
24. Knuth, D.: The Art of Programming. Addison-Wesley, Boston (1969). Citations in this document: §1, §6
25. Krovetz, T., Rogaway, P.: Fast universal hashing with small keys and no preprocessing: the PolyR construction. In: Won, D. (ed.) ICISC 2000. LNCS, vol. 2015, pp. 73–89. Springer, Heidelberg (2001). Citations in this document: §3.1
26. Mansour, Y., Nissan, N., Tiwari, P.: The computational complexity of universal hashing. In: Proceedings of the Twenty Second Annual ACM Symposium on Theory of Computing (1990), pp. 235–243. ACM Press (1990) Citations in this document: §1.1
27. MacWilliams, F.J., Sloane, N.J.A.: The Theory of Error-Correcting Codes. Elsevier, North-Holland (1977). Citations in this document: §1.1
28. Mehlhorn, K.: On the program size of perfect and universal hash functions. In: FOCS, pp. 170–175 (1982)
29. Motzkin, T.S.: Evaluation of polynomials and evaluation of rational functions. Bull Xmer. Math SOC. **61**, 163 (1955). Citations in this document: §6, §6
30. Nevelsteen, W., Preneel, B.: Software performance of universal hash functions. In: Stern, J. (ed.) EUROCRYPT 1999. LNCS, vol. 1592, p. 24. Springer, Heidelberg (1999)
31. Nguyen, L.H., Roscoe, A.W.: Short-output universal hash functions and their use in fast and secure data authentication. In: Canteaut, A. (ed.) FSE 2012. LNCS, vol. 7549, pp. 326–345. Springer, Heidelberg (2012). Citations in this document: §3.5
32. Nisan, N., Zuckerman, D.: Randomness is linear in space. J. Comput. Syst. Sci. **52**(1), 43–53 (1996)
33. Ostrowski, A.M.: On two problems in abstract algebra connected with Homers rule. Studies Presented to R. von Mises. Academic Press, New York (1954). Citations in this document: §6
34. Ya, V.: Pan, Methods of computing values of polynomials. Russ. Math. Surv. **21**, 105–136 (1966). Citations in this document: §6, §6, §6
35. Paterson, M.S., Stockmeyer, L.J.: On the number of nonscalar multiplications necessary to evaluate polynomials. SIAM J. Comput. **2**(1), 60–66 (1973). Citations in this document: §6, §6

36. Rabin, M.O., Winograd, S.: Fast evaluation of polynomials by rational preparation. Commun. Pure Appl. Math. **XXV**, 433–458 (1972). Citations in this document: §6
37. Rogaway, P.: Bucket hashing and its application to fast message authentication. J. Cryptology. **12**(2), 91–115 (1999). Citations in this document: §2, §3.5
38. Sarkar, P.: A new multi-linear universal hash family. Des. Codes Crypt. **69**, 1–17 (2012). Springer. Citations in this document: §3.5, §3.5
39. Sarkar, P.: A trade-off between collision probability and key size in universal hashing using polynomials. Des. Codes Crypt. **3**, 271–278 (2011)
40. 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). Citations in this document: §3.2, §3.5
41. Spielman, D.A.: Linear-time encodable and decodable error-correcting codes. STOC 1995 **4**(4), 388–397 (1995). Citations in this document: §1.1
42. Stinson, D.: Universal hashing and authentication codes. Des. Codes Crypt. **4**(4), 369–380 (1994)
43. Stinson, D.R.: On the connections between universal hashing, combinatorial designs and error-correcting codes. Proc. Congressus Numerantium **114**, 7–27 (1996). Citations in this document: §2, §2, §7
44. Stinson, D.R.: Universal Hash Families and the Leftover Hash Lemma, and Applications to Cryptography and Computing. Research report (University of Waterloo. Faculty of Mathematics) (2001). http://books.google.co.in/books?id=edldNAEACAAJ Citations in this document: §1
45. Taylor, R.: An integrity check value algorithm for stream ciphers. In: Stinson, D.R. (ed.) CRYPTO 1993. LNCS, vol. 773, pp. 40–48. Springer, Heidelberg (1994)
46. Winograd, S.: A new algorithm for inner product. IEEE Trans. Comput. **17**, 693–694 (1968). Citations in this document: §1.1, §3.3
47. Winograd, S.: On the number of multiplications necessary to compute certain functions. Commun. Pure Appl. Math. **XXIII**, 165–179 (1970). Citations in this document: §1, §6
48. Winograd, S.: On the number of multiplications required to compute certain functions. Proc. National Acad. Sci. USA **58**(5), 1840–1842 (1967)
49. Wegman, M., Carter, L.: New hash functions and their use in authentication and set equality. J. of Comp. Syst. Sci. **1981**(22), 265–279 (1981)

Appendix: Brief Survey on the Computation of $a_0 + a_1x + \ldots + a_nx^n$

We provide a brief survey on the lower bound of multiplications for computing a polynomial. Note that our interest in this paper is to provide a lower bound on number of multiplications for computing a multi variate polynomial which is an universal hash. Even though these two issues are very much related (some of the ideas in proving results are also similar), some immediate differences can be noted. For example, the existing bounds depend on the degree of the polynomials whereas we provide bound on the number of message blocks (degree could be arbitrarily higher). The existing works consider multivariate polynomials which has a special form: $P(a_0, \ldots, a_n, x_1, \ldots, x_m) := a_0 + \sum_{i=1}^{n} a_i \cdot \Phi_i(x_1, x_2, \ldots, x_m)$ where

Φ_i's are rational functions of $x_1, \ldots, x_m$. For an universal hash, the lower bound of our paper works for any multivariate polynomial (or even rational functions).

The function x^n can be computed in at most $2\lceil \log_2 n \rceil$ multiplications by using well known "square and multiply" algorithm. One can also compute $1 + x + \ldots + x^{n-1}$ using at most $2\lceil \log_2 n \rceil$ multiplications, one division and two subtractions since it is same as $\frac{x^n-1}{x-1}$ whenever $x \neq 1$. These are some simple examples of polynomials and there are some specific methods to simplify some polynomials. How does one can compute "*generically*" an arbitrary polynomial $f(x) = a_0 + a_1 x + \ldots + a_n x^n$, $a_i \in R$ (an underlying ring or field), of degree n with minimal number of operations, mainly multiplication and division? By generically we mean an algorithm which takes any a_i's and x as its inputs and computes the polynomial $f(x)$ (similar to an algorithm in uniform model). We know Horner's rule [16][4] to compute $f(x)$ in n multiplications and n additions.

§Minimum Number of Multiplications. Can we do better than n multiplications for computing an arbitrary polynomial? Or, can we prove that there are some polynomials for which n multiplications and division are necessary? The above question regarding the minimum number of multiplications to compute a given polynomial of small degree, was first investigated by Ostrowski [33]. He showed that at least n multiplications are required to evaluate a polynomial $f(x)$ of degree n for $1 \leq n \leq 4$. The results were further proved for any positive integer n by Pan [34] and a more general statement by Winograd [47]. Moreover, even if divisions are allowed, at least n multiplications/divisions are necessary to evaluate it. Belega [1] moreover proved that at least n additions or subtractions are required to compute f.

§General statement. The general statement by Winograd gives a lower bound for computation of any multivariate polynomial of the form

$$P(a_0, \ldots, a_n, x_1, \ldots, x_m) := a_0 + \sum_{i=1}^{n} a_i \cdot \Phi_i(x_1, x_2, \ldots, x_m)$$

where Φ_i's are rational functions of $x_1, \ldots, x_m$. If the rank (the maximum number of linear independent elements) of the set $S = \{1, \Phi_1, \ldots, \Phi_n\}$ is $u + 1$ then at least u multiplication and division are necessary. In particular, when $m = 1$, $\Phi_i(x_1) = x_1^i$ we have $P = f(x_1)$ and $u = n$. Thus, the result of Pan [34] is a simple corollary of it. When $m = n$, $\Phi_i(x_1, \ldots, x_n) = x_i$ and $a_0 = 0$ we have the classical dot-product $a_1 \cdot x_1 + \ldots + a_n \cdot x_n$ and the rank is again $n + 1$. So it also proves that to compute the dot-product we need at least n multiplications.

§Evaluation of a given Polynomial with Preprocessing. In the above results all types of multiplications are counted. More formally, the computation of the multivariate polynomial $F(a_0, \ldots, a_n, x) = a_0 + a_1 x + \ldots + a_n x^n$ have been considered in which coefficients are treated as variables or inputs of algorithms.

[4] Around 1669, Isaac Newton used the same idea which was later known as Newton's method of root finding (see 4.6.4, page 486 of [24]).

One of the main motivations of the above issue is to evaluate approximation polynomials of some non-algebraic functions, such as trigonometric functions. As the polynomials (i.e., a_i's) are known before hand, one can do some preprocessing or adaptation on coefficients to reduce some multiplications. To capture this notion, one can still consider the computation of F but the operations involving only a_i's are said to be the preprocessing of a_i's. Knuth [24] (see Theorem E, 4.6.4), Eve [12], Motzkin [29] and Pan [34] provide methods for F requiring $\lceil \frac{n}{2} \rceil$ multiplications ignoring the cost of preprocessing. However, these require preprocessing of *finding roots of higher degree equations* which involves a lot of computation and may not be exact due to numerical approximation. However, it is an one-time cost and is based on only coefficients. Later on, whenever we want to compute the polynomial for a given x, it can be computed faster requiring about $\lceil \frac{n}{2} \rceil$ multiplications. Rabin-Winograd [36] and Paterson-Stockmeyer [35] provide methods which require *rational preprocessing on coefficients (i.e., computing rational functions of coefficients only)* and afterwards about $\frac{n}{2} + O(\log n)$ multiplications for a given x.

§Minimum Number of Multiplications after Preprocessing. We have already seen that total n multiplication is necessary to compute F generically and Horner's rule is one algorithm which shows the tightness of the lower bound. Similarly, with preprocessing, $\lceil n/2 \rceil$ **multiplications for computing the multivariate polynomial** F **has been proved to be optimum** by Motzkin [29] and later on a more general statement by Winograd [47,48]. The bound $\lceil n/2 \rceil$ does not work for computing a known polynomial f since multiplication by constant could be replaced by addition, e.g. in $\mathbb{Z}$, $a_i \cdot x = x + \ldots + x$ (a_i times). In fact, Paterson and Stockmeyer [35] provided **methods which require about** $O(\sqrt{n})$ **multiplications** and **showed the bound is optimum**. Note that this method does not compute the polynomial generically which means that for every polynomial $f(x) = a_0 + a_1 x + \ldots + a_n x^n$ there is an algorithm $C_{a_0,\ldots,a_n}$ depending on the coefficients which computes $f(x)$ given x in $O(\sqrt{n})$ multiplications. This result and those by [29,36,47,48] (one algorithm works for F, i.e. for all polynomials f) can be compared with non-uniform and uniform complexity of Turing machine respectively. This justifies two different bounds of computation of a polynomial.

Collision Attack on 5 Rounds of Grøstl

Florian Mendel[1](✉), Vincent Rijmen[2], and Martin Schläffer[1]

[1] IAIK, Graz University of Technology, Graz, Austria
florian.mendel@iaik.tugraz.at
[2] Department ESAT/COSIC, KU Leuven and Security Department, iminds, Ghent, Belgium

Abstract. In this article, we describe a novel collision attack for up to 5 rounds of the Grøstl hash function. This significantly improves upon the best previously published results on 3 rounds. By using a new type of differential trail spanning over more than one message block we are able to construct collisions for Grøstl-256 on 4 and 5 rounds with complexity of 2^{67} and 2^{120}, respectively. Both attacks need 2^{64} memory. Due to the generic nature of our attack we can even construct meaningful collisions in the chosen-prefix setting with the same attack complexity.

Keywords: Hash functions · SHA-3 candidate · Grøstl · Collision attack

1 Introduction

In the last few years the cryptanalysis of hash functions has become an important topic within the cryptographic community. Especially the collision attacks on the MD4 family of hash functions (MD5, SHA-1) have weakened the security assumptions of these commonly used hash functions [26–28]. As a consequence NIST has decided to organize a public competition in order to design a new hash function, leading to the selection of Keccak as SHA-3 [19].

During the SHA-3 competition, the three classical security requirements (collision-, preimage- and second-preimage resistance) were not the main target of cryptanalytic attacks. Most results were published on building blocks such as the compression function, block cipher or permutation used in a hash function. Additionally, many distinguishers on these building blocks with minor relevance in practice were considered. Although these results are important from a theoretical point of view, vulnerabilities that can be exploited for the hash function are certainly more important.

In this work, we present new results on the collision resistance of the SHA-3 finalist Grøstl. Grøstl is an iterated hash function based on design principles very different from those used in the MD4 family. The compression function of Grøstl is built from two different permutations that follow the design strategy of the Advanced Encryption Standard (AES) [3,17]. The simple construction of the compression function and the byte-oriented design of Grøstl facilitates the security analysis. In the last years Grøstl has received a large amount of cryptanalysis. However, most of the analysis focus on the building blocks of Grøstl and only a few results have been published for the hash function so far.

C. Cid and C. Rechberger (Eds.): FSE 2014, LNCS 8540, pp. 509–521, 2015.
DOI: 10.1007/978-3-662-46706-0_26

Related Work. Grøstl is one of the SHA-3 candidates that has probably received the largest amount of cryptanalysis during the competition. Security analysis of Grøstl was initiated by the design team itself which led to the rebound attack [15]. Since then, several improvements to the rebound attack technique have been made, leading to new results on both the hash function [16] and its underlying components [6,14,22]. For the final round, Grøstl was been tweaked to thwart internal differential attacks [7,21] and to reduce the impact of the rebound attack and its extensions.

The best published attacks on the final version of both Grøstl-256 and Grøstl-512 are collision attacks on 3 rounds of the hash function and on 6 rounds of the compression function [9,23]. Preimage attacks for the compression function of Grøstl-256 and Grøstl-512 have been shown in [29] for 5 and 8 rounds, respectively. Additionally, non-random properties of the Grøstl permutation have been discussed in [1,8]. For a detailed overview of the existing attacks on Grøstl we refer to the ECRYPT II SHA-3 Zoo [4].

Our Contribution. By using a new type of differential trail we are able to show collision attacks on Grøstl for up to 5 rounds. The extension becomes possible by considering differential trails spanning over more than one message block to iteratively cancel differences in the chaining variable. Our new attack combines ideas of the attack on SMASH [13] with the rebound attack [15] on Grøstl. A similar approach has also been used in the attack on Grindahl [20]. The results are collision attacks on the Grøstl-256 hash function reduced to 4 and 5 rounds with a complexity of 2^{67} and 2^{120}, respectively. Both attacks have memory requirements of 2^{64}. Note that the best previously known collision attack on the Grøstl hash function was on 3 rounds with a complexity of 2^{64} [23]. We want to note that the same attack also applies to 5 rounds of Grøstl-512.

Additionally, we show that due to the generic nature of our attack we can construct collisions in the chosen-prefix setting with the same complexity. It has been demonstrated in [24,25] that chosen-prefix collisions can be exploited to construct colliding X.509 certificates and a rogue CA certificate for MD5. Note that in most cases constructing such collisions is more complicated than constructing (random) collisions. Our results and related work for the Grøstl hash function are shown in Table 1.

Outline. The paper is structured as follows. In Sect. 2, we give a short description of the Grøstl hash function. The basic attack strategy and the collision

Table 1. Collision attacks on the Grøstl-256 hash function.

Rounds	Complexity	Memory	Reference
3	2^{64}	-	[23]
4	2^{67}	2^{64}	This work
5	2^{120}	2^{64}	This work

attack for 4 rounds of the hash function is presented in Sect. 3. In Sect. 4, we describe the extension of the attack to 5 rounds, and in Sect. 5 the construction of meaningful collisions is discussed. Finally, we conclude in Sect. 6.

2 Short Description of Grøstl

The hash function Grøstl [5] was one of the 5 finalists in the SHA-3 competition [18]. Grøstl is an iterated hash function with a compression function built from two distinct permutations P and Q, which are based on the AES design principles. In the following, we describe the components of the Grøstl hash function in more detail.

2.1 The Hash Function

The two main variants, Grøstl-256 and Grøstl-512 are used for hash output sizes of $n = 256$ and $n = 512$ bits. The hash function first pads the input message M and splits the message into blocks $m_1, m_2, \ldots, m_t$ of ℓ bits with $\ell = 512$ for Grøstl-256, and $\ell = 1024$ for Grøstl-512. The message blocks are processed via the compression function $f(h_{i-1}, m_i)$ and output transformation $\Omega(h_t)$. The size of the chaining value h_i is ℓ bits as well.

$$\begin{aligned} h_0 &= IV \\ h_i &= f(h_{i-1}, m_i) \quad \text{for } 1 \leq i \leq t \\ h &= \Omega(h_t). \end{aligned}$$

The compression function f is based on two ℓ-bit permutations P and Q (sometimes denoted by P_ℓ and Q_ℓ) and is defined as follows:

$$f(h_{i-1}, m_i) = P(h_{i-1} \oplus m_i) \oplus Q(m_i) \oplus h_{i-1}.$$

The output transformation Ω is applied to h_t to give the final hash value h of size n, where $\text{trunc}_n(x)$ discards all but the least significant n bits of x:

$$\Omega(h_t) = \text{trunc}_n(P(h_t) \oplus h_t).$$

2.2 The Permutations P and Q

The two permutations P and Q are designed according to the wide trail strategy [2] and their structure is very similar to the AES. In Grøstl-256 each permutation updates an 8×8 state of 64 bytes in 10 rounds. In one round, the round transformation updates the state by means of the sequence of transformations

$$\mathsf{MB} \circ \mathsf{SH} \circ \mathsf{SB} \circ \mathsf{AC}\,.$$

In the following, we briefly describe the round transformations of P and Q used in the compression function f in more detail.

AddRoundConstant (AC). In this transformation, the state is modified by combining it with a round constant with a bitwise `xor` operation. Different constants are used for the permutations P and Q.

SubBytes (SB). The SubBytes transformation is the same for P and Q and is the only non-linear transformation of the permutations. It is a permutation consisting of an S-box applied to each byte of the state. The 8-bit S-box is the same as in the AES with good cryptographic properties against differential and linear attacks. For a detailed description of the S-box, we refer to [17].

ShiftBytes (SH). The ShiftBytes transformation is a byte transposition that cyclically shifts the rows of the state over different offsets. The ShiftBytes transformation is different for the two permutations P and Q.

MixBytes (MB). The MixBytes transformation is a permutation operating on the state column by column. To be more precise, it is a left-multiplication by an 8×8 MDS matrix over $\mathbb{F}_{2^8}$. The coefficients of the matrix are determined in such a way that the *branch number* of MixBytes (the smallest nonzero sum of active input and output bytes of each column) is 9, which is the maximum possible for a transformation with these dimensions. This transformation is the same for both permutations P and Q.

2.3 Alternative Description of `Grøstl`

To simplify the description of attack in the following sections, we use an equivalent alternative description of `Grøstl`. Let P' and Q' denote the permutation P and Q without the last application of MixBytes. Then, by setting

$$\begin{aligned} h'_0 &= \mathsf{MB}^{-1}(\mathrm{IV}) \\ h'_i &= P'(\mathsf{MB}(h'_{i-1}) \oplus m_i) \oplus Q'(m_i) \oplus h'_{i-1} \quad \text{for } 1 \le i \le t \\ h &= \Omega(\mathsf{MB}(h'_t)) \end{aligned}$$

with $h_i = \mathsf{MB}(h'_i)$, we get an equivalent description of `Grøstl`, where the last MixBytes transformation of the permutations has been swapped with the XOR operation of the feed-forward.

3 Collision Attack for 4 Rounds of `Grøstl`

To get improved attacks on the `Grøstl` hash function, we view `Grøstl` as a strengthened variant of SMASH [10]. The essential difference between the two designs is that `Grøstl` employs a second nonlinear permutation Q, where SMASH employs a scaling by the constant θ, i.e. a linear map (see Fig. 1). The hash function SMASH has been broken by subsequently controlling the output difference of the compression function using the linearity of θ. After the application of 257 respectively 513 message blocks, a colliding output difference can be constructed [13]. In this section, we show how to achieve the same for 4 rounds of `Grøstl` by having differences in only one permutation.

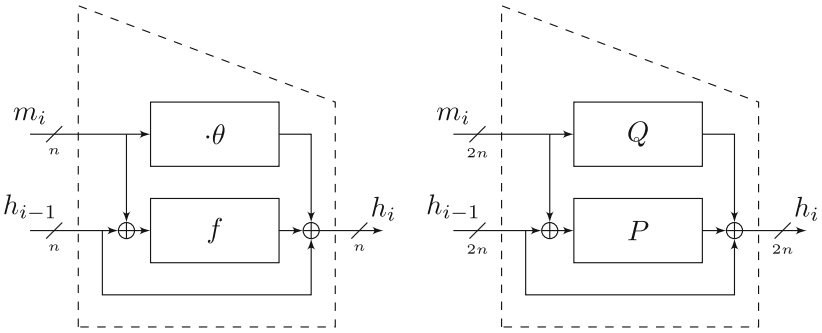


Fig. 1. The compression function of (a) SMASH and (b) Grøstl.

3.1 The Second-Preimage Attack on SMASH

The second-preimage attack on SMASH presented in [13] is based on the technique of *controllable output differences* for the compression function. By carefully selecting consecutive message blocks, an attacker can step-by-step convert an arbitrary starting difference in the chaining variable into an arbitrary output difference. The attack is deterministic and the number of consecutive controllable message blocks is equal to the length of the chaining variable. The nonlinearity of f is made ineffective by strictly controlling its input differences and values. Controlling the input values of f implies that the input values of the linear map are determined. Fortunately, for a linear map it suffices to know the input *difference* to compute the output difference. The output difference of the linear map is controlled by the number of message blocks.

3.2 Application to Reduced Grøstl

At the first sight, the attack on SMASH does not apply to Grøstl, because the strong nonlinearity of P and Q makes it difficult to control the output differences of both permutations. However, by having no differences in Q, we can use the whole freedom of the message block to control the differential propagation in P. Since we cannot control the differences completely, we need to apply a variation of the technique on SMASH, to get a zero output difference at the compression function.

Our attack will start from an arbitrary difference in the chaining variable and convert it into an output difference equal to zero after 9 steps. The first message block can be selected arbitrarily. The only requirement is a difference in the message. The next 8 message blocks are fully controlled by the attacker and must not contain any differences. Then, each of the 8 message blocks is used to cancel one eighth of the differences at the output of the compression function to result in a collision at the end (see Fig. 2).

3.3 Details of the Attack

To simplify the description of the attack we use the alternative description of Grøstl given in Sect. 2.3. Since the last MixBytes transformation is moved out of the compression function, the limited set of differences at the output are more clearly visible.

The core of our collision attack on the reduced hash function are truncated differential trails with only 8 active bytes at the output of P'. Two full active states are placed at the beginning and the number of active bytes for the 4-round trail are given as follows:

$$64 \xrightarrow{r_1} 64 \xrightarrow{r_2} 8 \xrightarrow{r_3} 8 \xrightarrow{r_4} 8. \tag{1}$$

For such a truncated trail, we can construct a pair following the trail with an amortized complexity of 1 (even for a given input differences). We postpone the detailed explanation how to do so until Sect. 3.4.

The high-level overview of the 4-round attack is shown in Fig. 2. In each iteration, the differences in 8 bytes are canceled. Since this has a probability 2^{-64}, we need to compute 2^{64} pairs for P' (for the given input differences) to find a *right* pair that result in the desired output difference. The attack can then be summarized as follows:

1. Choose arbitrary message blocks m_1, m_1^* and compute h'_1. Repeat this until one gets a full active state in h'_1. Note that randomly selected m_1, m_1^* produce a full active state in h'_1 with probability at least 3/4.
2. Use a *right* pair for P' following the trail of (1) to cancel 8 bytes of the difference in the state h'_2, cf. Fig. 2.
3. Use a *right* pair for P' for a rotated variant of the trail of (1) to cancel another 8 bytes of the difference in the state h'_3.
4. Repeat steps 3–4 in total 8 times until a collision has been found in h'_9.

The complexity of the attack is 8 times finding a *right* pair for P' to iteratively cancel the difference in the state $h'_2, \ldots, h'_9$. We will show in the following section that such a *right* pair can be constructed with complexity of 2^{64}, resulting in a total attack complexity of $8 \cdot 2^{64} = 2^{67}$.

3.4 Finding a Right Pair for P'

In this section, we show how to find a *right* pair for P' reduced to 4 rounds following the truncated differential trail in (1) using the rebound attack [15]. Note that the input difference is fixed by $\mathsf{MB}(h'_{i-1})$ and we target an output difference such that 8 bytes of the difference in h'_i can be canceled. Unlike the classical rebound attack, the inbound phase is placed at the beginning and covers the first two rounds, while the outbound phase covers the last two rounds.

Using super-box matches [6,11,12], we can find 2^{64} pairs (solutions) for the inbound phase with a complexity of 2^{64} in time and memory. In the outbound phase, all these pairs will follow the 4 round truncated differential trail with a

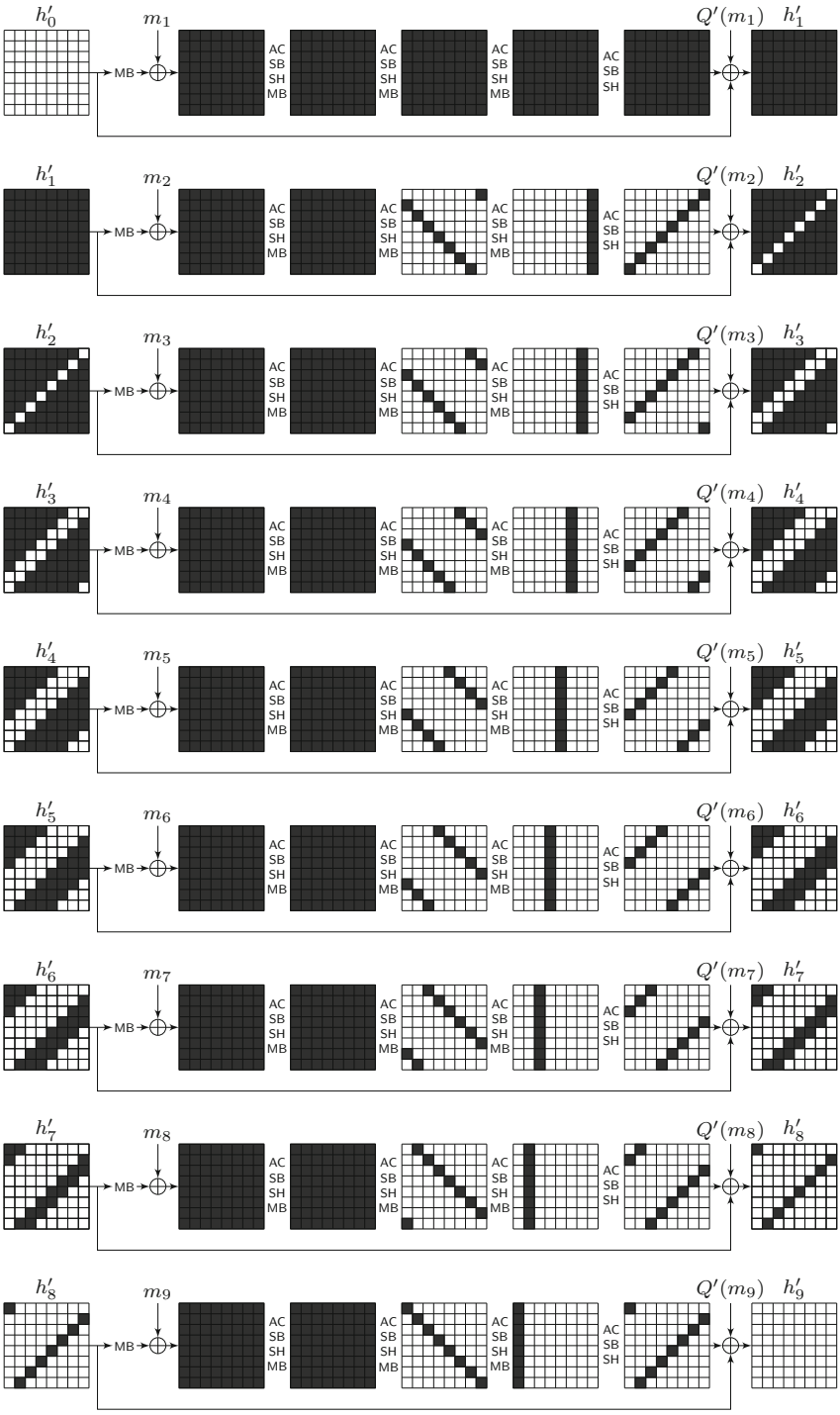


Fig. 2. Overview of the attack on 4 rounds.

probability of 1 and one of these 2^{64} pairs will match the desired output difference (condition on 64 bits). In other words, using the rebound attack we can find a *right* pair for P' with a complexity of 2^{64} in time and memory.

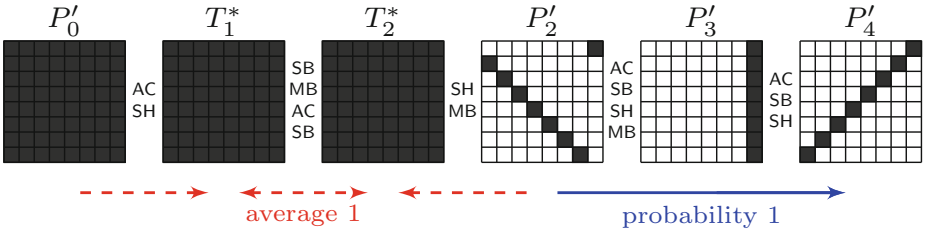


Fig. 3. Truncated differential trail for P' used in the attack on 4 rounds.

In order to make the subsequent description of the rebound attack easier, we swap the SubBytes and ShiftBytes transformation in the first round of permutation P' (see Fig. 3). Note that this can always be done without affecting the output of the round. Then, the attack can be summarized as follows:

1. Compute the input difference of the permutation (P'_0) forward to state T^*_1.
2. Compute all 2^{64} differences of state P'_2 backward to state T^*_2 and store them in a list L.
3. Connect the single difference of state T^*_1 with the 2^{64} differences of state T^*_2 using independent super-box matches. For each column $c = \{0, 1, \ldots, 7\}$ we proceed as follows:
   (a) Take all 2^{64} values for column c of state T^*_1 and compute both values and differences forward to column c of state T^*_2.
   (b) Check for matching 8-byte column differences in list L. Since we compute 2^{64} differences forward and have 2^{64} entries in L, we get 2^{64} solutions (differences and values) for the match. We update L to contain these 2^{64} solutions.
4. For each column and thus, for the whole inbound phase the number of resulting solutions is 2^{64}. The total complexity is 2^{64} in time and memory.

Since the truncated differential trail in the outbound part (the last 2 rounds) has probability 1, we get in total 2^{64} pairs following the truncated differential trail and one of these pairs is expected to be a right pair, i.e. result in the desired output difference (condition on 64 bits).

4 Extending the Attack to 5 Rounds

In this section, we present a collision attack for the `Grøstl`-256 hash function reduced to 5 rounds with a complexity of about 2^{120} and memory requirements of 2^{64}. The attack is an extension of the attack on 4 rounds. However, since

the freedom in finding right pairs for the 5-round trail is limited, we need more message blocks for the attack to succeed. In the attack, we use the following sequence of active bytes in P' (cf. Fig. 4):

$$64 \xrightarrow{r_1} 64 \xrightarrow{r_2} 8 \xrightarrow{r_3} 1 \xrightarrow{r_4} 8 \xrightarrow{r_5} 8. \tag{2}$$

However, it is important to note that for this truncated differential trail (with a fixed input difference) only 2^8 pairs exist, in contrast to 2^{64} for the 4 round trail. This complicates the application of the attack. The complexity of finding these 2^8 pairs is 2^{64} using the rebound attack, as described in Sect. 3.4.

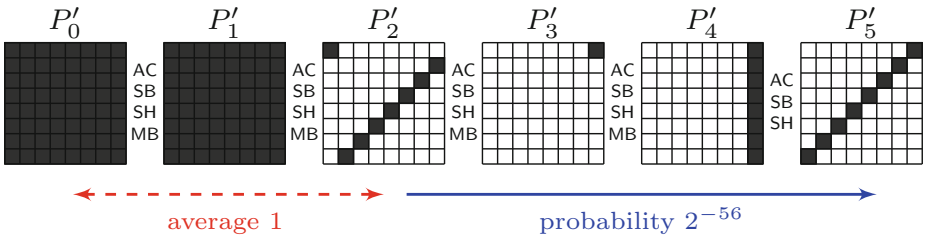


Fig. 4. Truncated differential trail for P' used in the attack on 5 rounds.

4.1 Details of the Attack

Since we can construct only 2^8 pairs following the truncated differential trail, but need to cancel a 64-bit difference at the output of the compression function, each step of the attack succeeds only with a probability of 2^{-56}. However, this can be compensated for by using more message blocks in each step of the attack. Then, the attack can be summarized as follows:

1. Use the rebound attack (cf. Sect. 3.4) to find 2^8 pairs following the truncated differential trail. This has a complexity of 2^{64} in time and memory.
2. For each of these 2^8 pairs check if it can be used to cancel the corresponding 8 bytes of differences in state h'_i. This has a probability of 2^{-56}.
3. If no such *right* pair exists, then choose arbitrarily one of the 2^8 pairs and compute the state h'_i. This generates a new starting point with new differences for the next iteration, while keeping the same bytes inactive.
4. After 2^{56} new starting points, we expect to find a *right* pair with the desired output difference.

Since we need 2^{56} new starting points to cancel the differences in 8 bytes of the state, the complexity of the attack is equivalent to $8 \cdot 2^{64+56} = 2^{123}$ compression function evaluations. Note that the length of such a colliding message is about $8 \cdot 2^{56} = 2^{59}$ blocks.

4.2 Reducing the Length of the Colliding Message Pair

Beside the large time and memory complexity of the attack, one might also see the length of the colliding message pair as a limiting factor of the attack. However, the length of the colliding message pair can be significantly reduced by using a tree-based approach. Instead of choosing only one of the 2^8 pairs to generate a new starting point, we can continue with all pairs in parallel. By using a huge tree with 8 levels and 2^8 branches at each level, we get $(2^8)^8 = 2^{64}$ nodes at level 8. One of these 2^{64} nodes will have the desired output difference. This way, the length of the colliding message pair can be reduced to $8 \cdot 8 + 1 = 65$ message blocks.

4.3 Improving the Complexity of the Attack

The complexity of the attack can be slightly improved by using denser characteristics except when canceling the last 8 bytes. Instead of using a truncated differential trail with a $8 \rightarrow 1$ transition in round 3 of the trail, we can use truncated differential trails with $8 \rightarrow 8, 8 \rightarrow 7, \ldots, 8 \rightarrow 2$ which have a probability greater than 2^{-48}. The complexity of the attack is then dominated by the last iteration where we still need an $8 \rightarrow 1$ transition. This will improve the attack complexity by a factor of 8 resulting in a total complexity of 2^{120} compression function evaluations and 2^{64} memory.

5 Collisions in the Chosen-Prefix Setting

In a collision attack on a hash function an attacker has to find two arbitrary messages M and M^* such that $H(M) = H(M^*)$. However, in practice it might be required that the two messages contain some meaningful information, such that it can be used to practically compromise a cryptographic system. Such an example are, for instance, collisions in the chosen-prefix setting, where an attacker searches for a pair (M, M^*) such that

$$H(M_{pre} \| M) = H(M^*_{pre} \| M^*) \tag{3}$$

for a chosen-prefix (M_{pre}, M^*_{pre}). In [24,25], it was shown that such a more powerful attack exists for MD5. Moreover, the application of the attack to construct colliding X.509 certificates and the creation of a rogue certification authority certificate has been shown.

However, in most cases constructing such collisions is more complicated than constructing (random) collisions. In the case of MD5 the collision attack in the chosen-prefix setting has a complexity of 2^{49}, while the currently best collision attack on MD5 has a complexity of 2^{16}. However, in `Grøstl` the collision attack in the chosen-prefix setting has the same complexity as the collision attack. Due to the generic nature of the collision attack, differences in the chaining variables can be canceled efficiently (cf. Sect. 3).

6 Conclusion

In this work, we have provided new and improved cryptanalysis results for the `Grøstl` hash function, which significantly improving on previously known results. To be more precise, by using a new type of differential trail we were able to show collision attacks on `Grøstl` for up to 5 rounds. The extension becomes possible by considering differential trails spanning over more than one message block.

Moreover, due to the generic nature of our attack we can also construct meaningful collisions, i.e. collisions in the chosen-prefix setting with the same complexity. It has been shown in the past that such collisions might be exploited for instance to construct colliding X.509 certificates.

Although our results do not threaten the security of `Grøstl`, we believe that they will lead to a better understanding of the security margin of the hash function.

Acknowledgments. The work has been supported in part by the Austrian Government through the research program COMET (Project SeCoS, Project Number 836628) and through the research program FIT-IT Trust in IT Systems (Project SePAG, Project Number 835919), by the Secure Information Technology Center-Austria (A-SIT), and by the Research Fund KU Leuven, OT/13/071.

References

1. Boura, C., Canteaut, A., De Cannière, C.: Higher-order differential properties of Keccak and *Luffa*. In: Joux, A. (ed.) FSE 2011. LNCS, vol. 6733, pp. 252–269. Springer, Heidelberg (2011)
2. Daemen, J., Rijmen, V.: The wide trail design strategy. In: Honary, B. (ed.) Cryptography and Coding 2001. LNCS, vol. 2260, pp. 222–238. Springer, Heidelberg (2001)
3. Daemen, J., Rijmen, V.: The Design of Rijndael: AES–The Advanced Encryption Standard. Springer, New York (2002)
4. European Network of Excellence in Cryptology: ECRYPT II SHA-3 Zoo. http://ehash.iaik.tugraz.at/wiki/The_SHA-3_Zoo
5. Gauravaram, P., Knudsen, L.R., Matusiewicz, K., Mendel, F., Rechberger, C., Schläffer, M., Thomsen, S.S.: Grøstl - a SHA-3 candidate. Submission to NIST (Round 3). January 2011. http://www.groestl.info
6. Gilbert, H., Peyrin, T.: Super-Sbox cryptanalysis: improved attacks for AES-like permutations. In: Hong, S., Iwata, T. (eds.) FSE 2010. LNCS, vol. 6147, pp. 365–383. Springer, Heidelberg (2010)
7. Ideguchi, K., Tischhauser, E., Preneel, B.: Improved collision attacks on the reduced-round Grøstl hash function. In: Burmester, M., Tsudik, G., Magliveras, S., Ilić, I. (eds.) ISC 2010. LNCS, vol. 6531, pp. 1–16. Springer, Heidelberg (2011)
8. Jean, J., Naya-Plasencia, M., Peyrin, T.: Improved rebound attack on the finalist Grøstl. In: Canteaut, A. (ed.) FSE 2012. LNCS, vol. 7549, pp. 110–126. Springer, Heidelberg (2012)
9. Jean, J., Naya-Plasencia, M., Peyrin, T.: Multiple limited-birthday distinguishers and applications. In: Lange, T., Lauter, K., Lisoněk, P. (eds.) SAC 2013. LNCS, vol. 8282, pp. 533–550. Springer, Heidelberg (2014)

10. Knudsen, L.R.: SMASH—a cryptographic hash function. In: Gilbert, H., Handschuh, H. (eds.) FSE 2005. LNCS, vol. 3557, pp. 228–242. Springer, Heidelberg (2005)
11. Lamberger, M., Mendel, F., Rechberger, C., Rijmen, V., Schläffer, M.: Rebound distinguishers: results on the full whirlpool compression function. In: Matsui, M. (ed.) ASIACRYPT 2009. LNCS, vol. 5912, pp. 126–143. Springer, Heidelberg (2009)
12. Lamberger, M., Mendel, F., Rechberger, C., Rijmen, V., Schläffer, M.: The rebound attack and subspace distinguishers: application to whirlpool. Cryptology ePrint Archive, Report 2010/198 (2010). http://eprint.iacr.org/
13. Lamberger, M., Pramstaller, N., Rechberger, C., Rijmen, V.: Second preimages for SMASH. In: Abe, M. (ed.) CT-RSA 2007. LNCS, vol. 4377, pp. 101–111. Springer, Heidelberg (2006)
14. Mendel, F., Peyrin, T., Rechberger, C., Schläffer, M.: Improved cryptanalysis of the reduced `Grøstl` compression function, `ECHO` permutation and AES block cipher. In: Jacobson Jr, M.J., Rijmen, V., Safavi-Naini, R. (eds.) SAC 2009. LNCS, vol. 5867, pp. 16–35. Springer, Heidelberg (2009)
15. Mendel, F., Rechberger, C., Schläffer, M., Thomsen, S.S.: The rebound attack: cryptanalysis of reduced whirlpool and `Grøstl`. In: Dunkelman, O. (ed.) FSE 2009. LNCS, vol. 5665, pp. 260–276. Springer, Heidelberg (2009)
16. Mendel, F., Rechberger, C., Schläffer, M., Thomsen, S.S.: Rebound attacks on the reduced `Grøstl` hash function. In: Pieprzyk, J. (ed.) CT-RSA 2010. LNCS, vol. 5985, pp. 350–365. Springer, Heidelberg (2010)
17. National Institute of Standards and Technology: FIPS PUB 197: advanced encryption standard. Federal Information Processing Standards Publication 197, U.S. Department of Commerce. November 2001. http://www.itl.nist.gov/fipspubs
18. National Institute of Standards and Technology: Announcing request for candidate algorithm nominations for a new cryptographic hash algorithm (SHA-3) family. Fed. Reg. 27(212):62212–62220 (2007). http://csrc.nist.gov/groups/ST/hash/documents/FR_Notice_Nov07.pdf
19. National Institute of Standards and Technology: SHA-3 selection announcement, October 2012. http://csrc.nist.gov/groups/ST/hash/sha-3/sha-3_selection_announcement.pdf
20. Peyrin, T.: Cryptanalysis of Grindahl. In: Kurosawa, K. (ed.) ASIACRYPT 2007. LNCS, vol. 4833, pp. 551–567. Springer, Heidelberg (2007)
21. Peyrin, T.: Improved differential attacks for ECHO and Grøstl. In: Rabin, T. (ed.) CRYPTO 2010. LNCS, vol. 6223, pp. 370–392. Springer, Heidelberg (2010)
22. Sasaki, Y., Li, Y., Wang, L., Sakiyama, K., Ohta, K.: Non-full-active super-Sbox analysis: applications to ECHO and Grøstl. In: Abe, M. (ed.) ASIACRYPT 2010. LNCS, vol. 6477, pp. 38–55. Springer, Heidelberg (2010)
23. Schläffer, M.: Updated differential analysis of Grøstl (2011). http://www.groestl.info/
24. Stevens, M., Lenstra, A.K., de Weger, B.: Chosen-prefix collisions for MD5 and colliding X.509 certificates for different identities. In: Naor, M. (ed.) EUROCRYPT 2007. LNCS, vol. 4515, pp. 1–22. Springer, Heidelberg (2007)
25. Stevens, M., Sotirov, A., Appelbaum, J., Lenstra, A., Molnar, D., Osvik, D.A., de Weger, B.: Short chosen-prefix collisions for MD5 and the creation of a rogue CA certificate. In: Halevi, S. (ed.) CRYPTO 2009. LNCS, vol. 5677, pp. 55–69. Springer, Heidelberg (2009)
26. Wang, X., Lai, X., Feng, D., Chen, H., Yu, X.: Cryptanalysis of the hash functions MD4 and RIPEMD. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 1–18. Springer, Heidelberg (2005)

27. 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)
28. Wang, X., Yu, H.: How to break MD5 and other hash functions. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 19–35. Springer, Heidelberg (2005)
29. Wu, S., Feng, D., Wu, W., Guo, J., Dong, L., Zou, J.: (Pseudo) Preimage attack on round-reduced Grøstl hash function and others. In: Canteaut, A. (ed.) Fast Software Encryption. LNCS, vol. 7549, pp. 127–145. Springer, Heidelberg (2012)

Cryptanalysis III

Differential Cryptanalysis of Round-Reduced Simon and Speck

Farzaneh Abed, Eik List(✉), Stefan Lucks, and Jakob Wenzel

Bauhaus-Universität Weimar, Weimar, Germany
{farzaneh.abed,eik.list,stefan.lucks,jakob.wenzel}@uni-weimar.de

Abstract. This paper presents differential attacks on Simon and Speck, two families of lightweight block ciphers that were presented by the U.S. National Security Agency in June 2013. We describe attacks on up to slightly more than half the number of rounds. While our analysis is only of academic interest, it demonstrates the drawback of the intensive optimizations in Simon and Speck.

Keywords: Differential cryptanalysis · Block cipher · Lightweight · Simon · Speck

1 Introduction

Due to the continuously growing impact of RFID tags, smartcards, and FPGAs, cryptographic algorithms which are suitable for resource-constrained devices become more and more important. Lightweight ciphers are optimized to operate in such environments which are limited with respect to their memory, battery supply, and computing power. For these applications, hard- and software efficiency are crucial, and designing cryptographic primitives which preserve security under these constraints is a major challenge.

During the last decade, many lightweight ciphers have been developed including but not limited to HIGHT [11], KATAN [8], KLEIN [9], L-Block [16], LED [10], mCrypton [12], PRESENT [6], and PRINCE [7]. In June 2012, Beaulieu et al. from the U.S. National Security Agency (NSA) contributed to this ongoing research process with the announcement of two novel families of lightweight cipher families, called Simon and Speck [3]. Both constructions support an uncommonly large range of block sizes from 32 to 128 and key sizes from 64 to 256 bits in order to suit a variety of implementations. Simon was thereby optimized for hardware (like KATAN, LED, or PRESENT), and Speck for software implementations (such as KLEIN); though, due to immense optimizations in their round functions, both cipher families perform well in hard- *and* software.

Related Work. Due to their simple structure, Simon and Speck were already target of various cryptanalytical efforts. Alkhzaimi and Lauridsen [2] presented – parallel to our work – differential attacks on up to 16, 18, 24, 29, and 40 rounds for Simon with 32-, 48-, 64-, 96-, and 128-bit state size, respectively. In addition, the authors showed impossible-differential attacks on up to 14, 15, 16, 19, and 22

C. Cid and C. Rechberger (Eds.): FSE 2014, LNCS 8540, pp. 525–545, 2015.
DOI: 10.1007/978-3-662-46706-0_27

rounds and discussed observations regarding rotational cryptanalysis and weak keys. Alizadeh et al. [1] recently presented the best linear attacks on SIMON, with attacks on 12, 15, 19, 28, and 35 rounds.

Biryukov and Velichkov [5] followed another promising approach, where they showed differential characteristics and trails on up to 14, 15, and 21 rounds of SIMON and 9, 10, and 13 rounds of SPECK with 32-, 48-, and 64-bit state size, respectively. The authors adapted Matsui's algorithm (which can find optimal differential characteristics for S-box-based ciphers) for ARX constructions by a concept they called *highways and country roads*. They pointed out that the computation of a complete differential distribution table (DDT) is infeasible for ARX-based primitives. To overcome this challenge, the authors constructed two partial DDTs: a first one with the characteristics of highest probability (highways), and a second one with trails of slightly lower probabilities (country roads) in order to connect and/or improve their previous characteristics.

Contribution and Outline. This paper describes our differential attacks on SIMON and SPECK, which are summarized in Table 1. In what follows, Sect. 2 first reviews the necessary details of the encryption functions of SIMON and SPECK. Section 3 recaps properties of the differential propagation through their respective round functions. Section 4 follows up with a description of how we constructed differential characteristics through parts of both ciphers, and how to extend these characteristics over further rounds. We later use these characteristics for basic differential key-recovery attacks, which we explain first for SIMON in Sect. 5. Then, Sect. 6 describes our differential attacks on SPECK. Section 7 shows rectangle attacks on SPECK. We conclude this work in Sect. 8.

Notions. We follow the notions of [3], where n denotes the word size in bits, $2n$ the state size in bits, and the tuple (L^r, R^r) (the left and right parts of) a state after the encryption of Round r. Further, k represents the length of the secret key. Furthermore, $\oplus$ denotes the bit-wise XOR, $+$ the addition modulo 2^n, $\wedge$ bit-wise AND, $\vee$ bit-wise OR, and $\overline{x}$ the bit-wise inverse of x. We denote by x_i the i-th least significant bit of value x, and enumerate the bits by $x = x_{n-1}x_{n-2}\ldots x_1x_0$. Alternatively, we write values in typewriter font, i.e., $\mathtt{x}$ for hex, and $\mathtt{x}_2$ for binary values, e.g., $\mathtt{1F} = 31$ and $\mathtt{110}_2 = 6$. Concerning differences, we denote by Δ_i a difference with all bits are zero, except for the i-th (least significant) bit, and by $\Delta_{i,[j,k,\ldots]}$ a difference where the i-th bit is active and the values of the bits in square brackets are unknown. Further, we denote a differential characteristic or trail from an input difference α to an output difference β by $\alpha \rightarrow \beta$.

2 Brief Description of SIMON and SPECK

SIMON and SPECK are two simple Feistel constructions that apply a combination of rotation, XOR, and either addition (SPECK) or the logical AND (SIMON) iteratively over many rounds. The encryption process of SIMON is given in Algorithm 1, that for SPECK in Algorithm 2. Both cipher families are defined for state sizes $2n$ and key sizes k: 32/64, 48/72, 48/96, 64/96, 64/128, 96/96, 96/144, 128/128, 128/192, and 128/256.

Table 1. Summary of our results on Simon and Speck. (*) = the time complexities assume that we have two independent filtering steps (cf. Remark 1). CP = chosen plaintexts, † = attack uses chosen ciphertexts.

Cipher	Attacked Rounds	Time (*)	Data (CP)	Memory (Bytes)	Success Rate
Differential					
Simon32/64	18/32 (0.56)	$2^{46.0}$	$2^{31.2}$	$2^{15.0}$	0.63
Simon48/72 †	19/36 (0.52)	$2^{52.0}$	$2^{46.0}$	$2^{20.0}$	0.98
Simon48/96 †	19/36 (0.52)	$2^{76.0}$	$2^{46.0}$	$2^{20.0}$	0.98
Simon64/96	26/42 (0.61)	$2^{63.9}$	$2^{63.0}$	$2^{31.0}$	0.86
Simon64/128	26/44 (0.59)	$2^{94.0}$	$2^{63.0}$	$2^{31.0}$	0.86
Simon96/96	35/52 (0.67)	$2^{93.3}$	$2^{93.2}$	$2^{37.8}$	0.63
Simon96/144	35/54 (0.64)	$2^{101.1}$	$2^{93.2}$	$2^{37.8}$	0.63
Simon128/128	46/68 (0.67)	$2^{125.7}$	$2^{125.6}$	$2^{40.6}$	0.63
Simon128/192	46/69 (0.66)	$2^{142.0}$	$2^{125.6}$	$2^{40.6}$	0.63
Simon128/256	46/72 (0.63)	$2^{206.0}$	$2^{125.6}$	$2^{40.6}$	0.63
Differential					
Speck32/64	10/22 (0.45)	$2^{29.2}$	2^{29}	2^{16}	0.99
Speck48/72	12/22 (0.54)	$2^{45.3}$	2^{45}	2^{24}	0.99
Speck48/96	12/23 (0.52)	$2^{45.3}$	2^{45}	2^{24}	0.99
Speck64/96	15/26 (0.57)	$2^{61.1}$	2^{61}	2^{32}	0.99
Speck64/128	15/27 (0.55)	$2^{61.1}$	2^{61}	2^{32}	0.99
Speck96/96	15/28 (0.51)	$2^{89.1}$	2^{89}	2^{48}	0.99
Speck96/144	15/29 (0.51)	$2^{89.1}$	2^{89}	2^{48}	0.99
Speck128/128	16/32 (0.50)	$2^{111.1}$	2^{116}	2^{64}	0.99
Speck128/192	16/33 (0.48)	$2^{111.1}$	2^{116}	2^{64}	0.99
Speck128/256	16/34 (0.47)	$2^{111.1}$	2^{116}	2^{64}	0.99
Rectangle					
Speck32/64	11/22 (0.50)	$2^{46.7}$	$2^{30.1}$	$2^{37.1}$	≈ 1
Speck48/72	12/22 (0.54)	$2^{58.8}$	$2^{43.2}$	$2^{45.8}$	≈ 1
Speck48/96	12/23 (0.52)	$2^{58.8}$	$2^{43.2}$	$2^{45.8}$	≈ 1
Speck64/96	14/26 (0.53)	$2^{89.4}$	$2^{63.6}$	$2^{65.6}$	≈ 1
Speck64/128	14/27 (0.51)	$2^{89.4}$	$2^{63.6}$	$2^{65.6}$	≈ 1
Speck96/144	16/29 (0.55)	$2^{135.9}$	$2^{90.9}$	$2^{94.5}$	≈ 1
Speck128/192	18/33 (0.54)	$2^{182.7}$	$2^{125.9}$	$2^{121.9}$	≈ 1
Speck128/256	18/34 (0.52)	$2^{182.7}$	$2^{125.9}$	$2^{121.9}$	≈ 1

For Simon, $f : \{0,1\}^n \to \{0,1\}^n$ is defined as $f(x) := (x \lll 1) \wedge (x \lll 8)$. The rotation constants in Speck are $\alpha = 8$ and $\beta = 3$ for the most versions of Speck; only Speck32/64 uses $\alpha = 7$ and $\beta = 2$.

Algorithm 1. Encryption with SIMON.

Input: $(L^0, R^0) \in \{0,1\}^{2n}$
Output: $(L^r, R^r) \in \{0,1\}^{2n}$
1: **for** $i = 1, \ldots, r$ **do**
2: $\quad L^i \leftarrow R^{i-1} \oplus f(L^{i-1})$
3: $\quad L^i \leftarrow L^i \oplus K^{i-1} \oplus (L^{i-1} \lll 2)$
4: $\quad R^i \leftarrow L^{i-1}$
5: **end for**
6: **return** (L^r, R^r)

Algorithm 2. Encryption with SPECK.

Input: $(L^0, R^0) \in \{0,1\}^{2n}$
Output: $(L^r, R^r) \in \{0,1\}^{2n}$
1: **for** $i = 1, \ldots, r$ **do**
2: $\quad L^i \leftarrow (L^{i-1} \ggg \alpha) + R^{i-1} \bmod 2^n$
3: $\quad L^i \leftarrow L^i \oplus K^{i-1}$
4: $\quad R^i \leftarrow (R^{i-1} \lll \beta) \oplus L^i$
5: **end for**
6: **return** (L^r, R^r)

3 Differential Properties of SIMON and SPECK

Differential Properties for the Round Function of SPECK. For SPECK, one requires only the well-known XOR-differential probabilty of the modular addition (xdp^+), which was studied in detail by Lipmaa and Moriai [13,14].

Definition 1 (XOR-Differential Probabilty of Addition [14]). *Let α, β, γ be fixed n-bit XOR differences, and $f(x, y) = x + y \bmod n$. Then, xdp^+ is defined as the probability over all $x \in \{0,1\}^n$, such that*

$$xdp^+(\alpha, \beta \rightarrow \gamma) = 2^{-2n} \left|\{(x, y) : f(x, y) \oplus f(x \oplus \alpha, y \oplus \beta) = \gamma\}\right|.$$

Differential Properties for the Round Function of SIMON. For SIMON, one has to consider the differential probability for the round function $f(x)$. At the end of this section, we provide an algorithm that yields the set and number of all possible output differences for a fixed input difference. In the following, we explain first the differential probability (DP) of logical AND; next, we derive the DP of AND in combination with rotation, and then consider the DP of AND with rotationally dependent inputs. We follow the notation by [5].

Property 1 (Absorption of Logical AND)**.** *Let $x, x', y, y' \in \{0,1\}$ and $f(x, y) = x \wedge y$. Let $\alpha = x \oplus x', \beta = y \oplus y', \gamma = f(x, y) \oplus f(x', y')$. Then, it applies that*

$$\Pr[\alpha, \beta \rightarrow \gamma = 0] = \begin{cases} 1 & \text{if } \alpha = \beta = 0, \\ 1/2 & \text{otherwise.} \end{cases}$$

Property 1 states that the differential output of the logical AND is biased: if α and β are 0, then γ must be 0. If α and/or β is 1, there is still a probability of $1/2$ that the AND operation will cancel the active bit in the output difference.

Definition 2 (XOR-Differential Probability of AND). *Let α, β, γ be fixed n-bit XOR differences, and let $f(x, y) = x \wedge y$. The XOR-differential probability of the logical AND ($xdp^\wedge$) is the probability over all $x, y \in \{0,1\}^n$, such that*

$$xdp^\wedge(\alpha, \beta \rightarrow \gamma) = 2^{-2n} \left|\{(x, y) : f(x, y) \oplus f(x \oplus \alpha, y \oplus \beta) = \gamma\}\right|.$$

Property 2 (XOR-Differential Probability of AND). *Let α, β, γ be fixed n-bit XOR differences and $hw(\cdot)$ the hamming-weight function. Then,*

$$xdp^{\wedge}(\alpha, \beta \rightarrow \gamma) = \begin{cases} 0 & \text{if } \gamma \wedge \overline{\alpha \vee \beta} \neq 0^n, \\ 2^{-hw(\alpha \vee \beta)} & \text{otherwise.} \end{cases}$$

Property 2 transfers Property 1 from bits to n-bit differences. Only those bits that are active in α and/or β can be active in γ – each with probability 1/2. This is reflected by the term $(\alpha \vee \beta)$. If γ contains active bits at other positions, then, $\gamma \wedge \overline{\alpha \vee \beta} \neq 0^n$ and $\Pr[\alpha, \beta \rightarrow \gamma] = 0$. Otherwise, all other possible differences γ are equally possible. Thus, the term $\alpha \vee \beta$ can be interpreted as the definition of a *set* of possible output differences, i.e., one can efficiently iterate over all possible combinations of values for its active bits and will obtain all possible output differences γ.

Definition 3 (XOR-Differential Probabilty of AND with Rotations). *Let α, β, γ be fixed n-bit XOR differences, $r \in [0, n-1]$ be a fixed rotation amount, and $f(x, y) = x \wedge (y \lll r)$. Then, $xdp^{\wedge, \lll}$ is defined as the probability over all $x, y \in \{0,1\}^n$, such that*

$$xdp^{\wedge, \lll}(\alpha, \beta \rightarrow \gamma) = 2^{-2n} \left| \{(x, y) : f(x, y) \oplus f(x \oplus \alpha, y \oplus \beta) = \gamma\} \right|.$$

Since rotation and bit-wise logical AND are linear, we can derive

$$\mathrm{xdp}^{\wedge, \lll}(\alpha, \beta \rightarrow \gamma) = \begin{cases} 0 & \text{if } \gamma \wedge \overline{\alpha \vee (\beta \lll r)} \neq 0^n, \\ 2^{-hw(\alpha \vee (\beta \lll r))} & \text{otherwise.} \end{cases}$$

We can easily transform $f(x) = (x \lll s) \wedge (x \lll t)$, with $s, t \in [0, n-1], s \neq t$ into $f(x) = x \wedge (x \lll r)$ with $r = s - t \bmod n$. In the following, we also take rotationally dependent inputs into account.

Definition 4 (XOR-Differential Probabilty of AND with Rotationally Dependent Inputs). *Let α, β be fixed n-bit XOR differences, $r \in [0, n-1]$ be a fixed integer, and $f(x) = x \wedge (x \lll r)$. Then, $xdp^{x \wedge (x \lll r)}$ is defined as the probability over all $x \in \{0,1\}^n$, such that*

$$xdp^{x \wedge (x \lll r)}(\alpha \rightarrow \beta) = 2^{-n} \left| \{x : f(x) \oplus f(x \oplus \alpha) = \beta\} \right|.$$

Property 3 (Differential Propagation of $xdp^{x \wedge (x \lll r)}$). *Let α be fixed n-bit XOR difference and $r \in [0, n-1]$ be a fixed integer. Let $f : \{0,1\}^n \rightarrow \{0,1\}^n$ be defined by $f(x) = x \wedge (x \lll r)$. Then, the set of possible output differences β for $xdp^{x \wedge (x \lll r)}$, can be efficiently computed in $O(n)$ as shown in Algorithm 3.*

Example: $n = 16, r = 7, \alpha = \texttt{0500}$.Let x, x' be two 16-bit values which serve as input to $f(x)$, with $x \oplus x' = \alpha$ and $\beta = f(x) \oplus f(x')$. We see that

$$\begin{aligned} \alpha = \alpha_{15} \ldots \alpha_0 &= \texttt{00000101 0000000}\ 0_2 \text{ (top)} \\ \alpha \lll r = \alpha_8 \ldots \alpha_0 \alpha_{15} \ldots \alpha_9 &= \texttt{10000000 0000001}\ 0_2 \text{ (bottom)} \\ \beta = \beta_{15} \ldots \beta_0 &= \texttt{1000010}*^1\texttt{000000}*^1 0_2 \end{aligned}$$

Algorithm 3 returns $\beta = \texttt{1000010}*^1\texttt{000000}*^1 0_2$, and $count = 3$. The star symbol $*$ denotes dependent bits and the index $*^i$, indicates pairs of bits that are related.

Algorithm 3. Given a n-bit input difference α, computes the set of possible output differences β for $f(x) = x \wedge (x \lll r)$.

Input: $\alpha \in \{0,1\}^n$ {Input difference}
Output: $\beta \in \{0,1\}^n$ {Set of all possible output differences},
  count {# of possible output differences}
  $\beta \leftarrow 0^n, count \leftarrow 0$
  for $i \leftarrow 0, \ldots, n-1$ **do**
    if $(\alpha_i \vee \alpha_{(i+r \bmod n)}) \wedge \beta_i = 0$ **then** {Bit β_i can be active}
      $\beta_i \leftarrow 1$
      $count \leftarrow count + 1$
    end if
    if $\overline{\alpha_i} \wedge \alpha_{(i-r \bmod n)} \wedge \alpha_{i+r \bmod n}$ **then** $\{\beta_i = \beta_{(i+r \bmod n)}\}$
      $\beta_i \leftarrow *^i$
      $\beta_{i+r \bmod n} \leftarrow *^i$
    end if
  end for
  return $(\beta, count)$

– β_1 depends on α_1 (top) and α_{10} (bottom), with $\alpha_1 = 0$ and $\alpha_{10} = 1$;
– β_8 depends on α_8 (top) and α_1 (bottom), with $\alpha_1 = 0$ and $\alpha_8 = 1$;

From $\alpha_1 = 0$ follows that $x_2 = x'_2$. When $x_2 = x'_2 = 0$ then $\beta_1 = \beta_8 = 0$; otherwise, when $x_2 = x'_2 = 1$, it must hold that $\beta_1 = \beta_8 = 1$. We call β_1 and β_8 *dependent bits*. Since β contains four active bits and one pair of them depends on each other, there are $2^{4-1} = 2^3$ possible output differences defined by β, namely:

$$\{0000, 0102, 0400, 0502, 8000, 8102, 8400, 8502.\},$$

Each difference can be formed by $2^{16-3} = 2^{13}$ possible pairs (x, x').

4 Search for Differential Characteristics and Differentials

During our analysis, we applied a two-step approach to find our differentials. Firstly, we employed Matsui's algorithm [15] to find some characteristics for the 32-, 48-, and 64-bit versions of Simon:

	Simon32/64	Simon48/k	Simon64/k
α	$(\Delta_5, 0)$	$(\Delta_{8,16}, \Delta_{6,14,18})$	$(\Delta_6, 0)$
β	$(\Delta_{14}, 0)$	$(\Delta_{6,14,18,22}, \Delta_{20})$	$(\Delta_{6,10,14}, \Delta_{12})$
Rounds	12	15	20
$\Pr[\alpha \rightarrow \beta]$	2^{-36}	2^{-52}	2^{-70}

Secondly, we applied a branch-and-bound search, similar to the approach of [2]. There, we started from the input difference α and propagated it round-wise in forward and backward direction. For each round, we collected all possible

output characteristics $\alpha \rightarrow \beta$ and their probability p as a tuple (β, p) in a set and used them as a starting point for the next round in a depth-first manner. Therefore, we used Algorithm 3 for Simon and a variant of the Algorithm by Lipmaa and Moriai [14] for Speck.

Since following each path is infeasible, we pruned the search tree by considering only characteristics $\alpha \rightarrow \beta$ with a probability above a chosen threshold. Therefore, we used the characteristic found with Matsui's algorithm as a reference, i.e., say Matsui's characteristic had probability $p = 2^{-q}$ after some round r, we only considered those characteristics β as input to round $r+1$ that had a probability $p \gg 2^{-q-thresh}$. We further pruned the search tree by only storing a (chosen) maximal number of characteristics.

Every time two differential characteristics lead to the same output difference β after a round, we merged them to one differential trail and added their probabilities. We emphasize that our characteristics have been found experimentally and do not necessarily represent the best possible ones. Further, note that we rely on the assumption that all possible round keys are equally probable and uniformly distributed for every round.

Extending Differential Characteristics to Attacks. A given differential can be extended by a few more rounds in a key-recovery attack for any version of Simon$2n/k$. Assume, we are given an r-round differential $(\alpha, \beta) \rightarrow (\gamma, \delta)$. Because Simon injects the subkey at the end of the rounds, the adversary itself can compute the output of $f(x)$ in the first round, choose $(\beta, \alpha \oplus f(\beta))$ as input difference and obtains an $(r+1)$-round differential with equal probability. A similar strategy can be applied at the output side. Given an output difference (γ, δ) after $(r+1)$ rounds, the difference after $(r+2)$ rounds is $(\delta \oplus f(\gamma), \gamma)$. Since the subkey in the last round of a characteristic does not affect the output difference $\delta \oplus f(\gamma)$, the adversary can compute $f(\gamma)$ itself and obtains an $(r+2)$-round differential with equal probability.

For the versions 48/72-, 64/96-, 96/144-, and 128/192-bit versions, one can append a further round by simply guessing its full subkey. The total computational effort for collecting plaintext-ciphertext pairs and testing all subkey candidates for the appended round remains significantly smaller than that for exhaustively searching the full key space. Moreover, for the 32/64-, 48/96-, 64/128-, and 128/256-bit versions, one can append another round by guessing its subkey.

5 Key-Recovery Attacks on Simon

In this section we describe a key-recovery attack on round-reduced Simon32/64. Since attacks on the further variants follow a similar procedure, we specify only their complexities at the end of this section. For Simon32/64 we use the 13-round differential characteristic with $p \approx 2^{-30.2}$ (see Table 4 in Appendix A) over the rounds $2-14$:

$$\Delta^1 = (0, \Delta_6) \rightarrow (\Delta_{14}, 0) = \Delta^{14}.$$

Note that we can choose the left part of the plaintext pairs P, P', s.t. we obtain the desired difference Δ^1 after the first round. We can append four additional rounds to the end of the cipher, where we will guess in total 18 key bits. From the obtained ciphertexts, we still know many bits from the truncated trail:

$$
\begin{aligned}
(\Delta L^{15}, \Delta R^{15}) &= (\Delta_{0,[6,15]}, \Delta_{14}),\\
(\Delta L^{16}, \Delta R^{16}) &= (\Delta_{2,[0,1,7,8,14]}, \Delta_{0,[6,15]}),\\
(\Delta L^{17}, \Delta R^{17}) &= (\Delta_{4,[0,1,2,3,6,8,9,10,15]}, \Delta_{2,[0,1,7,8,14]}),\\
(\Delta L^{18}, \Delta R^{18}) &= (\Delta_{6,[14]}, \Delta_{4,[0,1,2,3,6,8,9,10,15]}).
\end{aligned}
$$

Attack Procedure. The full attacking procedure can be split into a *collection*, a *pair-filtering*, a *key-guessing*, and a *brute-force phase*:

Collection Phase

1. Initialize an empty set $\mathcal{C} = \emptyset$.
2. Choose $2^{30.2}$ plaintext pairs (P_i, P_i'), s.t. their difference after the first round yields Δ^1.
3. Collect their corresponding ciphertext pairs (C_i, C_i') from an encryption oracle, where $C_i = E_K(P_i)$ and $C_i' = E_K(P_i')$.

Pair-Filtering Phase

4. For all ciphertext pairs, invert the final round to derive Δ^{17} and store all pairs (C_i, C_i') with the correct difference at the known bits Δ^{17} in $\mathcal{C}$. We know seven bits of ΔL^{17} and 11 bits of ΔR^{17}. Assuming the differences Δ^{17} are uniformly distributed, we can expect $2^{30.2-18} = 2^{12.2}$ pairs in average.

Key-Guessing Phase

5. Create a list of counters for all 2^{18} possible values of the round-key bits $K^{17}_{0,1,5,7-11,14,15}$, $K^{16}_{6-9,13,15}$, and $K^{15}_{9,7}$, and perform the following steps for each candidate:
   - For all pairs $(C_i, C_i') \in \mathcal{C}$:
     - Partially decrypt (C_i, C_i') to the state after the encryption of Round 14. If their difference matches Δ^{14}, increment the counter for the current key candidate.
6. Output the key candidate(s) which is/are associated to the highest counter values.

Brute-Force Phase

7. For all bits of K^{17}, K^{16}, K^{15}, and K^{14} that are not guessed in the previous steps, perform further encryptions to identify their correct values.

Attack Complexity. The attack requires $2^{31.2}$ chosen plaintexts. Regarding the memory complexity, we store $2 \cdot 2^{12.2}$ texts of 32 bits each, or $2^{15.2}$ bytes for the attack. The computational effort for the collection phase, C_{collect}, is equivalent to $2^{30.2}$ full encryptions performed by the oracle. The filtering effort, C_{filter}, is given by $2^{30.2}$ one-round decryptions to check 18 bits of Δ^{17}. The effort for the key-guessing phase, $C_{\text{key-guessing}}$, consists of decrypting the remaining pairs for each of the 2^{18} key candidates over three further rounds.

Assuming that both filtering steps of the pair-filtering and the key-guessing phase are independent from each other, we can identify the correct value of the 18 guessed key bits. A trivial brute-force search can find the correct value of the 46 remaining bits of the considered subkeys $K^{14}, K^{15}, K^{16}, K^{17}$ with about 2^{46} encryptions. The total computational complexity can be estimated by

$$\underbrace{2 \cdot 2^{30.2}}_{C_{\text{collect}}} + \underbrace{2 \cdot 2^{30.2} \cdot \frac{1}{18}}_{C_{\text{filter}}} + \underbrace{2 \cdot 2^{18} \cdot 2^{18} \cdot \frac{3}{18}}_{C_{\text{key-guessing}}} + \underbrace{2^{46}}_{C_{\text{bruteforce}}} \approx 2^{46} \text{ encryptions.}$$

Remark 1. Note that in the case that our assumption would not hold, we still have a differential that is satisfied with probability $p = 2^{-30.2}$, and a 32-bit filter at Δ^{14}. Hence, we can expect to be able to reduce the candidates of the 18 key bits we guess in the final four rounds to $2^{30.2} \cdot 2^{18} \cdot 2^{-32} = 2^{16.2}$, increasing the complexity of the brute-force step to $2^{46} \cdot 2^{16.2} = 2^{62.2}$ encryptions, which is still significantly faster than exhaustive search. In general case, the computational effort for our attacks would then be dominated by the costs for a simple exhaustive search on the remaining key space. Hence, the time complexities would then become approximately $2^k/(p \cdot 2^{2n})$ ($k = 64$, $n = 16$, $p \approx 2^{-30.2}$ for Simon32/64).

Success Rate. Since the probability of a pair to follow our differential is about $2^{-30.2}$, the probability that at least one correct pair occurs for the correct key can be approximated by

$$1 - \Pr[n = 2^{30.2}, p = 2^{-30.2}, x \leq 0] = 1 - 1/e \approx 0.632.$$

Similar Attacks on Further Versions. We can apply the same procedure to further versions of Simon. To cover one additional round, we use chosen ciphertexts in the attack on Simon48/k. Table 2 summarizes the probabilities, required number of pairs, known state bits to filter (1st filter), guessed key bits (key bits), and success rates (where false random shows the probability that no correct pair occurs during execution of the respective attack, and false real denotes the probability of a false-positive pair to occur) for each attack. The differential characteristics for the further version are illustrated in Tables 4, 5 and 6 in Appendix A.

Table 2. Parameters of our differential attacks on Simon. "1st Filter" denotes the number of bits that can be used to filter out pairs after inverting the final round; key bits = # guessed key bits; p = Probability of the used differential.

Cipher	Rounds	Pairs	1st Filter	Key Bits	Stored pairs	p	Succ. rate
Simon32/k	18	$2^{30.2}$	18	18	$2^{12.2}$	$2^{-30.2}$	0.632
Simon48/k	19	$2^{45.0}$	28	20	$2^{17.0}$	$2^{-43.0}$	0.981
Simon64/k	25	$2^{62.0}$	35	36	$2^{27.0}$	$2^{-61.0}$	0.863
Simon96/k	35	$2^{92.2}$	59	43	$2^{33.2}$	$2^{-92.2}$	0.632
Simon128/k	46	$2^{124.6}$	89	50	$2^{35.6}$	$2^{-124.6}$	0.632

6 Differential Attacks on SPECK

In this section we describe our differential analysis of SPECK. Since the small version of SPECK (SPECK32/64) allows a simple practical verification, in the following, we only discuss this version in detail. We apply the same strategy to the remaining family members of SPECK and present only their complexities at the end of this section.

6.1 Key-Recovery Attack on SPECK34/64

We use the characteristic for SPECK32/64 from Table 7 in Appendix A with $p \approx 2^{-24}$ over rounds $2-9$ to mount a 10-round attack.

$$\Delta^1 = (\Delta_{5,6,9,11}, \Delta_{0,2,9,14}) \rightarrow (\Delta_{1,3,5,15}, \Delta_{3,5,7,10,12,14,15}) = \Delta^9.$$

Attack Procedure. Again, we split the attacking procedure into a *collection*, a *key-guessing*, and a *brute-force phase*:

Collection Phase

1. Initialize an empty list $\mathcal{C} = \emptyset$.
2. Choose 2^{28} pairs (P_i, P_i') s.t. their difference after the first round is Δ^1.
3. Collect the corresponding ciphertext pairs (C_i, C_i') from a decryption oracle, where $C_i = E_K(P_i)$ and $C_i' = E_K(P_i')$. Derive $\Delta L^9_{0-3}, \Delta R^9$ and store all pairs (C_i, C_i') with $\Delta L^9_{0-3} = \Delta_3$ and $\Delta R^9 = \Delta_{3,5,7,10,12,14,15}$ in the list $\mathcal{C}$.

Key-Guessing Phase

4. Create a list of 2^{12} counters.
5. For all possible values of the 12 key bits K^9_{4-15}:
   - For all pairs $(C_i, C_i') \in \mathcal{C}$:
     - Partially decrypt (C_i, C_i') to the state after the encryption of Round 9, and derive ΔL^9. If $\Delta L^9 = \Delta_{1,3,5,15}$, then increment the counter for the current key candidate.
6. Output those keys as potentially correct for which their counter has a value of at least four.
7. Mark all pairs which yielded the correct Δ^9 for the potentially correct key(s) as correct pairs.

Brute-Force Phase

8. Partially decrypt all correct pairs round by round to get the correct subkey bits K^9_{0-3}, K^8, K^7, and K^6.

Success Rate. The probability that a pair follows our differential characteristic is about 2^{-24}. Hence, the probability that no more than three correct pairs occur when using SPECK (i.e., the correct subkey will not be found) is about

$$\Pr[n = 2^{28}, p = 2^{-24}, x \leq 3] \approx 9.31 \cdot 10^{-5},$$

and hence, the success probability of the attack approx. $1 - 9.31 \cdot 10^{-5} > 0.99$.

Table 3. Parameters of our differential attacks on Speck. "1st Filter" denotes the number of bits that can be used to filter out pairs after inverting the final round; key bits = # guessed key bits; p = Probability of the used differential.

Cipher	Rounds	Pairs	1st Filter	Key Bits	Stored pairs	p	Succ. rate
Speck32/k	10	2^{28}	20	12	2^{8}	$2^{-24.0}$	0.99
Speck48/k	12	2^{44}	25	23	2^{19}	$2^{-40.6}$	0.99
Speck64/k	15	2^{61}	35	29	2^{26}	$2^{-58.9}$	0.99
Speck96/k	15	2^{88}	54	42	2^{34}	$2^{-84.0}$	0.99
Speck128/k	16	2^{115}	67	61	2^{48}	$2^{-111.1}$	0.99

Attack Complexity. Our attack on Speck32/64 requires 2^{29} chosen plaintexts. The computational effort for C_{collect} covers 2^{29} full encryptions performed by an encryption oracle. The filtering effort, C_{filter}, is twofold. First, we partially decrypt all ciphertext pairs over the final round. There, we have a 20-bit filter from the four least-significant bits of ΔL^9 and the full ΔR^9. Assuming all differences occur uniformly at random, we expect to have $2^{28-20} = 2^8$ remaining pairs afterwards. Thereupon, for 2^{12} values of K^9_{4-15}, we derive the remaining 2^8 pairs and derive ΔL^9. In the brute-force phase, the adversary partially decrypts the remaining pairs round by round to identify the correct round keys. Therefore, the full computational complexity is given by

$$\underbrace{2^{29}}_{C_{\text{collect}}} + \underbrace{2^{29} \cdot \frac{1}{10} + 2^8 \cdot 2^{12} \cdot \frac{1}{10}}_{C_{\text{filter}}} + \underbrace{\left(2^4 + 2^{16} + 2^{16} + 2^{16}\right) \cdot 2^8 \cdot \frac{1}{10}}_{C_{\text{bruteforce}}} \approx 2^{29.16}$$

encryptions. Concerning the memory complexity, we store a list of counters for all key candidates, which requires 2^{12} bytes for the first filtering phase and 2^{16} bytes for the counters of the round keys in the brute-force phase.

We can apply a similar procedure for the remaining versions of Speck and obtain the results of Table 3. In all cases, the computational effort is dominated by the brute-force step. The differentials for the individual versions of Speck can be found in the Tables 7, 8, 9 and 10 in Appendix A.

7 Rectangle Attacks on Speck

Boomerangs and Rectangles. Boomerangs and rectangles allow to use two short differential characteristics with high probabilities instead of a single long differential. Therefore, one first splits a given cipher E into parts $E = E^2 \circ E^1$, and searches for two differentials $\alpha \xrightarrow[E^1]{p} \beta$ and $\gamma \xrightarrow[E^2]{q} \delta$. Next, one collects

quartets of plaintexts (P, P', Q, Q') with $P \oplus P' = Q \oplus Q' = \alpha$. In the following we denote by (R, R', S, S') their encryptions after E^1 and by (C, C', D, D') their encryptions after E^2.

Each quartet has a probability of p^2 that (R, R', S, S') fulfils $R \oplus R' = S \oplus S' = \beta$. We are interested in the case when $R \oplus S = \gamma$ since then, it automatically applies that $R' \oplus S' = \gamma$. With probability q^2, a ciphertext quartet (C, C', D, D') fulfils $C \oplus D = C' \oplus D' = \delta$. In this case, we call it a *right quartet*. If an adversary collects m pairs with difference α, then, the expected number of right quartets according to [4] is:

$$m^2 \cdot 2^{-n} \cdot (pq)^2.$$

Hence, it must apply that $pq < 2^{-n/2}$ in order to mount an attack on E.

As an improvement Biham et al. proposed in [4] to use quartets with any possible difference β' and γ' in the middle, as long as both pairs in a quartet share the same difference β' and γ' after E^1. Thus, the probabilities of p and q increase to

$$\hat{p} = \sqrt{\sum_{\beta'} \Pr[\alpha \to \beta']} \quad \text{and} \quad \hat{q} = \sqrt{\sum_{\gamma'} \Pr[\gamma' \to \delta]}.$$

In the remainder of this section, we describe in details a rectangle attack on 11 rounds of Speck32/64. Since our attacks on the further versions of Speck work similar, we only specify the used trails and their complexities in Tables 11 and 12 in Appendix B.

7.1 Rectangle Attack on Speck34/64

For the attack on Speck32/64 we use the following trails $\alpha \to \beta'$ and $\gamma' \to \delta$:

$$\alpha = (\Delta_{11,13}, \Delta_4) \xrightarrow[E_1]{\hat{p} \geq 2^{-8.01}} \beta' \text{ and } \gamma' \xrightarrow[E_2]{\hat{q} \geq 2^{-4.56}} (\Delta_{15}, \Delta_{1,3,10,15}) = \delta.$$

E_1 represents the rounds 2–6, and E_2 the rounds 7–10. Again, we can split the attacking procedure into a *collection*, a *key-guessing*, and a *brute-force phase*:

Collection Phase

1. Initialize two empty hash tables $\mathcal{C}$, $\mathcal{D}$, and a list $\mathcal{Q}$.
2. Choose $\frac{2^{(n+2)/2}}{\hat{p}\hat{q}} = \frac{2^{34/2}}{2^{-8.01}2^{-4.56}} = 2^{29.57}$ plaintext pairs (P, P') s.t. their difference after the first round is α.
3. Ask for the encryption of (P, P') and receive the corresponding ciphertext pair (C, C'). Then, partially decrypt C, C' over the final round to the state after Round 10, $(R^{10}, {R'}^{10})$ and store the result in $\mathcal{C}$. XOR the right part of δ to $(R^{10} \oplus \Delta_{1,3,10,15}, {R'}^{10} \oplus \Delta_{1,3,10,15}$ and store them in $\mathcal{D}$.

4. Prior, lookup if there is already an entry in $\mathcal{D}$ under the index

$$(R^{10} \oplus \Delta_{1,3,10,15}, R'^{10} \oplus \Delta_{1,3,10,15}).$$

If there is, label the existing ciphertext pair in $\mathcal{D}$ as (D, D') and store the quartet (C, C', D, D') in $\mathcal{Q}$. We can build $(2^{29.57})^2/2 = 2^{58.14}$ quartets from our pairs. Since this event requires a match in 16 bits of the first, and 16 bits of the second pair, we can expect to have at average a number of $2^{58.14-32} \approx 2^{26.14}$ false positive quartets for which this condition holds. Since the probability of a right quartet is $2^{2\cdot-8.01+2\cdot-4.56} = 2^{-25.14}$, we can expect $2^{58.14-25.14} = 2^{33}$ right quartets in addition. We approximate $2^{33} + 2^{26.14} \approx 2^{33}$ hereafter.

Filtering Phase

5. Initialize a table $\mathcal{T}$ of 2^{16} counters.
6. For all possible values of the subkeys K^{10}:
   6.1 Decrypt all quartets over the final round and check whether their difference ΔL^{10} is equal to Δ_{15}. If yes, then increment the counter for the current key candidate in $\mathcal{T}$.
7. Output the key candidate(s) with the maximal count(s) in $\mathcal{T}$.

Brute-Force Phase

8. Partially decrypt the remaining pairs round by round to identify the further round keys K^9, K^8, and K^7.

Attack Complexity. The attack requires $2^{30.07}$ chosen plaintexts. We have to store the corresponding ciphertexts, the remaining 2^{33} quartets, and a list of 2^{16} counters for all round-key candidates. So, we can approximate the required memory by $(2^{30.07}+4\cdot 2^{33})\cdot 32/8+2^{16} \approx 2^{37.1}$ bytes. The computational effort for the collection phase, C_{collect}, consists of $2^{30.07}$ full encryptions performed by the oracle, and $2^{30.07}$ half-round decryptions. Additionally, we need $2^{30.07}$ memory accesses to look up potential quartets and about $4 \cdot 2^{33}$ memory accesses in average to store the remaining quartets.

To use consistent units, we overestimate a memory access by a half-round computation. In the filtering phase, we have to perform $2^{16} \cdot 4 \cdot 2^{33} = 2^{51}$ half-round decryptions to obtain the difference in the left word after Round 10. Summing up, we obtain a computational effort of

$$\underbrace{2^{30.07} + (2^{30.07} + 2^{30.07} + 4 \cdot 2^{26.14}) \cdot \frac{1}{22}}_{C_{\text{collect}}} + \underbrace{2^{51.14} \cdot \frac{1}{22}}_{C_{\text{filter}}} + \underbrace{2^{16} + 2^{16} + 2^{16}}_{C_{\text{bruteforce}}} \approx 2^{46.68}$$

encryptions.

8 Discussion and Conclusion

This work presented differential attacks on round-reduced versions of the SIMON and SPECK. Furthermore, we briefly considered rectangle attacks on SPECK. We also studied rectangle attacks on SIMON and impossible-differential attacks; however, we omitted those since they did not improve our results with conventional differentials.

Our analysis can be seen as a starting point for further research on SIMON and SPECK. For SIMON, it demonstrates that up to half the number of rounds are vulnerable against differential attacks due to its highly optimizied round function. Moreover, the cipher shows a strong differential effect, i.e., there are many possible characteristics for given input and output difference.

SPECK is much closer to previous ARX designs such as ThreeFish than SIMON. However, while ThreeFish has been published four years ago, still only 1/3 of the rounds have been attacked so far, whereas the current analysis of SPECK already threatened the security of up to half of the rounds little time after publication. Moreover, any new analysis method on addition-based ARX would be a threat to both NSA constructions as well. In conclusion, we can learn from SIMON that ARX designs should incorporate additions to provide reasonably fast diffusion.

Acknowledgments. We thank all reviewers of the FSE 2014 for their helpful comments and furthermore, we would like to thank Christian Forler, Ivica Nikolić, Douglas Shors, and Vesselin Velichkov for fruitful discussions.

References

1. Alizadeh, J., Bagheri, N., Gauravaram, P., Kumar, A., Sanadhya, S.K.: Linear Cryptanalysis of Round Reduced SIMON. Cryptology ePrint Archive, Report 2013/663 (2013). http://eprint.iacr.org/
2. Alkhzaimi, H.A., Lauridsen, M.M.: Cryptanalysis of the SIMON Family of Block Ciphers. Cryptology ePrint Archive, Report 2013/543 (2013). http://eprint.iacr.org/
3. Beaulieu, R., Shors, D., Smith, J., Treatman-Clark, S., Weeks, B., Wingers, L.: The SIMON and SPECK Families of Lightweight Block Ciphers. Cryptology ePrint Archive, Report 2013/404 (2013). http://eprint.iacr.org/
4. 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)
5. Biryukov, A., Velichkov, V.: Automatic Search for Differential Trails in ARX Ciphers (Extended Version). Cryptology ePrint Archive, Report 2013/853 (2013). http://eprint.iacr.org/
6. Bogdanov, A.A., Knudsen, L.R., Leander, G., Paar, C., Poschmann, A., Robshaw, M., Seurin, Y., Vikkelsoe, C.: PRESENT: An Ultra-Lightweight Block Cipher. In: Paillier, P., Verbauwhede, I. (eds.) CHES 2007. LNCS, vol. 4727, pp. 450–466. Springer, Heidelberg (2007)

7. Borghoff, J., et al.: PRINCE: A Low-Latency Block Cipher for Pervasive Computing Applications - Extended Abstract. In: Sako, K., Wang, X. (eds.) ASIACRYPT 2012. LNCS, vol. 7658, pp. 208–225. Springer, Heidelberg (2012)
8. De Cannière, C., Dunkelman, O., Knežević, M.: KATAN and KTANTAN: A Family of Small and Efficient Hardware-Oriented Block Ciphers. In: Clavier, C., Gaj, K. (eds.) CHES 2009. LNCS, vol. 5747, pp. 272–288. Springer, Heidelberg (2009)
9. Gong, Z., Nikova, S., Law, Y.W.: KLEIN: A New Family of Lightweight Block Ciphers. In: Juels, A., Paar, C. (eds.) RFIDSec 2011. LNCS, vol. 7055, pp. 1–18. Springer, Heidelberg (2012)
10. Guo, J., Peyrin, T., Poschmann, A., Robshaw, M.: The LED Block Cipher. In: Preneel, B., Takagi, T. (eds.) CHES 2011. LNCS, vol. 6917, pp. 326–341. Springer, Heidelberg (2011)
11. Hong, D., et al.: HIGHT: A New Block Cipher Suitable for Low-Resource Device. In: Goubin, L., Matsui, M. (eds.) CHES 2006. LNCS, vol. 4249, pp. 46–59. Springer, Heidelberg (2006)
12. Lim, C.H., Korkishko, T.: mCrypton - A Lightweight Block Cipher for Security of Low-Cost RFID Tags and Sensors. In: Song, J.-S., Kwon, T., Yung, M. (eds.) WISA 2005. LNCS, vol. 3786, pp. 243–258. Springer, Heidelberg (2006)
13. Lipmaa, H.: On Differential Properties of Pseudo-Hadamard Transform and Related Mappings. In: Menezes, A., Sarkar, P. (eds.) INDOCRYPT 2002. LNCS, vol. 2551, pp. 48–61. Springer, Heidelberg (2002)
14. Lipmaa, H., Moriai, S.: Efficient Algorithms for Computing Differential Properties of Addition. In: Matsui, M. (ed.) FSE 2001. LNCS, vol. 2355, pp. 336–350. Springer, Heidelberg (2002)
15. Matsui, M.: On Correlation Between the Order of S-boxes and the Strength of DES. In: De Santis, A. (ed.) EUROCRYPT 1994. LNCS, vol. 950, pp. 366–375. Springer, Heidelberg (1995)
16. Wu, W., Zhang, L.: LBlock: A Lightweight Block Cipher. In: Lopez, J., Tsudik, G. (eds.) ACNS 2011. LNCS, vol. 6715, pp. 327–344. Springer, Heidelberg (2011)

A Differential Characteristics for Simon and Speck

We use the following notions for the tables in this section. Each table contains at least a differential characteristic for one version of Simon or Speck. We denote by $\sum$ the total probability of the full characteristic, and by $\sum_{acc}$ the accumulated probability of all found trails from start to end difference.

Table 4. Differential characteristics for SIMON32/64 and SIMON48/k.

Rd.	SIMON32/64			SIMON48/k		
	ΔL^i	ΔR^i	$\log_2(p)$	ΔL^i	ΔR^i	$\log_2(p)$
0	0	Δ_6		$\Delta_{8,16}$	$\Delta_{6,14,18}$	
1	Δ_6	0	0	$\Delta_{6,10,14}$	$\Delta_{8,16}$	-4
2	Δ_8	Δ_6	-2	Δ_{12}	$\Delta_{6,10,14}$	-6
3	$\Delta_{6,10}$	Δ_8	-2	$\Delta_{6,10}$	Δ_{12}	-2
4	Δ_{12}	$\Delta_{6,10}$	-4	Δ_8	$\Delta_{6,10}$	-4
5	$\Delta_{6,10,14}$	Δ_{12}	-2	Δ_6	Δ_8	-2
6	$\Delta_{0,8}$	$\Delta_{6,10,14}$	-6	0	Δ_6	-2
7	$\Delta_{2,6,14}$	$\Delta_{0,8}$	-4	Δ_6	0	0
8	Δ_4	$\Delta_{2,6,14}$	-6	Δ_8	Δ_6	-2
9	$\Delta_{2,14}$	Δ_4	-2	$\Delta_{6,10}$	Δ_8	-2
10	Δ_0	$\Delta_{2,14}$	-4	Δ_{12}	$\Delta_{6,10}$	-4
11	Δ_{14}	Δ_0	-2	$\Delta_{6,10,14}$	Δ_{12}	-2
12	0	Δ_{14}	-2	$\Delta_{8,15,16}$	$\Delta_{6,10,14}$	-6
13	Δ_{14}	0	0	$\Delta_{6,14,18}$	$\Delta_{8,15,16}$	-6
14				Δ_{20}	$\Delta_{6,14,18}$	-6
15				$\Delta_{6,14,18,22}$	Δ_{20}	-2
Σ			-36			-50
Σ_{acc}			-30.22			-43.01

Table 5. Differential characteristics for SIMON64/k.

Rd.	ΔL^i	ΔR^i	$\log_2(p)$	Rd.	ΔL^i	ΔR^i	$\log_2(p)$
0	Δ_8	$\Delta_{6,10}$		11	$\Delta_{6,14,17,18}$	$\Delta_{14,15,20}$	-6
1	Δ_6	Δ_8	-2	12	$\Delta_{8,16}$	$\Delta_{6,14,17,18}$	-8
2	0	Δ_6	-2	13	$\Delta_{6,10,14}$	$\Delta_{8,16}$	-4
3	Δ_6	0	0	14	Δ_{12}	$\Delta_{6,10,14}$	-6
4	$\Delta_{7,8,14}$	Δ_6	-2	15	$\Delta_{6,10}$	Δ_{12}	-2
5	$\Delta_{6,10,16}$	$\Delta_{7,8,14}$	-6	16	Δ_8	$\Delta_{6,10}$	-4
6	Δ_{12}	$\Delta_{6,10,16}$	-6	17	Δ_6	Δ_8	-2
7	$\Delta_{6,10,14,16}$	Δ_{12}	-2	18	0	Δ_6	-2
8	$\Delta_{8,15,16,22}$	$\Delta_{6,10,14,16}$	-8	19	Δ_6	0	0
9	$\Delta_{6,14,18}$	$\Delta_{8,15,16,22}$	-8	20	Δ_8	Δ_6	-2
10	$\Delta_{14,15,20}$	$\Delta_{6,14,18}$	-6	21	$\Delta_{6,10}$	Δ_8	-2
Σ							-80
Σ_{acc}							-61.01

Table 6. Differential characteristics for Simon96/k and Simon128/k.

Rd.	Simon96/k			Simon128/k		
	ΔL^i	ΔR^i	$\log_2(p)$	ΔL^i	ΔR^i	$\log_2(p)$
0	Δ_{20}	$\Delta_{6,14,18,22}$		Δ_{12}	$\Delta_{6,10,14}$	
1	$\Delta_{6,14,18}$	Δ_{20}	-2	$\Delta_{6,10}$	Δ_{12}	-2
2	$\Delta_{8,16}$	$\Delta_{6,14,18}$	-6	Δ_{8}	$\Delta_{6,10}$	-4
3	$\Delta_{6,10,14}$	$\Delta_{8,16}$	-4	Δ_{6}	Δ_{8}	-2
4	Δ_{12}	$\Delta_{6,10,14}$	-6	0	Δ_{6}	-2
5	$\Delta_{6,10}$	Δ_{12}	-2	Δ_{6}	0	0
6	Δ_{8}	$\Delta_{6,10}$	-4	Δ_{8}	Δ_{6}	-2
7	Δ_{6}	Δ_{8}	-2	$\Delta_{6,10}$	Δ_{8}	-2
8	0	Δ_{6}	-2	Δ_{12}	$\Delta_{6,10}$	-4
9	Δ_{6}	0	0	$\Delta_{6,10,14}$	Δ_{12}	-2
10	Δ_{8}	Δ_{6}	-2	$\Delta_{8,15,16}$	$\Delta_{6,10,14}$	-6
11	$\Delta_{6,10}$	Δ_{8}	-2	$\Delta_{6,14,18}$	$\Delta_{8,15,16}$	-6
12	Δ_{12}	$\Delta_{6,10}$	-4	$\Delta_{14,15,20}$	$\Delta_{6,14,18}$	-6
13	$\Delta_{6,10,14}$	Δ_{12}	-2	$\Delta_{6,14,17,18}$	$\Delta_{14,15,20}$	-6
14	$\Delta_{8,15,16}$	$\Delta_{6,10,14}$	-6	$\Delta_{8,16}$	$\Delta_{6,14,17,18}$	-8
15	$\Delta_{6,14,18}$	$\Delta_{8,15,16}$	-6	$\Delta_{6,10,14}$	$\Delta_{8,16}$	-4

(*Continued*)

Table 6. (*Continued*)

Rd.	SIMON96/k			SIMON128/k		
	ΔL^i	ΔR^i	$\log_2(p)$	ΔL^i	ΔR^i	$\log_2(p)$
16	$\Delta_{14,15,20}$	$\Delta_{6,14,18}$	-6	Δ_{12}	$\Delta_{6,10,14}$	-6
17	$\Delta_{6,14,17,18}$	$\Delta_{14,15,20}$	-6	$\Delta_{6,10}$	Δ_{12}	-2
18	$\Delta_{8,16}$	$\Delta_{6,14,17,18}$	-8	Δ_{8}	$\Delta_{6,10}$	-4
19	$\Delta_{6,10,14}$	$\Delta_{8,16}$	-4	Δ_{6}	Δ_{8}	-2
20	Δ_{12}	$\Delta_{6,10,14}$	-6	0	Δ_{6}	-2
21	$\Delta_{6,10}$	Δ_{12}	-2	Δ_{6}	0	0
22	Δ_{8}	$\Delta_{6,10}$	-4	Δ_{8}	Δ_{6}	-2
23	Δ_{6}	Δ_{8}	-2	$\Delta_{6,10}$	Δ_{8}	-2
24	0	Δ_{6}	-2	Δ_{12}	$\Delta_{6,10}$	-4
25	Δ_{6}	0	0	$\Delta_{6,10,14}$	Δ_{12}	-2
26	Δ_{8}	Δ_{6}	-2	$\Delta_{8,15,16}$	$\Delta_{6,10,14}$	-6
27	$\Delta_{6,10}$	Δ_{8}	-2	$\Delta_{6,14,18}$	$\Delta_{8,15,16}$	-6
28	Δ_{12}	$\Delta_{6,10}$	-4	$\Delta_{14,15,20}$	$\Delta_{6,14,18}$	-6
29	$\Delta_{6,10,14}$	Δ_{12}	-2	$\Delta_{6,14,17,18}$	$\Delta_{14,15,20}$	-6
30	$\Delta_{8,16}$	$\Delta_{6,10,14}$	-6	$\Delta_{8,16}$	$\Delta_{6,14,17,18}$	-8
31				$\Delta_{6,10,14}$	$\Delta_{8,16}$	-4
32				Δ_{12}	$\Delta_{6,10,14}$	-6
33				$\Delta_{6,10}$	Δ_{12}	-2
34				Δ_{8}	$\Delta_{6,10}$	-4
35				Δ_{6}	Δ_{8}	-2
36				0	Δ_{6}	-2
37				Δ_{6}	0	0
38				Δ_{8}	Δ_{6}	-2
39				$\Delta_{6,10}$	Δ_{8}	-2
40				Δ_{12}	$\Delta_{6,10}$	-4
41				$\Delta_{6,10,14}$	Δ_{12}	-2
Σ			-106			-144
Σ_{acc}			-92.2			-124.6

Table 7. Differential characteristics for Speck32/64 and Speck48/k.

Rd.	Speck32/64			Speck48/k		
	ΔL^i	ΔR^i	$\log_2(p)$	ΔL^i	ΔR^i	$\log_2(p)$
0	$\Delta_{5,6,9,11}$	$\Delta_{0,2,9,14}$		$\Delta_{0,8,9,11,19,22}$	$\Delta_{0,3,14,16,19}$	
1	$\Delta_{0,4,9}$	$\Delta_{2,9,11}$	-5	$\Delta_{1,11,12,19}$	$\Delta_{1,3,6,11,17,22}$	-8
2	$\Delta_{11,13}$	Δ_4	-4	$\Delta_{1,4,6,22}$	$\Delta_{9,14,20,22}$	-7
3	Δ_6	0	-2	$\Delta_{9,17,23}$	$\Delta_{1,9,12}$	-5
4	Δ_{15}	Δ_{15}	0	$\Delta_{12,15}$	Δ_4	-4
5	$\Delta_{8,15}$	$\Delta_{1,8,15}$	-1	Δ_7	0	-2
6	Δ_{15}	$\Delta_{1,3,10,15}$	-2	Δ_{23}	Δ_{23}	0
7	$\Delta_{1,3,8,10,15}$	$\Delta_{5,8,10,12,15}$	-4	$\Delta_{15,23}$	$\Delta_{2,15,23}$	-1
8	$\Delta_{1,3,5,15}$	$\Delta_{3,5,7,10,12,14,15}$	-6	$\Delta_{2,7,23}$	$\Delta_{5,7,18,23}$	-3
9	$\Delta_{3,5,7,8,15}$	$\Delta_{0,1,3,8,9,12,14,15}$	-7	$\Delta_{5,7,15}$	$\Delta_{2,5,7,8,10,15,21}$	-4
10				$\Delta_{2,5,8,10,15,23}$	$\Delta_{0,2,11,13,15,18,23}$	-7
Σ			-31			-41
Σ_{acc}			-30.99			-40.55

Table 8. Differential characteristic for Speck96/k.

Rd.	ΔL^i	ΔR^i	$\log_2(p)$
0	$\Delta_{1,5,7,19,29,37,41,43,45}$	$\Delta_{0,11,19,21,22,29,32,33,37,41,44,45}$	
1	$\Delta_{0,19,22,32,35,44,47}$	$\Delta_{3,14,19,24,25,36,40}$	-13
2	$\Delta_{3,11,19,25,27,39}$	$\Delta_{3,6,11,17,19,22,25,28,43}$	-10
3	$\Delta_{6,22,25,28,31}$	$\Delta_{9,14,20,46}$	-10
4	$\Delta_{9,17,23}$	$\Delta_{1,9,12}$	-6
5	$\Delta_{12,15}$	Δ_4	-4
6	Δ_7	0	-2
7	Δ_{47}	Δ_{47}	0
8	$\Delta_{39,47}$	$\Delta_{2,39,47}$	-1
9	$\Delta_{2,31,47}$	$\Delta_{5,31,42,47}$	-3
10	$\Delta_{5,23,31,39,47}$	$\Delta_{2,5,8,23,31,34,39,45,47}$	-5
11	$\Delta_{2,5,8,15,34,47}$	$\Delta_{0,11,15,26,37,42,47}$	-9
12	$\Delta_{7,11,15,37,39,45,47}$	$\Delta_{2,3,7,11,14,15,18,29,37,39,40,47}$	-9
13	$\Delta_{2,11,14,15,18,31,40}$	$\Delta_{5,6,10,11,15,17,21,31,32,42,43}$	-12
Σ			-84
Σ_{acc}			-83.98

Table 9. Differential characteristic for SPECK64/k.

Rd.	ΔL^i	ΔR^i	$\log_2(p)$	Rd.	ΔL^i	ΔR^i	$\log_2(p)$
0	$\Delta_{5,21,24,27,30}$	$\Delta_{8,13,19,29}$		7	$\Delta_{4,6,7,14,22,30}$	$\Delta_{1,4,6,14,17,22,28,30}$	-7
1	$\Delta_{8,16,22}$	$\Delta_{0,8,11}$	-6	8	$\Delta_{1,4,7,17,31}$	$\Delta_{9,20,25}$	-9
2	$\Delta_{11,14}$	Δ_{3}	-4	9	$\Delta_{20,23,28,31}$	$\Delta_{12,20,31}$	-5
3	Δ_{6}	0	-2	10	$\Delta_{15,23,31}$	$\Delta_{2,31}$	-4
4	Δ_{30}	Δ_{30}	-1	11	$\Delta_{2,7,15,23,31}$	$\Delta_{5,7,15,23,31}$	-4
5	$\Delta_{22,30}$	$\Delta_{1,22,30}$	-2	12	$\Delta_{5,26}$	$\Delta_{2,5,8,10,18}$	-5
6	$\Delta_{1,14,30}$	$\Delta_{4,14,25,30}$	-4	13	$\Delta_{2,5,8,10,29}$	$\Delta_{2,10,11,13,21,29}$	-6
Σ							-59
Σ_{acc}							-58.9

Table 10. Differential characteristic for SPECK128/k.

Rd.	ΔL^i	ΔR^i	$\log_2(p)$
0	$\Delta_{5,10,16,26,27,35,37,42,48,49,54,58,60}$	$\Delta_{2,5,18,34,37,46,49,50}$	
1	$\Delta_{5,8,19,27,29,37,40,41,49,52,61}$	$\Delta_{19,21,27,29,41,53,61}$	-16
2	$\Delta_{0,11,22,27,28,32,33,44}$	$\Delta_{11,24,27,30,33,56}$	-13
3	$\Delta_{3,11,14,19,25,27,30,33,36}$	$\Delta_{3,11,19,25,59}$	-12
4	$\Delta_{6,17,22,28}$	$\Delta_{14,17,62}$	-9
5	$\Delta_{9,17,20}$	$\Delta_{1,9}$	-5
6	Δ_{12}	Δ_{4}	-3
7	0	Δ_{7}	-1
8	Δ_{7}	$\Delta_{7,10}$	-1
9	$\Delta_{7,10,63}$	$\Delta_{7,13,63}$	-2
10	$\Delta_{2,7,13,55}$	$\Delta_{7,10,13,16,55}$	-4
11	$\Delta_{5,7,10,13,16,47,55,58,63}$	$\Delta_{5,7,19,47,55,63}$	-8
12	$\Delta_{2,7,8,19,39,50,61}$	$\Delta_{7,10,19,22,39,58,61}$	-10
13	$\Delta_{0,7,10,19,22,31,39,42,53,61,63}$	$\Delta_{7,13,19,25,31,39,53,63}$	-13
14	$\Delta_{2,7,11,13,14,19,23,25,34,39,45,55,56}$	$\Delta_{7,10,11,13,14,16,19,22,23,25,28,39,42,45,55}$	-15
Σ			-112
Σ_{acc}			-111.16

B Rectangle Attacks on Speck

Table 11. Parameters of our rectangle attacks on Speck$2n/k$. $\hat{p}$ denotes the accumulated probability of the characteristics over E_1, $\hat{q}$ the probability of characteristics over E_2. Note that we prepend one round before E_1 and append one round after E_2 in our attacks.

Cipher	Rounds			$\hat{p}$	$\hat{q}$
	Attacked	E_1	E_2		
Speck32/64	11/22	5	4	$2^{-8.01}$	$2^{-4.56}$
Speck48/72	12/22	5	5	$2^{-9.06}$	$2^{-9.11}$
Speck48/96	12/23	5	5	$2^{-9.06}$	$2^{-9.11}$
Speck64/96	14/26	6	6	$2^{-15.02}$	$2^{-14.58}$
Speck64/128	14/27	6	6	$2^{-15.02}$	$2^{-14.58}$
Speck96/144	16/29	7	7	$2^{-22.46}$	$2^{-19.39}$
Speck128/192	18/33	8	8	$2^{-28.47}$	$2^{-28.39}$
Speck128/256	18/34	8	8	$2^{-28.47}$	$2^{-28.39}$

Table 12. Differential characteristics for our rectangle attacks on the individual versions of Speck. α denotes the input differences, δ the output differences.

Cipher	α	δ
Speck32/64	$(\Delta_{11,13}, \Delta_4)$	$(\Delta_{15}, \Delta_{1,3,10,15})$
Speck48/k	$(\Delta_{12,15}, \Delta_4)$	$(\Delta_{2,7,23}, \Delta_{5,7,18,23})$
Speck64/k	$(\Delta_{9,17,20}, \Delta_{1,9})$	$(\Delta_{1,14,30}, \Delta_{4,14,25,30})$
Speck96/k	$(\Delta_{9,17,23}, \Delta_{1,9,12})$	$(\Delta_{5,23,31,39,47}, \Delta_{2,5,8,23,31,34,39,45,47})$
Speck128/k	$(\Delta_{6,22,25,28,31}, \Delta_{9,14,20,62})$	$(\Delta_{2,5,8,31,50,63}, \Delta_{0,11,31,42,53,58,63})$

Differential Analysis of Block Ciphers SIMON and SPECK

Alex Biryukov, Arnab Roy(✉), and Vesselin Velichkov

Laboratory of Algorithmics, Cryptology and Security (LACS),
University of Luxembourg, Walferdange, Luxembourg
{Alex.Biryukov,Arnab.Roy,Vesselin.Velichkov}@uni.lu

Abstract. In this paper we continue the previous line of research on the analysis of the differential properties of the lightweight block ciphers Simon and Speck. We apply a recently proposed technique for automatic search for differential trails in ARX ciphers and improve the trails in Simon32 and Simon48 previously reported as best. We further extend the search technique for the case of differentials and improve the best previously reported differentials on Simon32, Simon48 and Simon64 by exploiting more effectively the strong differential effect of the cipher. We also present improved trails and differentials on Speck32, Speck48 and Speck64. Using these new results we improve the currently best known attacks on several versions of Simon and Speck. A second major contribution of the paper is a graph based algorithm (linear time) for the computation of the exact differential probability of the main building block of Simon: an AND operation preceded by two bitwise shift operations. This gives us a better insight into the differential property of the Simon round function and differential effect in the cipher. Our algorithm is general and works for any rotation constants. The presented techniques are generic and are therefore applicable to a broader class of ARX designs.

Keywords: Symmetric-key · Differential trail · Tools for cryptanalysis · Automatic search · ARX · Simon · Speck · Lightweight ciphers

1 Introduction

The past decade in technology has been marked by the ever decreasing size of computing devices. This, in combination with their increasingly ubiquitous use e.g. as smart devices, wearable systems, as part of the *Internet of Things* [12], has enabled humans to perform everyday activities more efficiently. At the same time these new technologies have also created new security challenges.

An important problem today is the design of cryptographic algorithms that are both efficient and secure, have small memory footprint and are low-cost and easy to implement and deploy on multiple platforms. Finding an optimal compromise between these, often conflicting, requirements is the difficult area researched by the field of *lightweight cryptography*. The applications of lightweight cryptographic

C. Cid and C. Rechberger (Eds.): FSE 2014, LNCS 8540, pp. 546–570, 2015.
DOI: 10.1007/978-3-662-46706-0_28

algorithms vary from mobile devices, through RFID tags to electronic locks and their importance is likely to continue increasing in the future.

To address the persistent need for secure and efficient lightweight primitives, numerous proposals have been made in the past few years. In the area of symmetric-key encryption some of the more prominent block ciphers that were proposed are: PRESENT [6], PICCOLO [17], KLEIN [9], TWINE [19], KATAN and KTANTAN [7], LED [10], HIGHT [11] and CLEFIA [18].

Most recently, in June 2013, yet two more algorithms have been put forth by researchers from the National Security Agency (NSA) of the USA – the block ciphers SIMON and SPECK [4]. Compared to their predecessors, the latter two have very competitive performance, small memory footprint and beat most existing lightweight ciphers in terms of efficiency and compactness. Furthermore, the two designs are very simple and elegant. They are both built on the ARX philosophy [13,20], using only basic arithmetic operations such as modular addition, `XOR`, bitwise `AND` and bit rotation.

Evidence of the performance and implementation advantages of SIMON and SPECK exists in the form of extensive results and comparisons to existing lightweight algorithms described in the design document [4]. However the latter does not provide any security evaluation of the two ciphers and no analysis of their cryptographic strength is given. Recently several external cryptanalytic results on SIMON and SPECK became available: [1–3]. The first two in particular analyze the differential properties of the ciphers and describe key-recovery attacks on reduced round variants.

Our Contribution. In this paper we further investigate the differential behavior of block ciphers SIMON and SPECK. We apply a recently proposed technique for automatic search for differential trails in ARX ciphers called *threshold search* [5]. We find better differential trails on SIMON32 and SIMON48 than the ones reported by [2] and claimed to be the best and, we confirm the trail on SIMON64. Improved trails that cover one round more than the previously reported best trails [1] on SPECK32, SPECK48 and SPECK64 are found. We further extend the threshold search technique for finding differentials. With the new tool we improve the differentials on SIMON32, SIMON48 and SIMON64 reported by [2] and we present new differentials on SPECK32, SPECK48 and SPECK64. We use these new results to improve the currently best known attacks on several versions of SIMON and SPECK.

The second major contribution of the paper is an efficient algorithm for the computation of the differential probabilities (DP) of the bitwise `AND` operation – the single source of non-linearity in the round function of SIMON. We describe algorithms for the computation of the exact DP of `AND` with independent inputs and rotationally dependent inputs (one input is equal to the rotation of the other one) as used in SIMON. In addition, methods for computing the maximum DP over all inputs and over all outputs of the `AND` operation are also proposed. All described algorithms have linear time complexity in the word size. These algorithms are used in the *threshold* search and in the differential search tool for SIMON.

Table 1. Summary of attacks on SIMON and SPECK. All listed attacks are chosen-plaintext.

Cipher	Key Size	#Rounds Total	#Rounds Attacked	Time Sects. 7, 8	Data Sects. 7, 8	Time [1,2]	Data [1,2]
SIMON32	64	32	18			2^{62}[a]	$2^{31.2}$
	64	32	19	2^{34}	$2^{31.5}$		
	64	32	19	2^{40}	2^{31}		
SIMON48	72	36	19	2^{46}	2^{46}	2^{52}	2^{46}
	72	36	20	2^{52}	2^{46}		
	96	36	19	2^{69}	2^{46}	2^{76}	2^{46}
	96	36	20	2^{75}	2^{46}		
SIMON64	96	42	26	2^{89}	2^{63}	2^{94}[b]	2^{63}
	128	44	26	2^{121}	2^{63}	2^{126}[c]	2^{63}
SPECK32	64	22	10			$2^{29.2}$	2^{29}
	64	22	11	2^{55}	2^{31}		
SPECK48	72/96	22	12	2^{43}	2^{43}	$2^{45.3}$	2^{45}
SPECK64	96/128	26	15			$2^{61.1}$	2^{61}
	96	26	16	2^{80}	2^{63}		
	96	26	16	2^{73}	2^{64}		
	128	27	16	2^{80}	2^{63}		
	128	27	16	2^{73}	2^{64}		

[a,b,c]Differs from [2]

Finally, we briefly comment on the strong differential effect in SIMON – a property already noted in [3]. In addition we provide new insights into the clustering of differential trails that causes this effect. A summary of the main results from the search on trails and differentials is provided in Table 2. Note that in this table is mentioned a figure for the time complexity of the attacks on SIMON32 and SIMON64 described in [2] that we were not able to verify (Table 1).[1]

The outline of the paper is as follows. We begin with Sect. 2 where the XOR differential probability of the AND operation is analyzed. Next in Sect. 3 are presented techniques for searching for trails and differentials in ARX algorithms. The block ciphers SIMON and SPECK are briefly described in Sect. 4. Full differential trails are presented in Sect. 5. Finally, in Sect. 6 we comment on the strong differential effect of SIMON. Section 9 concludes the paper.

A few words on notation: with x_i is denoted the i-th bit of the n-bit word x (x_0 is the LSB); $\overline{x_i}$ represents the modulo-2 complementation of x_i i.e. $\overline{x_i} = x_i \oplus 1$; the symbols $\wedge$ and $\vee$ denote respectively bitwise logical AND and OR operations; the left and right rotation of the bits of x by r positions is denoted respectively

[1] Since with one good pair in all the data the counting phase does not give the attacker unique partial key.

with $x \lll r$ and $x \ggg r$; $|S|$ represents the cardinality of the set S. The concatenation of the bit strings x and y is denoted by $x|y$.

2 The XOR Differential Probability of AND

2.1 Independent Inputs

Definition 1 ($\mathrm{xdp}^{\&}$ **with independent inputs).** *Let α, β and γ be fixed n-bit XOR differences. The XOR differential probability (DP) of the logical AND operation ($\mathrm{xdp}^{\&}$) is the probability with which α and β propagate to γ through the AND operation, computed over all pairs of n-bit inputs (x, y):*

$$\mathrm{xdp}^{\&}(\alpha, \beta \to \gamma) = 2^{-2n} \cdot |\{(x, y) : ((x \oplus \alpha) \wedge (y \oplus \beta)) \oplus (x \wedge y) = \gamma\}| \ . \quad (1)$$

In the remaining of the text the acronym DP will be used to denote *XOR differential probability* unless specified otherwise. When the input differences α and β

Table 2. Summary of the best found differential trails and differentials in SIMON and SPECK;

Cipher	# rounds	$\log_2 p$, trail	$\log_2 p$, diff	# trails	ref
SIMON32	12	−34			Sect. 5
		−36			[2]
	13	−36	−29.69	45083	Sect. 5
			−28.56	Experimental[a]	Sect. 5
		−36	−30.20		[2]
SIMON48	15	−48	−42.11	112573	Sect. 5
		−52	−43.01		[2]
SIMON64	20	−70	−58.07	533163	Sect. 5
		−70	−59.01		[2]
	21	−72	−60.40	450536	Sect. 5
		−72	−61.01		[2]
SPECK32	8	−24	−24	1	Sect. 5
		−24	−24	1	[1]
	9	−31	−31	1	Sect. 5
SPECK48	10	−40	−39.75	137	Sect. 5
			−40.55		[1]
	11	−47	−46.48	384	Sect. 5
SPECK64	13	−58	−57.67	198	Sect. 5
			−58.90		[1]
	14	−60	−59.02	934	Sect. 5

[a]Measured as the average prob.over 128 keys chosen at random and using the full codebook.

are independent, the DP $\mathrm{xdp}^{\wedge}(\alpha, \beta \rightarrow \gamma)$ can be efficiently computed according to the following theorem.

Theorem 1. *For fixed n-bit XOR differences* α, β *and* γ *the probability* $\mathrm{xdp}^{\&}(\alpha, \beta \rightarrow \gamma)$ *is equal to*

$$\mathrm{xdp}^{\&}(\alpha, \beta \rightarrow \gamma) = 2^{-n} \cdot \prod_{i=0}^{n-1} \left((2 \cdot (\overline{\alpha}_i \wedge \overline{\beta}_i \wedge \overline{\gamma}_i)) \vee (\overline{\overline{\alpha}_i \wedge \overline{\beta}_i}) \right) \wedge \overline{(\overline{\alpha}_i \wedge \overline{\beta}_i \wedge \gamma_i)} \ . \tag{2}$$

Proof. Note that $\mathrm{xdp}^{\&}(\alpha, \beta \rightarrow \gamma) = 0 \iff \exists i : \ 0 \leq i < n : \ (\overline{\alpha}_i \wedge \overline{\beta}_i \wedge \gamma_i) = 1$. Therefore whenever the probability is zero, the term $(\overline{\alpha}_i \wedge \overline{\beta}_i \wedge \gamma_i)$ evaluates to zero and hence the right hand-side of (2) is also zero. If the probability is non-zero and $\alpha_i = \beta_i = \gamma_i = 0$ at bit position i then $(\overline{\alpha}_i \wedge \overline{\beta}_i \wedge \overline{\gamma}_i) = 1$ which is multiplied by the number of valid pairs (x_i, y_i) (cf. Definition 1) i.e. 4 . If $\alpha_i \neq \beta_i$ then exactly two pairs (x_i, y_i) satisfy the differential at bit position i irrespective of the value of γ_i. In this case $(\overline{\overline{\alpha}_i \wedge \overline{\beta}_i}) = 1$ and it is multiplied by the number of valid pairs (x_i, y_i) which is 2. Therefore for non-zero probability, the product on the right-hand size of (2) is a multiple of 2^n. The latter cancels with the term 2^{-2n} (cf. Definition 1) and so the final expression is multiplied by 2^{-n}. □

Theorem 1 implies the following corollary.

Corollary 1. *Given n-bit input differences* α, β *and output difference* γ*, the probability* $\mathrm{xdp}^{\&}(\alpha, \beta \rightarrow \gamma)$ *can be computed in* $\mathcal{O}(n)$ *time.*

Proof. Follows directly from Theorem 1. □

2.2 Rotationally Dependent Inputs

Note that when the inputs to the `AND` operation are dependent on each other, the DP computed with Theorem 1 is not accurate. In particular, let the two inputs x, y to `AND` be such that $y = (x \lll r)$. So, an input `XOR` difference α applied to x will result into an input difference $(\alpha \lll r)$ to y. Considering the dependencies between the input variables, the DP in this case is defined as follows:

Definition 2 ($\mathrm{xdp}^{\&}$ with dependent inputs). *For a fixed rotation constant* r *and n-bit input difference* α*, the DP of the bitwise* `AND` *operation is defined as*

$$\begin{aligned} &\mathrm{xdp}^{\&}(\alpha, (\alpha \lll r) \rightarrow \gamma) \\ &\quad = 2^{-n} \cdot \left| \{x : \left(x \wedge (x \lll r)\right) \oplus \left((x \oplus \alpha) \wedge ((x \oplus \alpha) \lll r)\right) = \gamma\} \right| \ . \end{aligned} \tag{3}$$

In the following part of this section we describe a method for the computation of the probability $\mathrm{xdp}^{\&}(\alpha, (\alpha \lll r) \rightarrow \gamma)$ (3) in linear time in the word size n. We begin by stating several necessary definitions and lemmas.

A *cycle* of *length* t is a special subset of the set of indices $\mathcal{I} = \{0, 1, \ldots, n-1\}$ $(= \mathbf{Z}_n)$ indicating the bit positions of an n-bit word x (index 0 denotes the LS bit of x). More formally:

Definition 3. *A cycle of length t is a set of bit indices $C_i = \{i, i+r, i+2r, \ldots, i+(t-1)r\} \subseteq \mathcal{I}$, where $t \in \mathbf{N}$ is such that $i + tr = i \mod (n)$ and i is the smallest element of C_i.*

In a cycle C_i of length t, $i + (s-1)r$ is said to be *preceding* element to $i + sr$ $(1 < s < t)$. Moreover, $i + (t-1)r$ is *preceding* element to i. Since each C_i for $i \in \mathcal{I}$ is an equivalence class, $\mathcal{I}$ can be partitioned into disjoint cycles:

Lemma 1. *For fixed $r \in \mathcal{I}$, $C_i \cap C_j = \emptyset$ iff $i \neq j : \ 0 \leq i, j < n$ and, $\mathcal{I} = \bigcup C_i$.*

Example 1. For $n = 8$ and $r = 2$ there are exactly 2 cycles each having length $t = 4$: $C_0 = \{0, 2, 4, 6\}$ and $C_1 = \{1, 3, 5, 7\}$.

By Definition 2, computing the probability $\text{xdp}^{\&}$ is equivalent to counting the number of values x that satisfy the differential $(\alpha, (\alpha \lll r) \rightarrow \gamma)$. For simplicity, let r be such that the set of bit indices $\mathcal{I}$ of x has a single cycle $C_0 = \{0, r, 2r, \ldots, n-r\}$. Within this cycle the bits of the input and output differences are represented as a sequence of 3-tuples in the following way:

$$(\alpha_0, \alpha_{(n-r)}, \gamma_0),\ (\alpha_r, \alpha_0, \gamma_r),\ (\alpha_{2r}, \alpha_r, \gamma_{2r}), \ldots, (\alpha_{(n-r)}, \alpha_{(t-1)r}, \gamma_{(n-r)})\ . \quad (4)$$

Note that in sequence (4), for each 3-tuple the index of the second element is a preceding index of the index of the first element.

Example 2. Let $n = 5$ and $r = 2$. Consider the input differences $\alpha = \alpha_4\alpha_3\alpha_2\alpha_1\alpha_0$ and $(\alpha \lll 2) = \alpha_2\alpha_1\alpha_0\alpha_4\alpha_3$ and the output difference $\gamma = \gamma_4\gamma_3\gamma_2\gamma_1\gamma_0$. In this case there is a single cycle C_0 of length $t = 5$: $C_0 = \{0, 1, 2, 3, 4\}$. The corresponding sequence of 3-tuples is:

$$(\alpha_0, \alpha_3, \gamma_0),\ (\alpha_2, \alpha_0, \gamma_2),\ (\alpha_4, \alpha_2, \gamma_4),\ (\alpha_1, \alpha_4, \gamma_1),\ (\alpha_3, \alpha_1, \gamma_3)\ . \quad (5)$$

The difference 3-tuples in (4) are satisfied by a number of possible bit assignments of x at the corresponding positions: (x_0, x_{n-r}), (x_r, x_0), $(x_{2r}, x_r), \ldots,$ $(x_{n-r}, x_{(t-1)r})$. In order to efficiently count the number of such assignments we use a variant of the technique proposed in [15] for the computation of the DP of modular addition and XOR.

Any 2-tuple of bits of the form $(x_{sr}, x_{(s-1)r})$ can have 4 values $\{(0,0), (0,1), (1,0), (1,1)\}$, where $(0 \leq s \leq n-1)$. These are viewed as a nodes of a graph. In total, for the full word length n the graph has $4n$ nodes. A *valid* assignment of two consecutive 2-tuples $(x_{sr}, x_{(s-1)r})$ and $(x_{(s+1)r}, x_{sr})$, $(0 \leq s < n-1)$ is represented as a directed edge between the corresponding nodes. In this way we can construct a *directed acyclic graph* (DAG) composed of $(n-1)$ edge-disjoint *bipartite* subgraphs. Each bipartite subgraph is formed by the nodes from two consecutive 2-tuples of bits of x and the edges between them. A *valid* path from an initial node (x_0, x_{n-r}) to a final node (x_{n-r}, x_{n-2r}) in the DAG corresponds to a value of x that satisfies the differential $(\alpha, (\alpha \lll r) \rightarrow \gamma)$. A path is said to be *valid* iff the initial and final nodes (x_0, x_{n-r}) and (x_{n-r}, x_{n-2r}) are consistent i.e. the value assigned to x_{n-r} in both nodes is the same.

The DAG constructed as explained above is represented as a sequence of 4×4 adjacency matrices, each corresponding to one bipartite subgraph. Computing the probability $\mathrm{xdp}^{\wedge}$ is then equivalent to counting the number of valid paths in the DAG. This can be performed in linear time in the word size by a sequence of $(n-1)$ multiplications of adjacency matrices. If the number of such paths is N, the final probability $\mathrm{xdp}^{\&}(\alpha, \beta \rightarrow \gamma)$ is $\frac{N}{2^n}$. This process is further illustrated with the Example 3.

In case of more than one cycle the described process can be performed independently for each cycle $C^j, 0 \leq j < m$ due to the fact that all cycles are disjoint (cf. Lemma 1). Let N_j be the number of paths in the DAG for the j-th cycle. Then the DP is given by $\frac{\prod_1^m N_j}{2^n}$.

Example 3. Assume the same setting as in Example 2: $n = 5$, $r = 2$ and let $\alpha = 00110_2$ and $\gamma = 00000_2$. Consider the resulting sequence of 3-tuples (5). In the DAG (Fig. 1), the dependency between the bits of x corresponding to two consecutive 3-tuples must be satisfied. For example, an edge between $(x_0, x_3) = (0, 1)$ corresponding to $(\alpha_0, \alpha_3, \gamma_0) = (1, 0, 0)$ and $(x_2, x_0) = (1, 0)$ corresponding to $(\alpha_2, \alpha_0, \gamma_2) = (0, 1, 0)$ is drawn, because $x_0 = 0$ for both the nodes. However there is no edge between $(x_2, x_0) = (0, 0)$ and $(x_4, x_2) = (0, 1)$ since, x_0 is not equal for both the nodes.

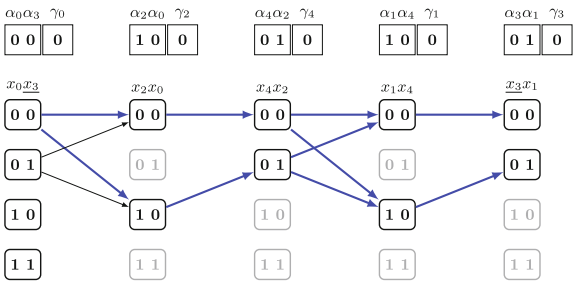


Fig. 1. DAG used in the computation of $\mathrm{xdp}^{\&}(\alpha, (\alpha \lll r) \rightarrow \gamma)$ for $n = 5$, $r = 2$, $\alpha = 00110_2$, $\gamma = 00000_2$. Every path composed of thick edges is a valid path and hence a valid assignment of bits of x. The fading nodes denote the bit assignments of x which do not satisfy the input output difference

A *valid* path from an initial node (corresponding to the first 3-tuple $(\alpha_0, \alpha_3, \gamma_0)$ in the sequence (5)) to a final node (corresponding to the last 3-tuple $(\alpha_3, \alpha_1, \gamma_3)$) in this graph is equivalent to a value of x that satisfies the differential. A valid path implies that the initial and final nodes are consistent with each other. For example, no path from the initial node $(x_0, x_3) = (0, 1)$ is valid, because, all final node have $x_3 = 0$. Since the total number of valid paths in the graph is $N = 4$ the DP is $\frac{4}{2^5} = 0.125$.

The method for the computation of the probability $\mathrm{xdp}^{\&}(\alpha, \beta \rightarrow \gamma)$ described above supports the following proposition.

Proposition 1. *For fixed n-bit differences α and γ, the probability* $\mathrm{xdp}^{\&}(\alpha, (\alpha \lll r) \rightarrow \gamma)$ *can be computed in $\mathcal{O}(n)$ time.*

Impossible Input-Output Difference. For a given (α, γ) an impossible difference can be of two types. Any input/output difference which leads to a difference 3-tuple $(\alpha_i, \alpha_{i-r}, \gamma_i) = (0, 0, 1)$, is an impossible input/output difference. The other types of input difference can be detected while computing the probability of the corresponding difference. Note that in the corresponding DAG, a path can be invalid even if every bipartite directed subgraph valid, e.g. $(\alpha, \gamma) = (11111_2, 00000_2)$. The following DAG shows this case.

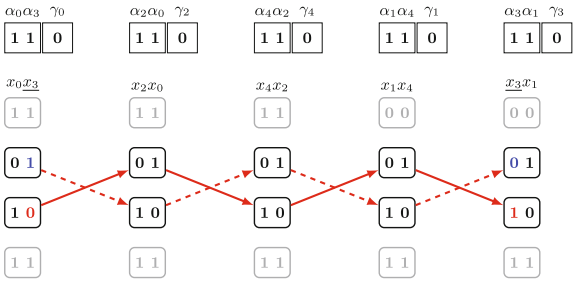


Fig. 2. DAG used in the computation of $\mathrm{xdp}^{\&}(11111_2, (11111_2 \lll 2) \to 00000_2)$. Both the paths, composed of thick edges and dashed edges, are invalid path, since there is contradiction in the bit value x_3. However, each directed bipartite subgraph is independently valid.

Proposition 2. *For a fixed n-bit input difference α and rotation r,*

$$\mathrm{DP}_{\max}(\alpha) = \max_{\gamma} \mathrm{xdp}^{\&}(\alpha, \alpha \lll r \to \gamma) \tag{6}$$

can be computed in $\mathcal{O}(n)$ time.

Finally, we note that the approach described above bears some similarity to the technique proposed in [15] for the computation of the DP of modular addition and XOR. Similarly to [15] we also map the problem of computing differential probabilities to the well-studied problem in graph theory of counting the number of paths in a graph. Apart from this similarity however, we would like to stress that the described method is fundamentally different than [15]. In the latter the nodes of the graph represent information that is propagated over the bit positions (namely, the carries and borrows resulting from the modular addition) and the edges represent the actual values of the pairs. In our case, the nodes of the graph represent the values of the pairs, while the edges describe the valid connections between the bits of those values so that the correct dependence due to the rotation operation is preserved.

3 Automatic Search for Trails and Differentials

3.1 Threshold Search

In [14] Matsui proposed a practical algorithm for finding the best differential trail for the DES block cipher. Given the best trail on i rounds and an over-

estimation of the best probability for $i+1$ rounds the algorithm finds the best trail on $i+1$ rounds. Starting from $i=1$ these steps are repeated recursively until $i+1=n$. In the process, the differential probabilities of the non-linear components of the target cipher (the S-boxes in the case of DES) are obtained from a pre-computed difference distribution table (DDT). The differentials for the i-th round are then processed in sorted order by probability. For each, the recursion proceeds to round $i+1$) only if the estimated probability of the trail for n rounds is equal to- or greater than the initial estimate.

Recently, in [5] a variant of Matsui's algorithm is proposed which is applicable to the class of ARX ciphers. What is special about the latter is that they do not have S-boxes. Instead they rely on basic arithmetic operations such as addition modulo n to achieve non-linearity. Computing a full DDT for the modular addition operation would require 4×2^{3n} bytes of memory and is therefore impractical for $n > 16$. To address this, in [5] a partial DDT (pDDT) rather than the full DDT is computed. A pDDT contains (a fraction of) all differentials that have probability above a fixed probability threshold (hence the name – *threshold search*).

Since some (possibly many) differentials are missing from the initial (also called *primary*) pDDT, at some point during the search it is likely that for a given input difference the algorithm will require a matching differential that is not present in the primary pDDT. Such differentials are computed on-demand and are stored in a *secondary* pDDT maintained dynamically during the search.

In order to prevent the size of the secondary pDDT from exploding while at the same time keeping the probability of the constructed trails high, [5] further introduce the notion of *highways* and *country roads* – resp. high and low probability differentials (w.r.t. the fixed threshold). Every differential from the primary pDDT is a *highway* while every differential from the secondary pDDT is a *country road*.

To further control the size of the country roads table, additional restrictions on the considered differences can be added. For example, it may be required that every country road at given round i is such that there is at least one transition at round $i+1$ that is a highway. This reduces the number of possible country roads while at the same time ensures that the considered paths have relatively high probability. This condition has been applied in the trail search for Simon. Another restriction can be on the Hamming weight of the considered differences. Such restriction has been applied in the differential search on Speck.

Several parameters control the performance of the threshold search technique. The most important ones are the probability threshold, which determines which differentials are considered as highways and the maximum size of the primary pDDT (note that it may be infeasible to compute and/or store all differentials that have probability above the threshold). The probability threshold influences the probability of the final trail: the lower the threshold, the more paths are considered and hence the more likely to find a high probability trail. At the same time, with the increase of the number of explored paths, the complexity of the algorithm also grows and hence it takes longer to terminate. The maximum size of the primary pDDT determines the precomputation time and the memory requirements for the algorithm.

3.2 Extension to Differentials

We further extend the method outlined above to the case of differentials. Given the best trail found by the threshold search and the corresponding array of best found probabilities for each round, a differential search proceeds according to the above strategy but always starting from the same input difference (corresponding to the best found trail). At every round are explored only paths whose estimated probabilities are by at most a factor ε away from best probability (e.g. $\varepsilon = 2^{-15}$). For example, let $B_i : 1 \leq i \leq n$ be the probabilities of the best found differentials resp. for $1, 2, \ldots, n$ rounds computed with the threshold search. Denote with $p_1, p_2, \ldots, p_{r-1}$ the probabilities of a partially constructed trail up to round $r-1$. At round r the differential search will explore all transitions that have probability $p_r \geq (\varepsilon B_n)/(p_1 \ldots p_{r-1} B_{n-r})$. A pseudocode of this procedure applied to SIMON is listed in Algorithm 1.

Algorithm 1. Search for clusters of trails belonging to the same differential.

Input: $\mathbf{T} = (\mathbf{T_0}, \ldots, \mathbf{T_n})$: the best found trail for n rounds with prob. B_n; $\mathbf{B} = (\mathbf{B_1}, \mathbf{B_2}, \ldots, \mathbf{B_n})$: probs. of best found trails for up to n rounds; **r**: current round; **H**: pDDT (the highway table) for the non-linear component f of the round function (e.g. for SIMON $f(x) = (x \lll 1) \wedge (x \lll 8)$; for SPECK f is the modular addition); ε: the algorithm searches for trails with probs. at most ε times worse than the best found prob.

Output: Cluster of trails Ω for the differential $(T_0 = (\alpha_0, \beta_0) \rightarrow T_n = (\alpha_n, \beta_n))$ and probabilitty $p_\Omega(T_0 \rightarrow T_n) \geq B_n$.

$(\alpha_0, \beta_0) \leftarrow T_0$; $(\alpha_n, \beta_n) \leftarrow T_n$; $\Omega \leftarrow T$; $T \leftarrow \emptyset$; $p_\Omega \leftarrow B_n$; $r \leftarrow 1$// Initialization
procedure cluster_trails$(n, r, \alpha_{r-1}, \beta_{r-1}, H, T, \Omega, p_\Omega)$ **do**
    if $r = n$ **then**
        // If at last round and trail matches output diff. add it to cluster and update the probability
        if $T_n = (\alpha_n, \beta_n)$ **then**
            add T to Ω; $p_\Omega \leftarrow p_\Omega + \overline{B}_n$; $T \leftarrow \emptyset$
        return
    if $r = 0$ **then**
        $p_0 = 1$; $T \leftarrow T_0 = (\alpha_0, \beta_0, p_0)$.
    $C \leftarrow \emptyset$ // Initialize the country roads table
    $p_{r,\min} \leftarrow (\varepsilon B_n)/(p_1 p_2 \cdots p_{r-1} B_{n-r})$ // The min. permissible probability for the new trail
    for all $\gamma_r : (p_r(\alpha_{r-1} \xrightarrow{f} \gamma_r) \geq p_{r,\min}) \wedge ((\alpha_{r-1}, \gamma_r, p_r) \notin H)$ **do**
        add $(\alpha_{r-1}, \gamma_r, p_r)$ to C // Update country roads table
    for all $(\alpha, \gamma, p) : \alpha = \alpha_{r-1}$ in H and all (α, γ, p) in C **do**
        $p_r \leftarrow p$, $\overline{B}_n \leftarrow p_1 p_2 \ldots p_r B_{n-r}$
        // Proceed to next round only if the estimated prob. is at most ε times worse than the best
        if $\overline{B}_n \geq (\varepsilon B_n)$ **then**
            $\alpha_r = \gamma_r \oplus \beta_{r-1} \oplus (\alpha_{r-1} \lll 2)$; $\beta_r \leftarrow \alpha_{r-1}$
            add $T_r = (\alpha_r, \beta_r, p_r)$ to T
            call **cluster_trails**$(n, r+1, \alpha_r, \beta_r, H, T, \Omega, p_\Omega)$
    return Ω, p_Ω

Note that a somewhat similar branch-and-bound approach has been applied by [1–3] to search for differentials in SIMON. The main difference is that according to the cited technique, at every round is maintained an array of the best differentials encountered so far ranked by probability. The search proceeds to the next round by considering the top N such differentials.

In our approach instead of storing intermediate differentials, we prune the search tree by limiting the search to an ε region within the best found probability, since the latter is already known from the threshold search.

Note that although the proposed technique searches for differentials starting with best trail found with the threshold search, it can easily be modified to search for multiple input and output differences, while keeping track of the best one. Finally, in order to improve the efficiency, the differential search can be further parametrized by limiting the maximum Hamming weight of the differences.

4 Description of SIMON and SPECK

The SIMON and SPECK families of lightweight block ciphers are defined for word sizes $n = 16, 24, 32, 48$ and 64 bits. The key is composed of m n-bit words for $m = 2, 3, 4$ (i.e. the key size mn varies between 64 and 256 bits) depending on the word size n. The block cipher instances corresponding to a fixed word size n (block size $2n$) and key size mn are denoted by SIMON$2n/mn$ and SPECK$2n/mn$.

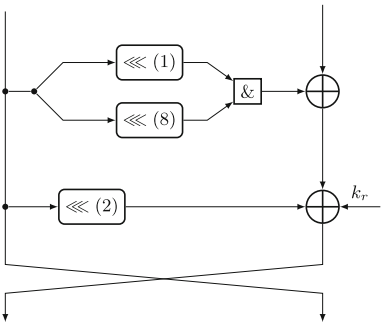


Fig. 3. SIMON round function

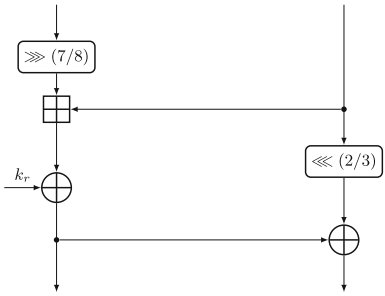


Fig. 4. SPECK round function

Block cipher SIMON has Feistel structure and its round function under a fixed round key k is defined on inputs x and y as:

$$R_k(x, y) = ((y \oplus f(x) \oplus k), x). \tag{7}$$

The function $f(\cdot)$ is defined as $f(x) = ((x \lll 1) \wedge (x \lll 8)) \oplus (x \lll 2)$, where the symbol $\wedge$ denotes the logical `AND` operation.

Block cipher SPECK has structure similar to Threefish – the block cipher used in the hash function Skein [8]. Its round function under a fixed round key k is defined on inputs x and y as:

$$R_k(x, y) = (f_k(x, y),\ f_k(x, y) \oplus (y \lll \beta)), \tag{8}$$

where the function $f_k(\cdot, \cdot)$ is defined as $f_k(x, y) = ((x \ggg \alpha) + y) \oplus k$. The rotation constants are $\alpha = 7, \beta = 2$ for block size 32 bits and $\alpha = 8, \beta = 3$ for all other block sizes. Although SPECK is not a Feistel cipher itself, it can be represented as a composition of two Feistel maps as described in [4]. The round functions of SIMON and SPECK are shown in Figs. 3 and 4 respectively. The number of rounds, block size and key size of the block ciphers are summarized in Tables 3 and 4.

Table 3. Parameters for SIMON

Block size	Key size	Key words	Rounds
32	64	4	32
48	72	3	36
	96	4	36
64	96	3	42
	128	4	44

Table 4. Parameters for SPECK

Block size	Key size	Key words	Rounds
32	64	4	22
48	72	3	22
	96	4	23
64	96	3	26
	128	4	27

5 Application to SIMON and SPECK

The trails obtained by using the threshold search technique and the differentials found with differential search tool (both described in Sect. 3) are presented in this section. The best found trails for SIMON and SPECK are shown respectively on Tables 5 and 6. In the tables, $\sum_r \log_2 p_r$ represents the probability of a single trail obtained as the sum of the probabilities of its transitions; p_{diff} is the probability of the corresponding differential and #trails is the number of trails clustered in the differential; max HW is the maximum Hamming weight allowed for the differences during the search; p_{thres} is the probability threshold used in the threshold search algorithm and pDDT denotes the number of elements in the partial DDT.

Note that all trails shown in Tables 5 and 6 were found using the technique described in Sect. 3 by starting the search from the top round and proceeding downwards. The only exception is the trail on SPECK48. Since this trail begins with a very low probability transition, when starting the search from the first round, it was computationally feasible to construct the shown trail only up to round 6. The full trail on 11 rounds shown in Table 6 was found by starting the search from a middle round (round 6) as has also been done in [1].

6 Differential Effect in SIMON

The clustering of multiple trails satisfying the same input/output difference (differential effect) in SIMON can be visualized by the digraph in Fig. 9. It depicts a cluster of more than 275 000 trails satisfying the 21 round differential $(\texttt{4000000}, \texttt{11000000}) \xrightarrow{21R} (\texttt{11000000}, \texttt{4000000})$. The thickness of an edge in the digraph is proportional to the probability of the corresponding input and output difference connected by this edge.

An interesting property clearly visible in the digraph in Fig. 9 is that it is composed of multiple smaller subgraphs positioned at alternate levels. Each such subgraph represents a biclique. Clearly, the bigger the number and size of such bicliques, the stronger the differential effect would be and hence the larger the probability of the differential. Therefore, the ability to obtain good estimation of the probability of a given differential for SIMON is intimately related to the ability to identify and characterize such complete bipartite subgraphs. Thus we take a closer look into those special structures below.

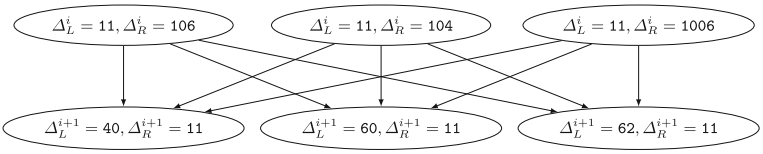


Fig. 5. Example of a bipartite subgraph embedded in the differential (graph) of Simon32.

Table 5. Differential trails for Simon32, Simon48 and Simon64.

	Simon32			Simon32			Simon48			Simon64		
r	Δ_L	Δ_R	$\log_2 p$	Δ_L	Δ_R	$\log_2 p$	Δ_L	Δ_R	$\log_2 p$	Δ_L	Δ_R	$\log_2 p$
0	400	1900	-0	0	40	-0	200020	80088	-0	4000000	11000000	-0
1	100	400	-2	40	0	-0	880008	200020	-4	1000000	4000000	-2
2	0	100	-2	100	40	-2	2	880008	-6	0	1000000	-2
3	100	0	-0	440	100	-2	880000	2	-2	1000000	0	-0
4	400	100	-2	1000	440	-4	200000	880000	-4	4000000	1000000	-2
5	1100	400	-2	4440	1000	-2	80000	200000	-2	11000000	4000000	-2
6	4200	1100	-4	101	4440	-6	0	80000	-2	60000000	11000000	-4
7	1D01	4200	-4	4044	101	-4	80000	0	-0	51000001	60000000	-4
8	500	1D01	-8	10	4044	-6	200000	80000	-2	4000004	51000001	-8
9	100	500	-3	4004	10	-2	880000	200000	-2	41000011	4000004	-4
10	100	100	-2	1	4004	-4	2	880000	-4	0	41000011	-8
11	500	100	-2	4000	1	-2	880008	2	-2	41000011	0	-0
12	1500	500	-3	0	4000	-2	200020	880008	-6	4000004	41000011	-8
13				4000	0	-0	80088	200020	-4	51000001	4000004	-4
14							200	80088	-6	60000000	51000001	-8
15							80888	200	-2	11000000	60000000	-4
16										4000000	11000000	-4
17										1000000	4000000	-2
18										0	1000000	-2
19										1000000	0	-0
20										4000000	1000000	-2
21										11000000	4000000	-2
$\sum_r \log_2 p_r$			-34			-36			-48			-72
$\log_2 p_{\text{diff}}$			-34.00			-29.69			-42.11			-60.40
#trails			1			45083			112573			450536
$\log_2 p_{\text{thres}}$			-4.05			-4.05			-4.05			-4.05
pDDT			128			128			128			128
Time:			36 min			47 min			132 min			$>$ 778 min.

In Fig. 5 is shown an example of a complete bipartite subgraph (biclique) similar to the ones composing the digraph in Fig. 9. Note that each node has the same left input difference Δ_L due to the Feistel structure of Simon.

Consider the pair of left and right input differences $(\Delta_L^i, \Delta_R^i) = (11, 106)$ (hexadecimal values). Through the non-linear component $f(x) = (x \lll 1) \wedge (x \lll 8)$ of the round function, the difference $\Delta_L^i = 11$ propagates to a set of output differences. This set has the form $\nabla =$ 000* 000* 00*0 00*0, where $*$ can take values 0/1. Note that for some assignments of the $*$ bits, the resulting difference may have zero probability as was explained in Sect. 2.2, Fig. 2.

Table 6. Differential trails for SPECK32, SPECK48 and SPECK64.

	SPECK32			SPECK48			SPECK64		
r	Δ_L	Δ_R	$\log_2 p$	Δ_L	Δ_R	$\log_2 p$	Δ_L	Δ_R	$\log_2 p$
0	8054	A900	-0	202040	82921	-0	9	1000000	-0
1	0	A402	-3	480901	94009	-7	8000000	0	-2
2	A402	3408	-3	80802	42084A	-7	80000	80000	-1
3	50C0	80E0	-8	400052	504200	-7	80800	480800	-2
4	181	203	-4	820200	1202	-5	480008	2084008	-4
5	C	800	-5	9000	10	-4	6080808	164A0848	-7
6	2000	0	-3	80	0	-2	F2400040	40104200	-13
7	40	40	-1	800000	800000	-0	820200	1202	-8
8	8040	8140	-1	808000	808004	-1	9000	10	-4
9	40	542	-2	800084	8400A0	-3	80	0	-2
10				80A0	2085A4	-4	80000000	80000000	-0
11				808424	84A905	-7	80800000	80800004	-1
12							80008004	84008020	-3
13							808080A0	A08481A4	-5
14							40024	4200D01	-8
$\sum_r \log_2 p_r$			-30			-47			-60
$\log_2 p_{\text{diff}}$			-30.00			-46.48			-59.02
#trails			1			384			934
max HW			7			7			7
$\log_2 p_{\text{thres}}$			-5.00			-5.00			-5.00
pDDT			2^{30}			2^{30}			2^{30}
Time:			$\approx$240 min			$\approx$260 min			$>$ 207 min.

For $\nabla = \{0122, 0102, 0120\}$ three distinct output differences Δ_L^{i+1} from one round of SIMON are produced. They are shown as the three lower level nodes in Fig. 5 and are obtained as $\nabla \oplus ((\Delta_L^i \lll 2) \oplus \Delta_R^i) = \nabla \oplus (44) \oplus \Delta_R^i$.

Another node with the same input difference Δ_L^i to the round function, but with different right difference Δ_R^i e.g. $(\Delta_L^i, \Delta_R^i) = (11, 104)$ (see Fig. 5) produces a corresponding set of output differences ∇', which may or may not have common elements with ∇ in general. For example, in this case $\nabla' = \{0100, 0120, 0122\}$ produced by the node $(11, 104)$. In either case though, ∇ and ∇' may still produce the same set of output differences $(\Delta_L^{i+1}, \Delta_R^{i+1})$. When this happens then a biclique is formed. This is shown in Fig. 5 where both ∇ and ∇' result in the same set of output differences $(\Delta_L^{i+1}, \Delta_R^{i+1}) \in \{(4, 11), (26, 11), (6, 11)\}$.

In general, when the sets ∇, ∇' produced from two different pairs of input differences have high (and possibly equal) probabilities, the complete subgraphs that are formed as a result, have thick edges (corresponding to high probability). Such subgraphs contribute to the clustering of differential trails in SIMON.

Note that the described subgraphs may not be formed for all possible elements in ∇ of an arbitrary node since, as already mentioned, some of them may propagate with 0 probability through the non-linear component f. Furthermore, because the complete bipartite subgraphs depend on the input differences, they can not occur at arbitrary positions in the digraph (Fig. 9). The frequent

occurrence of such special subgraph structures in SIMON in large numbers is the main cause for the strong differential effect observed experimentally using the tool for differential search.

7 Key Recovery Attack on SIMON32

In this section we describe a key recovery attack on SIMON32 with 64 bit key. The input difference to Round-r is denoted as Δ^{r-1} and, bit positions $i_1, i_2, .., i_t$ of x as $x[i_1, i_2, .., i_t]$. Also K^r denotes the round key for the Round-$(r+1)$ and $\mathcal{E}_r$ denotes the encryption function used with r rounds.

7.1 Attack on 19 Rounds

To attack 19 rounds of SIMON32 we add 2 rounds on top and 4 rounds at the bottom of a set of four 13 round differentials. For this attack consider the following 13 round differentials

$$\mathcal{D}_1 : (\texttt{2000}, \texttt{8000}) \rightarrow (\texttt{2000}, \texttt{0})$$
$$\mathcal{D}_2 : (\texttt{4000}, \texttt{0001}) \rightarrow (\texttt{4000}, \texttt{0})$$
$$\mathcal{D}_3 : (\texttt{0004}, \texttt{0010}) \rightarrow (\texttt{0004}, \texttt{0})$$
$$\mathcal{D}_4 : (\texttt{0008}, \texttt{0020}) \rightarrow (\texttt{0008}, \texttt{0})$$

each having probability $\approx 2^{-28.5}$. The truncated difference at the beginning of Round-0, for the above mentioned differentials look as following:

$$(\texttt{00*0 0000 1*00 0000, **00 001* *0*0 0000})$$
$$(\texttt{0*00 0001 *000 0000, *000 01** 0*00 000*})$$
$$(\texttt{0001 *000 0000 0*00, 01** 0*00 000* *000})$$
$$(\texttt{001* 0000 0000 *000, 1**0 *000 00** 0000})$$

Observing the active and inactive bit positions of the above truncated differentials we can construct a set 2^{25} plaintexts where each $\mathcal{P} = (P_L, P_R) \in \mathcal{P}$ has 9 bits, e.g. $P_L[0, 1, 4, 5, 9, 10, 15], P_R[1, 2]$ fixed to an arbitrary value. We can identify 2^{25} pairs of plaintexts (for each differential) from $\mathcal{P}$ so that the pairs satisfy the corresponding (Δ_L^2, Δ_R^2) after two rounds of encryption. For this we need to guess the following round-key bits – $(\mathcal{D}_1)K^0[8, 6]$, $(\mathcal{D}_2)K^0[9, 7], (\mathcal{D}_3)K^0[13, 11]$, $(\mathcal{D}_4)K^0[14, 12]$. Hence with 4 key guesses, 4 sets of 2^{23} pairs of plaintexts corresponding to a differential $\mathcal{D}_i$ can be identified (where each pair in a set follows the top 2-round differential obtained from $\mathcal{D}_i$). Note that by varying some fixed bits of plaintexts in $\mathcal{P}$ we can identify $2^{30.5}$ pairs for each differential and for each (2 bits) key guess.

Each set of identified $2^{30.5}$ pairs of plaintexts is filtered by verifying the fixed bits of the corresponding truncated difference Δ^{18}. This reduces the number of pairs to $2^{30.5-18} = 2^{12.5}$ for each differential. In order to partially decrypt

each pair of ciphertext it is necessary to guess the following key bits (and linear combinations of key bits) from last 3 rounds.

$$\mathcal{D}_1^K = \{K^{18}, K^{17}[3, 5-8, 12, 14], K^{16}[6] \oplus K^{17}[4], K^{16}[4] \oplus K^{17}[2]\} \quad (9)$$

$$\mathcal{D}_2^K = \{K^{18}, K^{17}[4, 6-9, 13, 15], K^{16}[7] \oplus K^{17}[5], K^{16}[5] \oplus K^{17}[3]\} \quad (10)$$

$$\mathcal{D}_3^K = \{K^{18}, K^{17}[8, 10-13, 1, 3], K^{16}[11] \oplus K^{17}[9], K^{16}[9] \oplus K^{17}[7]\} \quad (11)$$

$$\mathcal{D}_4^K = \{K^{18}, K^{17}[9, 11-14, 2, 4], K^{16}[12] \oplus K^{17}[10], K^{16}[10] \oplus K^{17}[8]\} \quad (12)$$

In each differential we need to guess 25 bits (and linear combination of bits) from last 3 round-keys. So, for any differential $\mathcal{D}_i$ it is necessary to guess: $25 + 2$ (from K^0) $= 27$ bits. For key recovery attack let us first consider the two differentials $\mathcal{D}_1$ and $\mathcal{D}_2$. Note that there are 19 bits common between $\mathcal{D}_1^K$ and $\mathcal{D}_2^K$. For detecting the correct key we maintain an array of counters of size 2^{27} for each $\mathcal{D}_1$ and $\mathcal{D}_2$. A counter is incremented when it is correctly verified using a partially decrypted pair of plaintexts by comparing with corresponding Δ^{15}. For each differential $\mathcal{D}_1$ and $\mathcal{D}_2$, we expect to have $(2^{27} \times 2^{12.5})/2^{14} = 2^{25.5}$ increments. We expect approximately 4 correct pairs for each differential and the probability of a counter being incremented is $1/2^2$. So, it is expected to have $(\frac{1}{4})^4 \times 2^{25.5} = 2^{17.5}$ counters with 4 increments for each case. Let these two sets of counters be T_1 and T_2. Since $\mathcal{D}_1^K$ and $\mathcal{D}_2^K$ has 19 common key bits, after combining T_1 and T_2 we expect to obtain $2^{17.5} \times (2^{17.5}/2^{19}) = 2^{16}$ candidates for $19 + 6 + 6 + 4 = 35$ bits. Let us denote this set of counters as T' (Fig. 6).

Next we partially decrypt 2^{12} pairs of ciphertexts corresponding to $\mathcal{D}_3$ to verify the difference Δ^{15} for each 27 bit key guess. As described previously, we maintain an array of 2^{27} counters. A counter is incremented when it is verified correctly by a pair of ciphertexts. The expected number of counters having value 4 is $2^{17.5}$. Let us denote this set of counters as T_3. $\mathcal{D}_3^K$ and $\mathcal{D}_1^K \cup \mathcal{D}_2^K$ has 20 common round-key bits. Hence, combining T_3 and T' we expect to get $2^{16} \times (2^{17.5}/2^{20}) = 2^{13.5}$ candidates for $35 + (25 + 2 - 20) = 42$ round-key bits (out of which 36 bits correspond to last 3 round keys).

Using the fourth differential $\mathcal{D}_4$ in a similar way we obtain 2^9 candidates for $42 + (25 - 22) + 2 = 47$ bits of round-keys, from which we can determine 39 bits of last 3 round-keys.

In order to recover the key we should know all the last 4 round-keys. For the remaining $64 - 39 = 25$ bits of last four round-keys we use exhaustive search. Hence the total number of key guesses is $2^{9+25} = 2^{34}$ (Fig. 7).

Attack Complexity. The time complexity for encrypting plaintexts is $2^{31.5}$. In the key guessing phase 2^{12} filtered pairs are decrypted for last 4 rounds for each 2^{25} key guesses. This is done for each differential. The partial decryption of ciphertext pairs (and increment of the counters) can be done in steps with partial key guess at each step of the last four rounds. This is done by filtering (due to the fixed bits of the truncated differences) at the beginning of Round-16 to Round-18. The complexity for this process is given as:

$$4 \cdot 4 \cdot (2^{12.5} \cdot 2^{16} + 2^{12.5} \cdot 2^9 \cdot 2^7 + 2^{12.5} \cdot 2^2 \cdot 2^2) \cdot \frac{1}{19} \approx 2^{33} \quad (13)$$

For identifying the $2^{30.5}$ pairs with 4 key guesses for each differential requires $2^{31.5} \cdot 2^2 = 2^{33.5}$ two round encryptions. The complexity due to this part is $2^{33.5} \times 4 \times (2/19) \approx 2^{32}$. Hence the total complexity of the attack is $\approx 2^{34}$.

We also show attacks on round-reduced SIMON48 and SIMON64. The details of these attacks are described in Appendixes (C and D).

8 Key Recovery Attack on SPECK32

In this section we describe a chosen plaintext (CP) attack on 11 rounds of SPECK32 using the same notations as in Sect. 7. To attack SPECK32 we use the 9 round differential trail with probability 2^{-30} given in Table 6. We add one round (Round-1) at the top of the trail and one round at the bottom (Round-11)

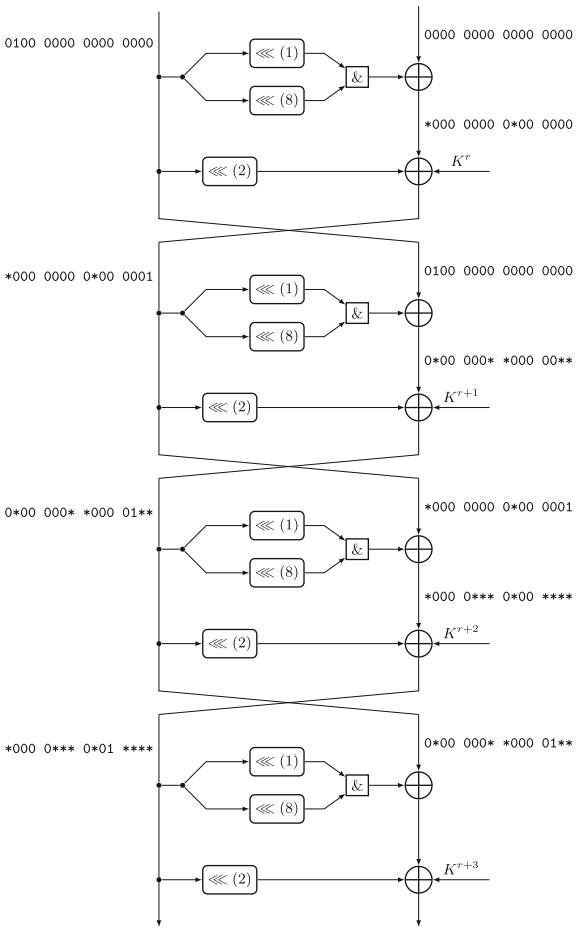


Fig. 6. Truncated difference (in binary notation) in the last 4 rounds of the 18 and 19 round key-recovery attacks on SIMON32.

of the trail to cover 11 rounds in total. If we encrypt 2^{30} pairs of plaintexts such that (Δ_L^1 = 8054, Δ_R^1 = A900), then it is expected to produce $2^{30} \times \frac{1}{2^{30}} = 1$ pair of plaintext satisfying the input/output differences at Round-2 and Round-10 and, $2^{30} \times \frac{1}{2^{28}} = 4$ pairs of plaintexts satisfying the input/output differences at Round-2 and Round-9 (Fig. 8).

The key recovery attack is performed according to the following steps:

1. *Filtering:* The least significant 7 bits of difference after the modular addition at Round-10 are always 100 0000. This implies that Δ^{10} should be of the form **** **** *100 0000, where * denotes unknown bit values. Hence 2^{30} pairs of plaintext/ciphertexts can be filtered by unrolling the output difference of ciphertexts and verifying the 7 bits of Δ^{10}. This reduces the number of pairs to $2^{30-7} = 2^{23}$.

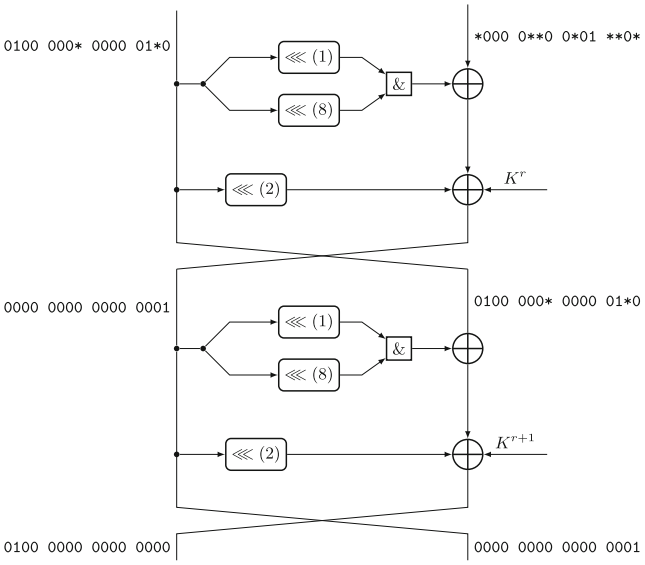


Fig. 7. Top 2 rounds in the attack of SIMON32

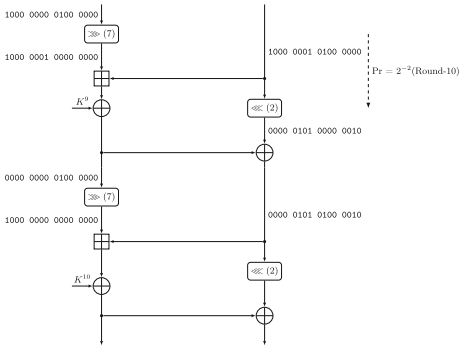


Fig. 8. Differential trail for Round-10 and Round-11 in SPECK32

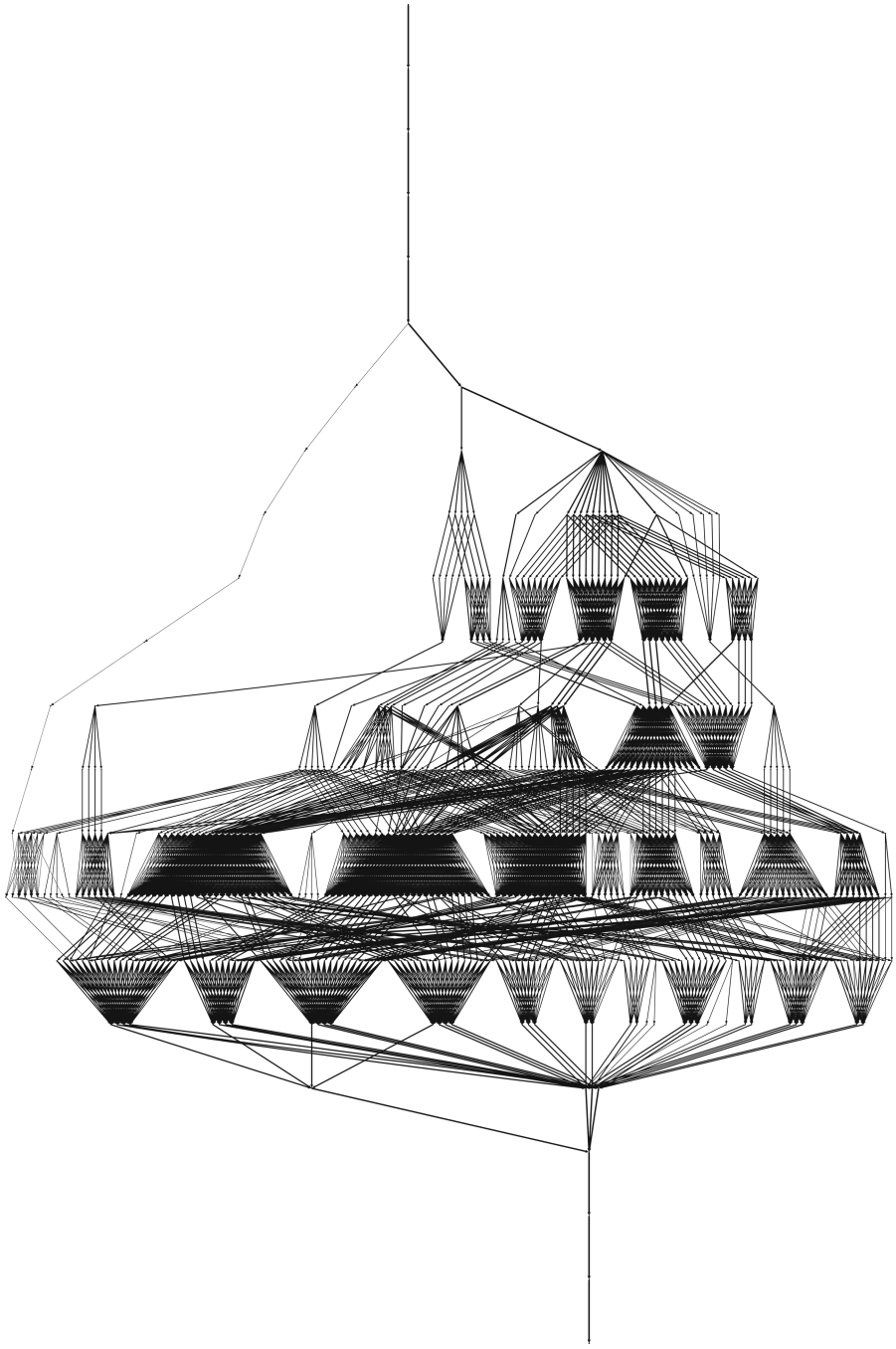


Fig. 9. Clustering of multiple trails satisfying the 21 round differential $(\mathtt{4000000}, \mathtt{11000000}) \rightarrow (\mathtt{11000000}, \mathtt{4000000})$ in SIMON64. The thickness of the edges between two nodes is proportional to the number of right pairs that follow the differential. The graph depicts more than 275 000 differential trails in total.

2. *Partial Key Guessing:* For the filtered pairs, we guess all 16 bits of K^{10} and 11 bits of K^9(e.g. $K^9[5-15]$) and, one carry bit at bit position 5 (in the modular addition at Round-10). For each of these 2^{28} partial key (and carry bit) guess we keep a counter. A counter is incremented if after partially decrypting (last 2 rounds) a pair of ciphertexts satisfies the difference $(\Delta_L^9, \Delta_R^9) = (8040, 8140)$. This will result in $(2^{28} \times 2^{23})/2^{25} = 2^{26}$ increments of all the counters. Probability of a counter getting incremented is $2^{26}/2^{28} = \frac{1}{2^2}$ and, 4 pairs are expected to satisfy the condition at the end of Round-9. Hence, number of counters incremented by 4 are $2^{26} \times (\frac{1}{2^2})^4 = 2^{18}$.
3. *Exhaustive Search:* For the remaining $64 - 27 = 37$ bits from the last rounds keys K^9, K^8, K^7 we use exhaustive search.

Attack Complexity. The complexity for decrypting of 2^{23} ciphertext pairs for each 2^{28} guesses of key bits and carry bit is, $(2^{28} \cdot 2^{23}) \cdot \frac{1}{11} \approx 2^{47}$. The total number of key guesses is $2^{18} \cdot 2^{37} = 2^{55}$. Hence, total complexity is dominated by $\approx 2^{55}$.

With the same attack strategy, we also attack SPECK48 and SPECK64. The details of those attacks are described in Appendixes (A and B).

9 Conclusion

In this paper were presented new results on the differential analysis of lightweight block ciphers SIMON and SPECK. In particular, by applying new techniques for the automatic search of trails and differentials in ARX ciphers, several previous results were improved. Those improvements were further used to mount the currently best known attacks on several versions of SIMON and SPECK. In addition an efficient algorithm for the computation of the DP of the AND operation was presented. A detailed analysis of the strong differential effect in SIMON was given and the reason for it was analyzed. The described methods are general and are therefore applicable to other ARX designs.

Acknowledgments. We thank our colleagues from the laboratory of algorithmics, cryptology and security (LACS) at the university of Luxembourg for the useful discussions. We are especially thankful to Yann Le Corre for pointing out several inconsistencies in the experimental results on SIMON32. We further thank the anonymous reviewers for their time and useful comments. Some of the experiments presented in this paper were carried out using the HPC facility of the University of Luxembourg.

References

1. Abed, F., List, E., Lucks, S., Wenzel, J.: Cryptanalysis of the speck family of block ciphers. Cryptology ePrint Archive, Report 2013/568 (2013). http://eprint.iacr.org/
2. Abed, F., List, E., Lucks, S., Wenzel, J.: Differential and linear cryptanalysis of reduced-round simon. Cryptology ePrint Archive, Report 2013/526 (2013). http://eprint.iacr.org/

3. Alkhzaimi, H.A., Lauridsen, M.M.: Cryptanalysis of the SIMON family of block ciphers. Cryptology ePrint Archive, Report 2013/543 (2013). http://eprint.iacr.org/
4. Beaulieu, R., Shors, D., Smith, J., Treatman-Clark, S., Weeks, B., Wingers, L.: The SIMON and SPECK families of lightweight block ciphers. Cryptology ePrint Archive, Report 2013/404 (2013). http://eprint.iacr.org/
5. Biryukov, A., Velichkov, V.: Automatic search for differential trails in ARX ciphers. In: Benaloh, J. (ed.) CT-RSA 2014. LNCS, vol. 8366, pp. 227–250. Springer, Heidelberg (2014)
6. Bogdanov, A.A., Knudsen, L.R., Leander, G., Paar, C., Poschmann, A., Robshaw, M., Seurin, Y., Vikkelsoe, C.: PRESENT: an ultra-lightweight block cipher. In: Paillier, P., Verbauwhede, I. (eds.) CHES 2007. LNCS, vol. 4727, pp. 450–466. Springer, Heidelberg (2007)
7. De Cannière, C., Dunkelman, O., Knežević, M.: KATAN and KTANTAN — a family of small and efficient hardware-oriented block ciphers. In: Clavier, C., Gaj, K. (eds.) CHES 2009. LNCS, vol. 5747, pp. 272–288. Springer, Heidelberg (2009)
8. Ferguson, N., Lucks, S., Schneier, B., Whiting, D., Bellare, M., Kohno, T., Callas, J., Walker, J.: The Skein Hash Function Family. Submission to the NIST SHA-3 Competition (Round 2) (2009)
9. Gong, Z., Nikova, S., Law, Y.W.: KLEIN: a new family of lightweight block ciphers. In: Juels, A., Paar, C. (eds.) RFIDSec 2011. LNCS, vol. 7055, pp. 1–18. Springer, Heidelberg (2012)
10. Guo, J., Peyrin, T., Poschmann, A., Robshaw, M.: The led block cipher. In: Preneel and Takagi [16], pp. 326–341
11. Hong, D., Sung, J., Hong, S.H., Lim, J.-I., Lee, S.-J., Koo, B.-S., Lee, C.-H., Chang, D., Lee, J., Jeong, K., Kim, H., Kim, J.-S., Chee, S.: HIGHT: a new block cipher suitable for low-resource device. In: Goubin, L., Matsui, M. (eds.) CHES 2006. LNCS, vol. 4249, pp. 46–59. Springer, Heidelberg (2006)
12. Internet Resource. Internet of Things. Wikipedia (2013). http://www.en.wikipedia.org/wiki/Internet_of_Things. Accessed July 2013
13. Khovratovich, D., Nikolić, I.: Rotational cryptanalysis of ARX. In: Hong, S., Iwata, T. (eds.) FSE 2010. LNCS, vol. 6147, pp. 333–346. Springer, Heidelberg (2010)
14. Matsui, M.: On correlation between the order of s-boxes and the strength of DES. In: De Santis, A. (ed.) EUROCRYPT 1994. LNCS, vol. 950, pp. 366–375. Springer, Heidelberg (1995)
15. Mouha, N., Velichkov, V., De Cannière, C., Preneel, B.: The differential analysis of s-functions. In: Biryukov, A., Gong, G., Stinson, D.R. (eds.) SAC 2010. LNCS, vol. 6544, pp. 36–56. Springer, Heidelberg (2011)
16. Preneel, B., Takagi, T. (eds.): CHES 2011. LNCS, vol. 6917. Springer, Heidelberg (2011)
17. Shibutani, K., Isobe, T., Hiwatari, H., Mitsuda, A., Akishita, T., Shirai, T.: Piccolo: an ultra-lightweight blockcipher. In: Preneel and Takagi [16], pp. 342–357
18. Shirai, T., Shibutani, K., Akishita, T., Moriai, S., Iwata, T.: The 128-bit blockcipher CLEFIA (extended abstract). In: Biryukov, A. (ed.) FSE 2007. LNCS, vol. 4593, pp. 181–195. Springer, Heidelberg (2007)
19. Suzaki, T., Minematsu, K., Morioka, S., Kobayashi, E.: *TWINE*: a lightweight block cipher for multiple platforms. In: Knudsen, L.R., Wu, H. (eds.) SAC 2012. LNCS, vol. 7707, pp. 339–354. Springer, Heidelberg (2013)
20. Weinmann, R.-P.: AXR - Crypto Made from Modular Additions. XORs and Word Rotations. Dagstuhl Seminar09031, January 2009

Appendix

A Attack on SPECK48

To attack SPECK48, with key size 72, we use 10 round differential in table with probability $2^{-39.46}$. By adding one round at the top and one round at the bottom we can attack 12 rounds of the cipher. We encrypt 2^{42} pairs of plaintext so that they satisfy the input difference $(\texttt{202040}, \texttt{82921})$ at the beginning of Round-2. The expected number of good pairs at the end of Round-11 is ≈ 4. By partially decrypt the last round, 2^{42} pairs of plaintext is filtered to yield $2^{42-25} = 2^{17}$ pairs of plaintexts. Next, we guess 24 bits of the last round key and decrypt the last round. If we keep a counter for each key guess, which are incremented when a key is suggested as correct candidate by a pair of cipher texts, then the total number of increment is $(2^{24} \times 2^{17})/2^{23} = 2^{18}$. Since, the probability of a counter having an increment is $2^{18}/2^{24} = \frac{1}{2^6}$, we expect exactly one counter having 4 increments, which corresponds to the correct key. By guessing the next two round key successively in the same way we can recover the whole key. The complexity due to the key guessing phase of attack is: $2^{18} \times 2^{24} \times \frac{1}{12} + 2^{18} \times 2^{24} \times \frac{2}{12} + 2^{18} \times 2^{24} \times \frac{3}{12} = 2^{41}$. Hence, the total complexity is $2^{41} + 2^{43} \approx 2^{43}$.

In order to attack SPECK48 with key size 96 bit, we should recover the last four round keys. The attack can be performed in the similar way as described above. However, we need to proceed one more round from the bottom during the key guessing phase. The complexity for this phase is: $2^{18} \times 2^{24} \times \frac{1}{12} + 2^{18} \times 2^{24} \times \frac{2}{12} + 2^{18} \times 2^{24} \times \frac{3}{12} + 2^{18} \times 2^{24} \times \frac{4}{12} \approx 2^{42}$. Hence, the overall complexity remains the same as before.

B Attack on SPECK64

To attack SPECK64 with key size 96, we use the 14 round differential trail obtained by threshold search method. To attack 16 rounds of the cipher we add one round at the top and one round at the bottom. 2^{62} pairs of plaintexts are chosen so that they satisfy the input difference $(\texttt{9}, \texttt{1000000})$ to Round-2. Since probability of 14 round differential is 2^{-60}, it is expected to have approximately 4 good pairs at the end of Round-15. Note that the difference at the end of Round-15 is $(\Delta_L, \Delta_R) = (\texttt{40024}, \texttt{4200d01})$. So, the difference after modular addition at Round-16 should have the least significant bit active. Using this together with Δ_R^{15} (input difference on the right to round-16) we filter out some pairs of cipher texts (obtained from the 2^{62} chosen plaintext pairs) by using the partial difference at the end of Round-15. $2^{62-33} = 2^{29}$ pairs are left after filtering. We guess the round key K^{15} by keeping an array of counter for each guess and, a counter is incremented when it is verified with the fixed difference after partial decryption of pair of cipher texts. Total number of increments is given as $(2^{32} \times 2^{29})/2^{31} = 2^{30}$. Since probability of a counter being incremented once is 2^2, we expect $\frac{1}{(2^2)^4} \times 2^{30} = 2^{22}$ counters having 4 increments. Each of these 2^{22} counter corresponds to a candidate for round key K^{15}. In order to

recover the key we need to know the round keys K^{14} and K^{13}. So, we proceed by guessing K^{14}. But, in this phase we maintain an array of 2^{54} counters, each for $2^{22+32} = 2^{54}$ possible key candidates for K^{15}, K^{14}. Expected number of increments of the counters is: $(2^{22} \times 2^{29} \times 2^{32})/2^{32} = 2^{51}$. The number of correct pair after decrypting the cipher-text pairs for last two rounds is expected to be $2^2 \cdot 2^8 = 2^{10}$. Hence, after this phase we will get one key candidate for K^{15}, K^{14} (total 64 bits). Recall that for recovering the key we need to know the last 3 round-keys. For K^{13} we use exhaustive search which requires 2^{32} key guessings.

Attack Complexity. The complexity of the attack is dominated by the key guessing phase. The complexity during this process is given as

$$2^{32} \times 2^{30} \times \frac{1}{16} + 2^{22} \times 2^{30} \times 2^{32} \times \frac{1.5}{16} \approx 2^{80}$$

Hence the attack complexity is dominated by 2^{80}.

The attack on Speck64 with 128 bit key can be performed in a similar way as described above. However, in this case we need to know the last four round keys. Hence, we perform the partial decryption of ciphertext pairs by guessing a round-key for one more round. The complexity for the key guessing and verifying with fixed difference after partial decryption is

$$(2^{32} \times 2^{30} \times \frac{1}{16} + 2^{22} \times 2^{30} \times 2^{32} \times \frac{1.5}{16} + 2^{32} \times 2^{22} \times \frac{3}{16}) \approx 2^{80}$$

Hence in this case also the complexity of the attack is $\approx 2^{80}$.

Note that for both 128 bit and 96 bit key sizes the complexity can be reduced to $\approx 2^{73}$ if we use the full code book i.e. 2^{64} data.

C Attack on Simon48

C.1 Attacking 19 Rounds

We use the 15 round differential in Table 5 to attack 19 rounds of Simon48, with key size 72. We add one round on top of this differential and 3 rounds at the end. Since, the probability of the differential is $2^{-42.11}$, using 2^{45} pairs of chosen plaintexts we expect to get approximately 8 good pairs at end of Round-16. Propagating the difference $(\texttt{80888}, \texttt{200})$ through Round-17 and Round-18 we get a truncated difference $\Delta_L^{17} = (\texttt{001* *000 *01* *00* 001* *000})$ and $\Delta' = (\texttt{**** *00* 0*** *0*0 1*** *000})$, where Δ' is the output difference before XORing of round-key and $\Delta_L^{17} \lll 2$. Using this truncated difference and unrolling the differences from Round-19 we can filter the plaintext/ciphertext pairs. This leaves $2^{45-25} = 2^{20}$ pairs of plaintexts.

By guessing 29 round-key bits $K^{18}[1-6, 8-23], K^{17}[0, 4, 10, 12, 14, 18, 20]$ and, a linear combination $K^{17}[2] \oplus K^{18}[0]$ we can partially decrypt a pair of

ciphertexts to verify $\Delta^{16} = (80888, 200)$ This is done by maintaining an array of 2^{30} counters. A counter is incremented when it is suggested by a plaintext pair. Hence, the total number of increment is $(2^{30} \times 2^{20})/2^{23} = 2^{27}$, implying that the number of counters incremented by 8 is expected to be $(\frac{1}{2^3})^8 \times 2^{27} = 2^3$. For the remaining $(72 - 30) = 42$ bits of the last three round-keys we use exhaustive search. Hence the total number of key guesses is $2^{42} \cdot 2^3 = 2^{45}$.

Attack Complexity. The complexity for encrypting plaintexts is 2^{46}. During the guessing process of 30 bits of key(and linear combination) complexity is given as

$$(2^{21} \times 2^{22} + 2^{21} \times 2^7 \times 2^7) \times \frac{1}{19} \approx 2^{39}$$

Hence the complexity is $\approx 2^{46}$.

For attacking SIMON48 with key size 96 bits, we use the same technique described above. However in this case we need to know last 4 round-keys. So, after the first stage of the key guessing, we will be left with $96 - 30 = 66$ bits to guess. For this we use exhaustive search and, the total number of key guesses is $2^{66+3} = 2^{69}$.

C.2 Attacking 20 Rounds

To attack 20 rounds of the cipher we use the same technique used for SIMON32. The idea is to add 2 rounds on top of the 15 round differential instead of 1 round. If the difference to Round-3 $\Delta^3 = (200020, 80088)$, is propagated through Round-2 and Round-1, the we get

$$\Delta_L^0 = \texttt{000* 0000 *000 *01* 000* *000}$$
$$\Delta_R^0 = \texttt{***0 *0** 00** ***0 1*** 1000}$$

We construct a set of plaintexts $\mathcal{P}$ of size 2^{20} by varying the bit positions in $P = (P_L, P_R)$ corresponding to $*$ in all possible ways and keeping other bit positions fixed to an arbitrary value. Using this set $\mathcal{P}$ and guessing 6 bits of the round key K^0 we can identify 2^{20} pairs which satisfy the input difference to Round-3. Now by varying the fixed bit positions(2^{25} times) we can identify 2^{45} pairs of plaintexts satisfying the input difference to Round-3. Next we perform the attack as described before (on 19 rounds) for each key guess of K^0. The complexity of the attack will be roughly $2^6 \times 2^{46} = 2^{52}$ and $2^6 \times 2^{69} = 2^{75}$ for key size 72 and 96 respectively.

D Attack on SIMON64

D.1 Attack on 26 Rounds

To attack SIMON64, with key size 96, we use the 21 round differential with probability $2^{-60.53}$ (see Table 2). We add one round on the top and 4 rounds at the

bottom of the differential. Since the probability of the differential is $2^{-60.53}$, using 2^{62} pairs of chosen plaintexts we expect to get approximately $2-3$ good pairs at end of Round-22. Propagating the difference $(\Delta_L^{22}, \Delta_R^{22}) = (\texttt{11000000}, \texttt{4000000})$ through Round-23, Round-24 and Round-25 we get a truncated difference with 35 known bits, input to Round-26:

$$\Delta_R^{25} = (\texttt{**01 **01 0000 0000 000* 000* 0**0 0**1})$$
$$\Delta_L^{25} = (\texttt{*0** *1*0 000* 000* 0**0 0*** **0* ****})$$

Using this truncated difference and unrolling the differences from Round-26 we can filter the plaintext/ciphertext pairs. This leaves $2^{62-35} = 2^{27}$ pairs of plaintexts. By guessing 49 round-key bits

$$\text{Round} - 26 : K^{25}[0-15, 17-31]$$
$$\text{Round} - 25 : K^{24}[0, 2-5, 7, 9, 11, 13, 16, 20, 22, 23, 25, 27, 29, 30]$$
$$\text{Round} - 24 : K^{23}[31]$$

and linear combinations of round-key bits $K^{23}[3] \oplus K^{24}[1]$, $K^{23}[17] \oplus K^{24}[15]$, $K^{23}[21] \oplus K^{24}[19]$ we can partially decrypt a pair of ciphertexts to verify $(\Delta_L^{22}, \Delta_R^{22}) = (\texttt{11000000}, \texttt{4000000})$. This is done by maintaining an array of 2^{52} counters. A counter is incremented when it is suggested by a plaintext pair. Hence, the total number of increments is $(2^{52} \times 2^{27})/2^{29} = 2^{50}$, implying that the number of counters incremented by 2 is expected to be $(2^{50}/2^{52})^2 = (\frac{1}{2^2})^2 \times 2^{49} = 2^{45}$. For the remaining 44 bits of the last three round-keys we use exhaustive search. Hence the total number of key guesses is $2^{45} \cdot 2^{44} = 2^{89}$.

Attack Complexity. For partial decryption part of the attack the complexity is

$$(2^{28} \times 2^{31} + 2^{28} \times 2^{16} \times 2^{17} + 2^{28} \times 2^{16} \times 2^7 \times 2^4) \times \frac{1}{26} \approx 2^{59}.$$

Hence the complexity is dominated by the total number of key guessing part, which is $\approx 2^{89}$.

For attacking SIMON64 with key size 128 bits, we use the same technique described above. However in this case we need to know the last 4 round-keys. So, after the first stage of the key guessing, we will be left with $128 - 52 = 76$ bits to guess. For this we use exhaustive search and, the total number of key guesses is $2^{45+76} = 2^{121}$.

Equivalent Key Recovery Attacks Against HMAC and NMAC with Whirlpool Reduced to 7 Rounds

Jian Guo[1(✉)], Yu Sasaki[2], Lei Wang[1], Meiqin Wang[3], and Long Wen[3]

[1] Nanyang Technological University, Singapore, Singapore
ntu.guo@gmail.com, wang.lei@ntu.edu.sg
[2] NTT Secure Platform Laboratories, Tokyo, Japan
sasaki.yu@lab.ntt.co.jp
[3] Key Laboratory of Cryptologic Technology and Information Security, Ministry of Education, Shandong University, Jinan 250100, China
mqwang@sdu.edu.cn, longwen@mail.sdu.edu.cn

Abstract. A main contribution of this paper is an improved analysis against HMAC instantiating with reduced Whirlpool. It recovers equivalent keys, which are often denoted as K_{in} and K_{out}, of HMAC with 7-round Whirlpool, while the previous best attack can work only for 6 rounds. Our approach is applying the meet-in-the-middle (MITM) attack on AES to recover MAC keys of Whirlpool. Several techniques are proposed to bypass different attack scenarios between a block cipher and a MAC, *e.g.,* the chosen plaintext model of the MITM attacks on AES cannot be used for HMAC-Whirlpool. Besides, a larger state size and different key schedule designs of Whirlpool leave us a lot of room to study. As a result, equivalent keys of HMAC with 7-round Whirlpool are recovered with a complexity of (Data, Time, Memory) $= (2^{481.7}, 2^{482.3}, 2^{481})$.

Keywords: HMAC · NMAC · Whirlpool · Universal forgery · Key recovery

1 Introduction

A cryptographic hash function is a public algorithm that compresses arbitrary long messages into short and random digests. An important application is a Message Authentication Code (MAC). A MAC is a keyed algorithm that takes a secret key and an arbitrary long message as input, and produces a short random string as the tag. The tag provides the authenticity and the integrity for the original messages. In this paper, we mainly study the security of one dedicated hash-based MAC, HMAC based on the hash function Whirlpool.

Whirlpool was proposed by Barreto and Rijmen in 2000 [1]. Its security was evaluated and approved by NESSIE [2]. Moreover, Whirlpool has been

M. Wang—This work has been partially supported by 973 Program (No. 2013CB834205), NSFC Project (No. 61133013), Program for New Century Excellent Talents in University of China (NCET-13-0350), as well as Interdisciplinary Research Foundation of Shandong University (No. 2012JC018).

C. Cid and C. Rechberger (Eds.): FSE 2014, LNCS 8540, pp. 571–590, 2015.
DOI: 10.1007/978-3-662-46706-0_29

internationally standardized by ISO/IEC, and practically implemented in various cryptographic software libraries such as Crypto++ [3]. Many cryptanalysis results have been published on Whirlpool [4–9]. Particularly, collision attack and preimage attacks on Whirlpool hash function reach 5 and 6 rounds out of 10 rounds respectively [5,9]. Moreover, a distinguisher on full-round Whirlpool compression function was found in [5]. Although it is an interesting and beautiful attack, the security impact of such a distinguisher seems limited. Thus Whirlpool still stands secure after receiving more than 10 years consecutive analysis from worldwide cryptanalysts.

The HMAC scheme was designed by Bellare *et al.* in 1996 [10], and becomes the most well-known hash-based MAC scheme. HMAC has been standardized by many international standardization organizations including ANSI, IETF, ISO and NIST. Also it has been widely deployed in various practical protocols including SSL, TLS and IPSec. Cryptanalysts have been continuously evaluating the security of both HMAC and HMAC based on dedicated hash functions. Generic attacks on HMAC include [11–14]. The attacks on HMAC with popular dedicated hash functions can be found in [15–21].

Due to the important roles of Whirlpool and HMAC in current cryptography as briefly described above, the security evaluation of HMAC-Whirlpool is important and interesting. Very recently in October 2013, ENISA (The European Union Agency for Network and Information Security) published a report for recommending cryptographic algorithms [22]. In particular, the report recommends (for future applications in industry) three dedicated hash functions with Whirlpool included, and two hash-based MAC with HMAC included. So it can be expected that HMAC-Whirlpool is going to have more applications in industry in the coming years, Thus HMAC-Whirlpool should receive a careful security evaluation from the cryptographic community in advance.

The first cryptanalysis of HMAC-Whirlpool was published by Guo *et al.* [23], which is also the only algorithmic security evaluation on HMAC-Whirlpool so far to our best knowledge. They proposed key recovery attacks on HMAC with Whirlpool reduced to 5 and 6 rounds out of 10 rounds.

Our Contributions. This paper presents improved analysis on HMAC-Whirlpool. HMAC, from the original key K, derives two keys K_{in} and K_{out} which are usually referred to as *equivalent* keys. If both K_{in} and K_{out} are recovered, an adversary can perform the universal forgery attack. In this paper, we present an equivalent key recovery attack on HMAC with 7-round Whirlpool, which extends the number of attacked rounds by one round compared with previous work [23].

The design of Whirlpool is based on the AES block cipher. Our idea is applying the recent meet-in-the-middle (MITM) attack on 7-round AES [24,25] to recover MAC keys of 7-round Whirlpool. The analysis seems quite simple at a short glance, however, such an extension is not trivial at all due to differences of attack scenarios between a block cipher and a MAC. For example, MITM attacks on AES work under the chosen plaintext model, while the input message for the outer function of HMAC is a hash digest of the inner function, which cannot be chosen by the attacker. The output of AES block cipher, *i*.e. ciphertext, can

Table 1. Summarization of key-recovery results on `HMAC-Whirlpool`

Key type	#Rounds	Complexity			Reference
		Time	Memory	Data	
Original key	5	2^{402}	2^{384}	2^{384}	[23]
	6	2^{496}	2^{448}	2^{384}	[23]
Equivalent keys	5	2^{448}	2^{377}	2^{321}	[23]
	6	2^{451}	2^{448}	2^{384}	[23]
	7	$\mathbf{2^{482.3}}$	$\mathbf{2^{481}}$	$\mathbf{2^{481.7}}$	**Ours**

be observed by the attacker, while the output of the intermediate compression function in `HMAC` cannot be observed. Besides, a larger state size and different key schedule designs of `Whirlpool` leave us a lot of room to study.

A summary of our results and previous *key recovery* attacks is given in Table 1. Our attack can also be applied to `NMAC-Whirlpool`. It is interesting to recall that the current best collision and preimage attacks on `Whirlpool` hash function reach only 6 rounds. Such a phenomenon is not common particularly for key recovery attacks. For example, key recovery attack on `HMAC-SHA-1` reaches only 34 rounds out of 80 rounds [18], while collision attack on `SHA-1` hash function reaches full rounds [26] and preimage attack reaches 57 rounds [27].

Throughout this paper, we target `HMAC-Whirlpool` that uses a 512-bit key and produces full size, *i.e.,* 512-bit, tags. Targeting this case has theoretical interests since `HMAC` is defined to use a key of any bit size. Moreover, `HMAC` instantiating with a key size of one block of an underlying hash function (512 bits for `Whirlpool`) and with full size tags is utilized in cryptographic protocols. One example is `HMAC`-based Extract-and-Expand Key Derivation Function [28].

Besides `HMAC`, we briefly discuss other `MAC`s. For `Prefix-MAC` with 7-round `Whirlpool`, we can also recover the equivalent key. On the other hand, for `LPMAC` with 7-round `Whirlpool`, we cannot recover the equivalent key. Nevertheless, we modify the attack procedure and manage to launch universal forgery attack.

Organization of the Rest Paper. Section 2 describes previous related works. Section 3 presents an overview of our attack on `HMAC` with 7-round `Whirlpool`. Section 4 describes the details of our attacks and shows the application to other `MAC`. Finally we conclude the paper in Sect. 5.

2 Related Work

2.1 `Whirlpool` Hash Function

`Whirlpool` [1] takes any message with less than 2^{256} bits as input, and outputs a 512-bit hash value. It adopts the Merkle-Damgård structure. The input message M is padded into a multiple of 512 bits. The 256-bit binary expression of the bit length ℓ is padded according to the MD-strengthening, *i.e.* $M\|1\|0^*\|\ell$.

The padded message is divided into 512-bit blocks $M_0 \| M_1 \| \cdots \| M_{N-1}$. Let H_n be a 512-bit chaining variable. First, an initial value IV is assigned to H_0. Then, $H_{n+1} \leftarrow \mathcal{CF}(H_n, M_n)$ is computed for $n = 0, 1, \ldots, N-1$, where $\mathcal{CF}$ is a compression function. H_N is produced as the hash value of M.

The compression function $\mathcal{CF}$ consists of an AES-based block-cipher E_k with the Miyaguchi-Preneel mode, which takes a 512-bit chaining variable H_i as a key and a 512-bit message block M_i as a plaintext. The output of $\mathcal{CF}$ is computed by $H_{n+1} \leftarrow E_{H_i}(M_i) \oplus M_i \oplus H_i$. Inside the block cipher E_k, an internal state is represented by an $8 * 8$ byte array. At first, H_i is assigned to the key value k_{-1}, and M_i is assigned to the plaintext s_{-1}. Then, the whitening operation with the master key k_{-1} is performed and the result is stored into a variable s_0, *i.e.* $s_0 \leftarrow k_{-1} \oplus s_{-1}$. The cipher generates ten 512-bit subkeys $k_0, k_1, \ldots, k_9$ from k_{-1} by the key schedule function, and updates s_0 through ten rounds with generated subkeys. The computation of the block cipher output s_{10} is as follows:

$$\text{Key Schedule: } k_n \leftarrow \mathrm{AC} \circ \mathrm{MR} \circ \mathrm{SC} \circ \mathrm{SB}(k_{n-1}), \text{ for } n = 0, 1, \ldots, 9,$$
$$\text{Data Processing: } s_n \leftarrow \mathrm{AK} \circ \mathrm{MR} \circ \mathrm{SC} \circ \mathrm{SB}(s_{n-1}), \text{ for } n = 0, 1, \ldots, 9,$$

where the details of each operation are as follows.

- SubBytes (SB): apply the AES S-Box to each byte.
- ShiftColumns (SC): cyclically shift the j-th column downwards by j bytes.
- MixRows (MR): multiply each row of the state matrix by an MDS matrix.
- AddRoundConstant (AC): XOR a 512-bit pre-specified constant.
- AddRoundKey (AK): XOR a 512-bit subkey k_n.

We sometimes swap the order of MR and AC for the key schedule and MR and AK for the data processing. In this case, the AK operation XORs $\mathrm{MR}^{-1}(k_n)$. Hereafter, we denote $\mathrm{MR}^{-1}(k_n)$ by u_n.

Notations. The byte position in the i-th row and the j-th column of state S is denoted by two-dimensional integers $S[i][j]$, where $0 \leq i, j \leq 7$. We denote the initial state for round n by x_n. Internal states immediately after SB, SC and MR in round n are denoted by y_n, z_n and w_n, respectively. We often denote several byte positions by using comma, *e.g.,* 8 bytes in the top row of state S are denoted by $S[0][0, 1, \ldots, 7]$. We also use the following notations:

- $S[row(i)]$: 8 byte-positions in the i-th row of state S,
- $S[\mathrm{SC}(row(i))]$: 8 byte-positions which SC is applied to $S[row(i)]$,
- $S[\mathrm{SC}^{-1}(row(i))]$: 8 byte-positions which SC^{-1} is applied to $S[row(i)]$.

We use $\mathcal{H}$ to denote a hash function, and $\mathcal{CF}(ch, M)$ to denote a compression function. For the ease of notation, M may be of multiple blocks, then $\mathcal{CF}$ acts the same as hash function $\mathcal{H}$ except no padding.

2.2 Hash Based MACs

`HMAC` and `NMAC`. `HMAC` and `NMAC` [29] are hash-based MACs proposed by Bellare *et al.* [10,30]. `NMAC` requires two keys K_{in} and K_{out} while `HMAC` requires only a

single key K, and generates two equivalent keys by processing $K \oplus \texttt{ipad}$ and $K \oplus \texttt{opad}$, where opad and ipad are two public constants. Let $\mathcal{H}$ be a hash function. Also, let $\mathcal{H}(\mathrm{IV}, \cdot)$ represent that the initial value of $\mathcal{H}$ is IV. On an input message M, NMAC and HMAC computes the tag value as

$$\texttt{NMAC-}\mathcal{H}_{K_{in},K_{out}}(M) = \mathcal{H}\big(K_{out}, \mathcal{H}(K_{in}, M)\big),$$
$$\texttt{HMAC-}\mathcal{H}_{K}(M) = \mathcal{H}(K \oplus \texttt{opad} \| \mathcal{H}(K \oplus \texttt{ipad} \| M)).$$

Prefix-MAC and LPMAC. Prefix-MAC is a classical MAC construction [31], which computes a tag of message M by $\mathcal{H}(K\|M)$. It is known to be vulnerable against the length extension attack, *i.e.,* from a given pair of message and tag (M, t), the attacker can forge the tag for $M\|x$ for any x. However, no generic attack is known in terms of the key recovery. We suppose that the prefix K is processed independently of M, *i.e.,* the tag is computed by $\mathcal{H}(K\|\texttt{pad}\|M)$ in which pad is a padding string and the size of $K\|\texttt{pad}$ is a multiple of the block size. LPMAC is a strengthened version of Prefix-MAC [31] so that the length-extension attack is prevented. Let ℓ be the input message size. A tag is computed by $\mathcal{H}(K\|\ell\|\texttt{pad}\|M)$, where the size of $K\|\ell\|\texttt{pad}$ is a multiple of the block size.

2.3 Generic Internal State Recovery Attack on HMAC

Leurent *et al.* [14] provide a generic inner state recovery attack against HMAC/NMAC with time complexity $2^{3n/4}$. The result is that, $h_{in} = \mathcal{CF}(IV, K\|M_p^{2^{n/4}})$ can be recovered, with $M_p^{2^{n/4}}$ satisfying the padding rule, and of about $2^{n/4}$ blocks. We can then recover inner state $\mathcal{CF}(IV, K\|M_p^1)$, with M_p^1 of one block. This can be done through detecting colliding tags, *i.e.,* we randomly choose $2^{n/2}$ messages x, x' independently so that the two messages $M_p^{2^{n/4}}\|x$ and x' follow the padding rules, and query their tags, denoted as t and t', respectively. When t and t' collide, the chance that they collide at inner hash, *i.e.,* $h'_{in} = \mathcal{CF}(IV, K\|x') = \mathcal{CF}(IV, K\|M_p^{2^{n/4}}\|x) = \mathcal{CF}(h_{in}, x)$ is roughly 1/3.

2.4 6-Round Key Recovery Attack on HMAC-Whirlpool

The first cryptanalysis of HMAC-Whirlpool was published by Guo *et al.* [23], which showed a key recovery attack on HMAC reduced to 6 rounds. They first apply the generic internal state recovery in Sect. 2.3, and then find a message pair satisfying a particular differential characteristic. The fact that the pair satisfies the characteristic reduces a possible differential patterns of internal states. This allows an attacker to exhaustively guess internal state values and differences, and the correct guess is identified by the MITM attack. On one hand, the MITM attack in [23] is a classic type which divides the computation into two independent sub-functions, *e.g.,* [32–35]. On the other hand, the MITM attacks on AES later explained in Sect. 2.5 are based on a different framework,

2.5 Meet-In-The-Middle Attack on AES

The unified view of a series of MITM attacks on AES [24,25,36,37] is well-summarized in [24]. The concept of *δ-set* takes an important role of the attack.

Definition 1 (δ-set [38]). *Let a δ-set be a set of 256 states that are all different in one state bytes (the active byte) and all equal in the other state bytes (the inactive bytes).*

The number of active bytes in the δ-set is often increased, *e.g.,* [39]. It is easy to extend the concept of the δ-set to deal with multi-active bytes.

Definition 2 *(n-δ-set).* *Let an n-δ-set be a set of $(256)^n$ states that are all different in n state bytes and all equal in the other state bytes.*

The MITM attack divides the cipher into three parts:

$$s_0 \longrightarrow (s_{n_1} \longrightarrow s_{n_2}) \longrightarrow s_{\text{last}},$$

so that the middle part can satisfy a certain property, which is later verified with the partial encryption for the first part and the partial decryption for the last part. The general attack consists of the following five successive steps:

Precomputation Phase

1. A lookup table T is built which contains all the possible sequences constructed from a δ-set such that one message verifies the property for the middle part.

Online Phase

2. Through the oracle query, candidates of the plaintext-ciphertext pair that satisfies the target property for the middle part are searched.
3. For each candidate pair, subkeys for the first part that achieves the property for the middle part are guessed, and then the internal state value at the beginning of the middle part, s_{n_1}, is modified so that a δ-set containing a state value verifying the desired property is constructed.
4. With the guessed subkeys for the first part, the δ-set at s_{n_1} is decrypted to obtain the corresponding plaintexts. Those plaintexts are queried to the encryption oracle, and the corresponding ciphertexts are obtained.
5. Finally, subkeys for the last part are guessed, and the obtained ciphertexts associated by the δ-set at s_{n_1} are partially decrypted through the last part and tested whether it belongs to T.

If the analyzed pair is the right one and the guessed subkeys are right ones, the result of Step 5 belongs to T with probability 1, and the key is recovered. Because of Step 4, the attack is a chosen plaintext attack.

Previous work consider a function $f : \{0,1\}^8 \to \{0,1\}^8$ that maps the active byte value of a δ-set to another byte of the state after four rounds, s_{n_2}. Gilbert and Minier [37] found that an ordered sequence $(f(0), \ldots, f(255))$ for AES four rounds are parameterized only by 25 bytes, which takes significantly smaller

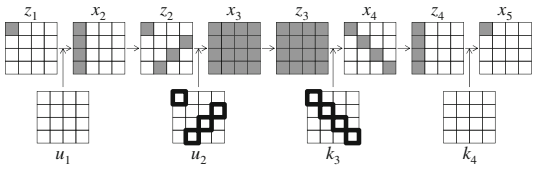


Fig. 1. 4-round differential characteristic for MITM attacks on `AES` [25]. Grey bytes are active. Subkey bytes represented by bold square are the ones used as parameters.

space, *i.e.,* $2^{25 \cdot 8} = 2^{200}$, than the theoretically possible space, $2^{8 \cdot 2^8} = 2^{2048}$. Considering the difference $(f(0)-f(0), f(1)-f(0), \ldots, f(255)-f(0))$, the attack is improved so that the function is parameterized only by 24 bytes. Also note that the effect of the filtering with T is strong enough even with a fraction of $(f(0), \ldots, f(255))$. On average, storing $(f(0), \ldots, f(31))$ is enough to make the key space sufficiently small. Such optimization was discussed in [40].

Dunkelman *et al.* introduced the four-round truncated differential characteristic in the middle part [25], which is shown in Fig. 1. The characteristic is parameterized only by 16 bytes, *i.e.,* the states x_3 and z_3 can only take 2^{32} differences each so that the number of solutions for these two states is 2^{64}. Then, at most 4 bytes in u_2 and 4 bytes in k_3 can affect the characteristic. Hence, the number of paired state values (z_1, z_1') satisfying the characteristic is at most 2^{128}. For each of such (z_1, z_1'), the attacker constructs the δ-set at x_1 and can compute the 1-byte difference at x_5. They also introduced the concept of the multiset rather than the ordered sequence. This enables the attacker to avoid guessing 1 subkey byte at the online phase, *i.e.,* the partial decryption at the online phase becomes from x_1 to plaintext instead of from z_1 to plaintext, which avoids guessing 1 subkey byte to bypass the SB operation between x_1 and z_1. Because the theoretically possible numbers of multisets with 256 elements is $2^{467.6}$, the 2^{128} possible patterns for `AES` is significantly small, which is enough to filter out all the noise.

The latest attack by Derbez *et al.* [24] is an improvement of the attack in [25]. They found that the four-round characteristic in Fig. 1 is parameterized only by 10 bytes, *i.e.,* the number of solutions for the characteristic is at most 2^{80}. They also consider the multi-active bytes at z_1 and multi-differential characteristics for the middle four-round characteristic so that the active byte positions of z_1 and x_5 can take any of $\binom{4}{2}$ patterns and $\binom{4}{1}$ patterns respectively.

3 Overview of Our Attacks

This section gives a high level overview of our attacks on `HMAC` with 7-round `Whirlpool`. The `HMAC` computation structure in our attack is shown in the upper half of Fig. 2. Our goal is to recover the two equivalent keys K_{in} and K_{out}.

In the mode-of-operation level, we follow the approach of the previous work [23]. Namely, with the generic attack [14], we first find a single-block message M_p

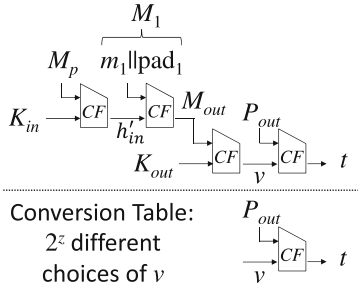


Fig. 2. Overall strategy

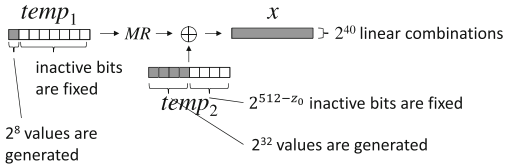


Fig. 3. Optimized choice for table inputs

whose compression function output h'_{in} is recovered. The knowledge of h'_{in}, for any message of the form $M_p \| x$, allows the attacker to compute the input value for the outer hash function, M_{out}. We then recover the output value of the first compression function in the outer hash function, v, from the observed tag value, t. By iterating this procedure, the attacker collects many pairs of (M_{out}, v). We then recover $E_{K_{out}}$ by using our compression function analysis approach which will be explained in the next paragraph. Once $E_{K_{out}}$ is recovered, K_{in} can be recovered by the same analysis as K_{out}.

In the compression function level, to recover K_{out} or K_{in}, we plan to extend the recent 7-round MITM attack on `AES` [24] to attack `HMAC-Whirlpool`. More precisely, the first message block in the inner and the outer hash functions are computed as $E_{K_{in}}(M) \oplus M \oplus K_{in}$ and $E_{K_{out}}(M) \oplus M \oplus K_{out}$ respectively, where $E_{K_{in}}$ and $E_{K_{out}}$ are two `AES`-like block ciphers. The target values K_{in} and K_{out} are used as the key for the `AES`-like block ciphers. Therefore, by regarding the input value of the inner/outer hash functions as the plaintext and by regarding the output value as the ciphertext, K_{in} and K_{out} should be recovered by applying the MITM attack on `AES`. However, immediately we find that such an extension is not trivial at all because of the following differences in the attack scenarios.

- The control ability of the attacker on choosing plaintexts is different. Particularly for the outer hash function in `HMAC-Whirlpool`, the input message is the inner hash digest, and thus cannot be chosen by the attacker.
- The knowledge of the ciphertext is different. In `HMAC Whirlpool`, the output is the intermediate hash values, which are confidential to the attacker.
- The state size is different. `Whirlpool` has a larger state size then `AES`. This yields both advantages and disadvantages for the attack.
- The key schedule function is different. Several research, in the context of hash function, show that the similar diffusions between the key schedule and the data processing of `Whirlpool` is easier to analyze than `AES` [5,9,23].

3.1 On Recovering Ciphertexts: Generating Conversion Table

Recall Fig. 2. To recover K_{out}, the corresponding value of v is necessary. However, due to the additional padding block P_{out}, the attacker cannot observe v.

We solve this problem by generating a conversion table denoted by T_c. Note that P_{out} is a fixed value because the input message length to the outer hash function is always the same (512 bits). We then precompute $t \leftarrow \mathcal{CF}(v, P_{out})$ for many choices of v, and store pairs of (v, t) as a look-up table. Later at online phase, v can be recovered by matching the observed t and the elements in T_c.

Suppose 2^z pairs of (v, t) are generated to construct T_c. Then, for any M_{out}, we can recover the v from t with probability about 2^{z-512}.

3.2 On Choosing Plaintexts

M_{out} can be computed but cannot be chosen by the attacker. Therefore, the chosen plaintext attack cannot be used to recover K_{out} by analyzing $\mathcal{CF}(K_{out}, M_{out})$. This is crucial to apply the previous MITM attack on `AES` [24].

We solve this problem by converting the attack into the known plaintext attack. In the previous procedure in Sect. 2.5, queries are made in Step 2 and Step 4. Regarding Step 2, the previous work used the structure, while we generate more plaintexts at random so that the difference is satisfied probabilistically.

Converting Step 4 to the known plaintext attack is much harder. The previous attack on `AES` uses 256 plaintexts for the δ-set, and the corresponding multiset is queried. Without the chosen plaintext model, we cannot guarantee that all 256 plaintexts for the δ-set are known. To solve this problem, we use an n-δ-set with a relatively big n instead of a δ-set. Because n is big, we can obtain sufficient information even only with a fraction of the 2^{8n} plaintexts.

3.3 On Large State Size

A large state size is easier to analyze than a small state size. For example, guessing one row of a subkey requires only 1/8 of the entire key space for `Whirlpool` while it is 1/4 for `AES`. Then, we have more choices of attack parameters *e.g.,* the number of active bytes in the n-δ-set, the number of active rows in the input and output, *etc.*. As a side-effect, identifying the best parameters becomes harder. We optimize the attack with exhaustively trying all parameters by programming.

The theoretical number of multisets for 256 elements is $2^{467.6}$, which is unlikely to occur on `AES`-128 where the attack complexity is below 2^{128}. However, the key space of `HMAC-Whirlpool` is 2^{512}, hence we sometimes cannot filter out all the noise. This problem can be solved by the weak key schedule of `Whirlpool`.

3.4 On Key Schedule

Recall the attack on `AES` in Fig. 1. For `AES`, 4-byte values in u_2 and k_3 do not reveal any other subkey byte in u_1 and k_4, respectively. However, for `Whirlpool`, 8 bytes for each inverse diagonal in $u_2[\mathrm{SR}(row(i))]$ and 8 bytes for each diagonal in $k_3[\mathrm{SR}^{-1}(row(i))]$ reveal 8 bytes of $u_1[row(i)]$ and 8 bytes of $k_4[row(i)]$, respectively. This means, we do not need previous smart ideas of [25,36], *i.e.,* when we generate a look-up table for the middle 4-round characteristic for each

parameter, we can compute rows of x_5 and rows of x_1. Hence, each element in the look-up table T_δ can be an ordered sequence of values instead of a multiset of differences. This also gives us another advantage that the theoretically possible numbers of ordered sequences is much larger than the multisets. As explained before, $2^{467.6}$ possibilities of multisets are not enough to analyze `Whirlpool` with a 512-bit key. By using the ordered sequence, this problem can be avoided.

3.5 Overview of the Attack

We detect that using the 4-round differential characteristic $12 \rightarrow 24 \rightarrow 64 \rightarrow 8 \rightarrow 1$ for the middle 4 rounds optimizes the attack on `Whirlpool`. The entire differential characteristic is constructed by extending this middle 4-round characteristic by 1 round in backwards and 2 rounds in forwards. The number of solutions to satisfy the characteristic is 2^{370}. We then construct a 12-δ-set for each of all possible 2^{370} pairs, and store them as a look-up table T_δ.

Precomputation Phase

0. A conversion table T_c, containing 2^z pairs of (v, t) is generated. Hereafter, for any M_{out}, we can recover the v from t with probability 2^{z-512}.
1. A lookup table T_δ is built which contains all possible 2^{370} pairs satisfying the middle 4-round characteristic. For each of 2^{370} pairs, a 12-δ-set is constructed at s_{n_1}, and an ordered sequence of 2^{96} values $f(0), \ldots, f(2^{96}-1)$ that map 12 bytes values at s_{n-1} to 1 byte value at s_{n-2} is stored.

Online Phase

2. 2^Q random messages of the form $M_p \| M_1$ are queried, and only the ones whose t belongs to T_c are picked. The number of expected (M_{out}, v) is $2^{Q+(z-512)}$. To find candidate pairs satisfying the middle 4-round differential characteristic, we make about $2^{2(Q+(z-512))-1}$ pairs of (M_{out}, v), and only pick the ones satisfying the input and output differential forms.
3. For each candidate pair, subkeys for the first part and also for the last part that satisfy the middle 4-round characteristic are exhaustively guessed.
4. The internal state value at the beginning of the middle part, s_{n_1}, is modified so that a 12-δ-set is constructed. With the guessed subkeys for the first part, the 12-δ-set at s_{n_1} is decrypted to the plaintext M_{out}. If M_{out} belongs to the ones generated at Step 2, the corresponding v is recovered and (M_{out}, v) is stored as data associated with a fraction of the 12-δ-set.
5. Finally, with the guessed subkeys for the last part, each of the obtained v is partially decrypted and tested whether it belongs to T_δ. Note that the obtained (p, v) at online is the data associated with only a fraction of the 12-δ-set. Which of $f(0), \ldots, f(2^{96}-1)$ is used for the match cannot be fixed, and thus we cannot properly sort the elements in T_δ so that the match can be done only with 1 computation. We later solve this problem in Sect. 4.

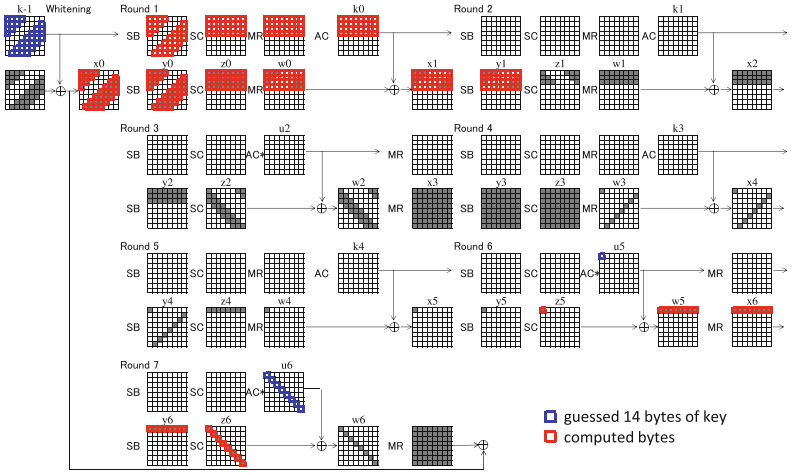


Fig. 4. Differential characteristic used in our 7-round attack.

4 Recovering K_{in} and K_{out} of HMAC/NMAC-Whirlpool

We first apply the generic internal state recovery attack by Leurent *et al.* [14] to find a single-block message M_p and its compression function output h'_{in}.

As described in Sect. 3.1, in order to invert the last compression function call of the outer layer and recover the output of the attacked compression function, we build a conversion table of size 2^{512-z}, for some $z \geq 0$ to be decided later. It is important to note that, the attacker is not able to choose the M_{out}, the output of the inner layer, and M_{out} plays the role similar to "plaintext" for block ciphers in our attack. Hence, our attack setting for the key recovery is similar to the "known-plaintext" attack for block ciphers. Also, due to collisions of the compression function, the chance that a lookup gives the right v is of probability $1-1/e$. At the moment, the input values v of the conversion table are randomly chosen, later we show how the choices of v can be used to optimize the overall attack.

4.1 The 4-Round Differential Characteristic

Our attack follows the previous MITM attacks on AES, which relies on a 4-round differential characteristic, from state y_1 of round 2 to x_5 of round 6 of the Whirlpool compression function, as depicted in Fig. 4. The number of active bytes, in gray color, follows $12 \rightarrow 24 \rightarrow 64 \rightarrow 8 \rightarrow 1$. Let us denote the set of bytes at positions $[0][0,1,2], [1][0,1,7], [2][0,6,7], [3][5,6,7]$ as B_{in}, and byte at position $[0][0]$ as B_{out}, then the input/output differences of the middle 4-round characteristic can be simply denoted as $y_1[B_{in}]$ and $x_5[B_{out}]$. With this characteristic, one round and two rounds are added before and after it, to form the 7-round Whirlpool as the attack target.

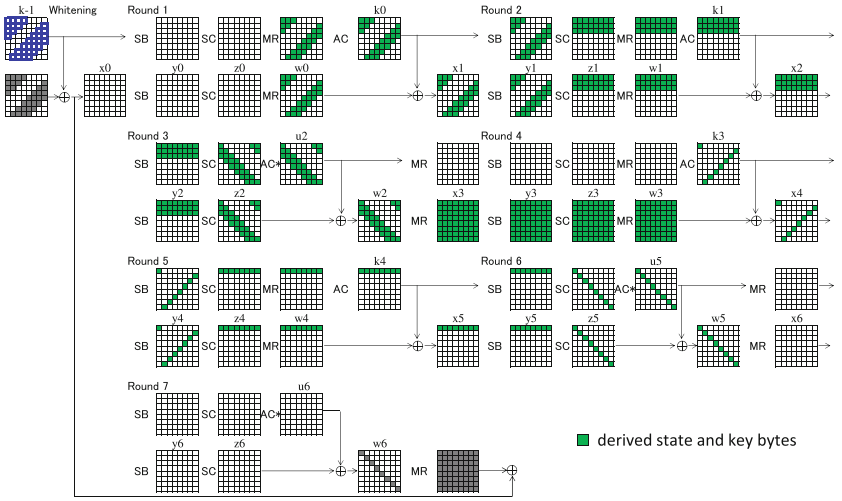


Fig. 5. Derived state and key bytes from the differential characteristic.

4.2 Computing the δ-set Table

Based on the 4-round differential characteristic above, we build the δ-set table, denoted as T_δ. Given a difference $\Delta y_1[B_{in}]$, and a state value of first three rows of x_2, denoted as $x_2[row(0,1,2)]$, both value and difference in active bytes of z_2 can be derived, followed by difference in active bytes of w_2 and x_3. Similarly, given a difference $\Delta x_5[B_{out}]$ and state value of $z_4[row(0)]$, $\Delta x_4[\mathrm{SC}^{-1}(row(0))]$ can be obtained, followed by Δy_3. For each pair of $(\Delta x_3, \Delta y_3)$, one solution of state value x_3 and y_3 is expected on average. With actual values in the active bytes of z_2 and w_2, $u_2[\mathrm{SC}(row(0,1,2))]$ can be derived. Due to the diffusion property of the key schedule, we can further derive other key bytes, *i.e.,* $k_1[row(0,1,2)]$. Similarly $k_3[\mathrm{SC}^{-1}(row(0))]$ is derived and we can further derive $k_4[row(0)]$. We denote all the subkey bytes obtained as k_{ob}. Due to the newly recovered k_{ob}, we can further compute more bytes in the data processing part. Finally, both value and difference in all active bytes of the 4-round characteristic can be determined. All the recovered state and key bytes can be found in Fig. 5.

These state information at every round of the differential characteristic allows to derive the value of $x_5[B_{out}]$ from any $\Delta' y_1[B_{in}]$, even if $\Delta' y_1[B_{in}]$ does not follow the characteristic. We briefly illustrate it here. Given $\Delta' y_1[B_{in}]$ and $x_2[row(0,1,2)]$, both difference and actual values of $x_2[row(0,1,2)]$ can be obtained, followed by difference in active bytes of y_2, z_2, w_2 and x_3. Together with actual value of x_3, we derive difference in y_3, z_3, w_3 and x_4. Note, for an arbitrary given $\Delta' y_1[B_{in}]$, the difference in w_3 and x_4 may not follow the characteristic any more. However, since we only care about the difference in those active bytes in gray, *i.e.,* $\Delta x_4[\mathrm{SC}^{-1}(row(0))]$, those unwanted bytes values can be discarded. Together with actual value of $x_4[\mathrm{SC}^{-1}(row(0))]$, derive the differences in the first row of z_4, w_4 and x_5. Together with actual value of $x_5[B_{out}]$,

Algorithm 1. Construction of the δ-set table

Empty a lookup table T_δ.
for all 12 bytes differences in $\Delta y_1[B_{in}]$ and 24 bytes values of $x_2[row(0,1,2)]$ **do**
  Deduce differences in x_3.
  for all 1 byte differences in $\Delta x_5[B_{out}]$ and 8 bytes values of $z_4[row(0)]$ **do**
    Deduce differences in y_3.
    Use the differential property of SBox to deduce the values in x_3, x_3', y_3, y_3'.
    Deduce k_{ob} and all state values of the active bytes of the characteristic.
    Empty an ordered sequence M.
    for all 12 bytes values in $y_1[B_{in}]$ **do** // construct 12-δ-set at y_1
      Compute the corresponding $x_5[B_{out}]$ and add it to M.
    end for
    Add M to the lookup table T_δ, indexed by ($\Delta y_1[B_{in}]$, $\Delta x_5[B_{out}]$, $x_2[row(0,1,2)]$, $z_4[row(0)]$), M and k_{ob} as entry values.
  end for
end for
Output: T_δ of 2^{456} entries. //2^{360} indices each containing 2^{96} relations of M

one can get the value of $x_5'[B_{out}]$. The details of the δ-set table computation are shown in Algorithm 1.

4.3 The Online Phase

For Step 2 of the procedure in Sect. 3.5, we search for paired message candidates satisfying the middle 4-round characteristic. To ensure at least one pair follows the entire 7-round characteristic as in Fig. 4, we count the number of conditions of the 7-round characteristic, there are 32 bytes conditions in the input state, 20 bytes for $z_0 \rightarrow w_0$, 56 bytes for $z_3 \rightarrow w_3$, 7 bytes for $z_4 \rightarrow w_4$, and the rest is almost for free. In total 115 bytes = 920 bits, hence we need $2^{(920+1)/2} \simeq 2^{461}$ random inputs to generate enough pairs. The overall data complexity will be $2^{461} \times 2^{512-z}$.

With the provided data (p, v) with p as input and v as output of the attacked compression function, we filter the pairs according to the input/output differences, *i.e.,* $\Delta p[\mathrm{SC}^{-1}(row(4,5,6,7))] = 0$, and output difference follows the pattern in w_6 as in Fig. 4, *i.e.,* $\Delta \mathrm{MR}^{-1}(p+v)[\mathrm{SC}(row(1,2,\ldots,7))] = 0$. We sort all the data according to the value of these non-active bytes, and find the right pairs, as done in the first for loop in Algorithm 2. A randomly generated pair satisfies both the input and output differences with a probability of $2^{-256-448} = 2^{-704}$. Hence, $2^{920-704} = 2^{216}$ candidate pairs will remain.

For Step 3 of the online phase as in Sect. 3.5, with any pair (p, v) and (p', v'), we partially encrypt the p, p' by 1 round, and decrypt v, v' by 2 rounds. To do that, we guess the whitening key bytes $k_{-1}[\mathrm{SC}^{-1}(row(0,1,2,3))]$ in a linear space of size $2^{12*8} = 2^{96}$ so that w_0 fulfills the desired difference pattern. Note that the guess can be done diagonal wise independently. For example, one can guess 3 bytes ([0][0], [7][1], [6][2]) of the first diagonal in k_{-1} first, compute both difference and value of these three bytes in x_0, followed by the difference in $w_0[row(0)]$,

backwards compute the other 5 byte differences in the first diagonal of x_0, from which the other 5 byte value in the first diagonal of k_{-1} can be derived. Repeat the same procedure for the other diagonals, hence each key guess can be done in computation 1. Due to the similarity of the round function and key schedule, $k_0[row(0,1,2,3)]$ can be derived from $k_{-1}[\mathrm{SC}^{-1}(row(0,1,2,3))]$, then both state value and difference of $y_1[row(0,1,2,3)]$ can be obtained. Similarly, one can guess $u_6 \oplus u_{-1}[\mathrm{SC}(row(0))]$ and $u_5[0][0]$, and compute the $\Delta x_5[0][0]$. For each given (p, v), (p', v') pair, there will be 2^8 guessed keys $u_6 \oplus u_{-1}[\mathrm{SC}(row(0))]$ on average making it follow the characteristic. Overall, with each data pair, we guessed 14-byte key values. The total number of candidates so far is $2^{216+112} = 2^{328}$ pairs.

For Step 4 as in Sect. 3.5, with each candidate pair, the 12-δ-set is constructed with respect to $y_1[B_{in}]$. With the guessed subkeys $k_{-1}[\mathrm{SC}^{-1}(row(0,1,2,3))]$ and $k_0[row(0,1,2,3)]$, 2^{96} state values in the set are decrypted to the plaintext. Each of 2^{96} plaintexts are tested if the corresponding v is stored in Step 2. Because 2^{461} (p, v) relations are generated, the probability of having the corresponding v is $2^{461-512} = 2^{-51}$. Therefore, $2^{96-51} = 2^{45}$ plaintexts can have the corresponding v and stored as D_f that is associated with a fraction of the 12-δ-set. Note that we do not need to use all these 2^{45} elements in the 12-δ-set. Instead, we will select 2^8 elements from them for the attack steps later.

For Step 5 as in Sect. 3.5, together with the guessed subkeys $u_6 \oplus u_{-1}$ $[\mathrm{SC}(row(0))]$ and $u_5[0][0]$, $x_5[B_{out}]$ is computed from the output values of 2^8 selected texts in D_f. Denote them as $M = \{(x_1^1, x_5^1), \ldots, (x_1^{2^8}, x_5^{2^8})\}$. Finally, we can check if 2^8 elements of $y_1[B_{in}]$ and $x_5[B_{out}]$ match with one of elements in the precomputed look-up table T_δ. However, because T_δ is not sorted with respect to the 2^8 elements, the match cannot be done with 1 computation.

The precomputation table T_δ is ordered by the index $(\Delta y_1, x_2[row(0,1,2)], z_4$ $[row(0)], \Delta x_5)$. For each pair analyzed in the online phase, we have the knowledge of Δy_1 and Δx_5. Moreover, we have the knowledge of $y_1[row(0,1,2,3)]$ and $k_0[row(0,1,2,3)]$. If we switch the computation order of MR and AC in round 2 and let $w_1' = z_1 \oplus u_1$, then we have $x_2 = \mathrm{MR}(w_1')$. From $y_1[row(0,1,2,3)]$ and $k_0[row(0,1,2,3)]$, we can compute 12 bytes of $w_1'[row(0,1,2)]$. Then we exhaustively guess all the remaining 12 bytes of $w_1'[row(0,1,2)]$, which gives us the value of $x_2[row(0,1,2)]$. We also exhaustively guess all the values of $z_4[row(0)]$. Actually we need to guess the values of $z_4[0][1,2,\cdots,7]$ only, since we guessed $u_5[0][0]$ and computed $z_5[0][0]$, $z_4[0][0]$ can be derived from these two values. This gives us in total 2^{152} index of T_δ. Then we look up T_δ to get the corresponding 12-δ-set for each index. After that, we look up x_5^i for x_1^i in the 12-δ-set, and match it to the value of x_1^i in previously computed M for all $1 \le i \le 2^8$. We adopt early aborting technique when we look up the 12-δ-set. Namely if the x_1^i in the 12-δ-set is not equal to x_1^i in M, we immediately abort and will not match for the remaining elements. By using aborting technique, the number of table looking up for matching M to each 12-δ-set can be counted as one on average. Overall, it needs 2^{152} table lookups to match one M to the table T_δ.

Algorithm 2. Online phase: attack on 7-round `Whirlpool` compression function

Input 1: T_δ obtained from Algorithm 1.
Input 2: Conversion table T_c of size 2^z.
Input 3: 2^d (p, t) pairs, with p input message block, and t the corresponding tag.
Look up the table T_c for all t, and obtain $2^{z+d-512}$ (p, v) pairs.
Empty a temporary table T.
for obtained (p, v) **do**
  $index \leftarrow \mathrm{MR}^{-1}(p \oplus v)[\mathrm{SC}(row(1, 2, 3, 4, 5, 6, 7))]$ and $p[\mathrm{SC}^{-1}(row(4, 5, 6, 7))]$.
  Add (p, t) to $T[index]$.
end for
for all collisions in T, *i.e.,* pair (p, v) and (p', v') **do**
  for all $k_{-1}[\mathrm{SC}^{-1}(row(0, 1, 2, 3))]$ s.t. $\Delta w_0[\mathrm{SC}^{-1}(row(4, 5, 6, 7))] = 0$ **do**
    Construct a 12-δ-set D at y_1 by modifying 12 active bytes in $y_1[B_{in}]$.
    Decrypt 2^{96} elements in D to the plaintext p, and check if p is stored in T_c.
    If stored, add (p, v) to D_f, which is associated to a fraction of 12-δ-set.
    for all $(u_6 \oplus u_{-1})[\mathrm{SC}(row(0))]$ s.t. $\Delta z_5[0][1, \ldots, 7] = 0$ **do**
      Decrypt each element in D_f to obtain $w_5[row(0)]$.
      for all values of $u_5[0][0]$ **do**
        Decrypt each element in D_f to obtain $w_5[0][0]$.
        Construct the ordered sequence M for D_f.
        Compute 12 bytes of $w'_1 = z_1 \oplus u_1$: $[0][0, 5, 6, 7]$, $[1][0, 1, 6, 7]$, $[2][0, 1, 2, 7]$.
        for all values of the 12 bytes of w'_1: $[0][1, 2, 3, 4]$, $[1][2, 3, 4, 5]$, $[2][3, 4, 5, 6]$,
         and $z_4[row(0)]$
          compute $x_2[row(0, 1, 2)]$ and construct the index of T_δ:
            $(\Delta y_1[B_{in}]$, $x_2[row(0, 1, 2)]$, $\Delta_5[0][0]$, $z_4[row(0)])$.
          Get 12-δ-set with the index in T_δ.
          Match M to 12-δ-set.
          if $M \in T_\delta$, **then**
            Exhaustively search the right key, **Output**: K if found.
          end if
        end for
      end for
    end for
  end for
end for

The probability of the false positive, *i.e.,* the probability of two random 2^8 ordered byte relations happen to match is $2^{-8 \cdot 2^8}$, which is negligible. Hence, only the right pair and the right subkey guesses can be detected. The details of the online phase attack is shown in Algorithm 2.

4.4 Complexities and Optimization

Optimizing the Conversion Table. We note that the conversion table is of size 2^z, and the inputs were randomly chosen from the overall 2^{512} choices. We found that, by carefully choosing the inputs, *i.e.,* the output v of the attacked

compression function, we are able to reduce the conditions of the 7-round differential characteristic, by forcing v, then the w_6 as in Fig. 4, into a subspace. We know $w_6 = \mathrm{MR}^{-1}(v \oplus p \oplus k_{-1})$. Since our trick is row wise, we will take the first row as an example. We need $\Delta w_6[0][1,2,3,4,5,6,7] = 0$, and we know there is no difference in k_{-1} and $\Delta p[row(0)]$ only takes 2^{32} values when it follows the characteristic. We force $\Delta v[row(0)]$ to be $\mathrm{MR}(\Delta w_6[0][0]) \oplus \Delta p[0][0,1,2,3]$, so that when we do $\mathrm{MR}^{-1}(\Delta p \oplus v)$, the $\Delta w_6[row(0)]$ is forced into a space of 2^{40} v.s. 2^{64}, this increases the chance $\Delta w_6[row(0)]$ to be the right pattern by a factor of 2^{24}. Note this forces the choices of $v[0]$ into a space of 2^{40} out of 2^{64}, we can apply the same trick to other rows, in the meanwhile, we need to ensure at least 2^z candidates of v are left for the conversion table.

By choosing balanced parameters $z = 481$, and $d = 481$, the overall complexity for data, and memory are 2^{481}, and time 2^{481}. Due to the false-positive of the conversion table, the data complexity increases by a factor of $(1-1/e)^{-1} = 2^{0.7}$ to $2^{481.7}$, and time complexity by a factor of $(1-1/e)^{-2} = 2^{1.3}$ to $2^{482.3}$.

Computer Search. There are several parameters, which affect the overall attack complexities, including:

1. r_1, c_1, for number of active rows and columns in state z_1, respectively.
2. r_2, c_2, for number of active rows and columns in state x_5, respectively.
3. size of conversion table, denoted as $c_t = \log_2(T_c)$.

We can derive the attack complexities from these parameters in different phases:

1. The conversion table costs Memory = Time = 2^{c_t}
2. The number of conditions for the entire 7-round characteristic are: $(8-c_1)\times 64$ for the input messages, $c_1 \times (8-r_1) \times 8$ for $z_0 \rightarrow w_0$ transition, $(8-r_2) \times 64$ for $z_3 \rightarrow w_3$ transition, $(8-c_2) \times 8$ for $z_4 \rightarrow w_4$ transition. The trick with the conversion table saves $c_{con} = 512 - c_t - \lceil (512-c_t)/(64-8 \times c_1) \rceil \times c_2 \times 8$ bits conditions. Denote the overall number of conditions in bits as c_{diff}, we then need $2^{c_{diff}}$ pairs to have at least one pair following the differential characteristic, due to the loss in the conversion table, overall the data complexity is Time = Data = $2^{(c_{diff}+1)/2+512-c_t}$.
3. The size of δ-set table T_δ is computed as follows. There are $r_1 \times c_1 \times 8 + r_2 \times c_2 \times 8$ bits in input/output differences, for each of them, we need to guess $r_1 \times 64 + r_2 \times 64$ state bits, and $r_1 \times c_1 \times 8$ bits state value in y_1 for the full δ-set. The overall number of bits is $\log_2(T_\delta) = r_1 \times c_1 \times 16 + r_2 \times c_2 \times 8 + r_1 \times 64 + r_2 \times 64$.
4. At online phase, we filter $(8-c_1) \times 64$ conditions in message, and $(8-c_2) \times 64 - c_{con}$ in ciphertext, and number of pairs left for online phase is $c_{pair} = c_d - (8-c_1) \times 64 - (8-c_2) \times 64 + c_{con}$. For each pair, we guessed $c_{key} = r_1 \times c_1 \times 8 + r_2 \times c_2 \times 8$ number of key bits. For each of the considered combination, we guessed $c_{mat} = r_1 \times 64 - r_1 \times c_1 \times 8 \times 2 + r_2 \times 64$ bits in the matching stage, the overall time complexity is $2^{c_{pair}+c_{key}+c_{mat}}$.

By a bruteforce search over all possible choices of the r_1, c_1, r_2, c_2 and c_t, we find the best parameters, $r_1 = 3, c_1 = 4, r_2 = 1, c_2 = 1, c_t = 481$ as presented already, give the best complexity $2^{481.7}$ and 2^{481} for Data, Memory and $2^{482.3}$ for Time.

4.5 Recovering K_{in} and Application to Prefix MAC

The attack setting for recovering K_{in} is different from K_{out}. The NMAC computation $\mathcal{H}(K_{out}, \mathcal{H}(K_{in}, M))$ can be derived to $\mathcal{CF}(IV, K_{out} \| \mathcal{CF}(K_{in}, M \| P_{in}) \| P_{out})$, with M of full blocks, P_{in} and P_{out} for padding blocks for inner and outer layer, respectively. When M be of a fixed length, the P_{in} and P_{out} are fixed. With knowledge of K_{out}, the computation after processing M contains no secret, let us denote it as g, *i.e.,* the expression is simplified to $g \circ \mathcal{CF}(K_{in}, M)$. To recover K_{in}, the setting is a little bit different from that for recovering K_{out}. Here, we are able to *choose* the message M, the setting is similar to "chosen plaintext" attack to block ciphers. As done in previous attacks against AES, we choose the M in structures, where in each structure, the non-active bytes are fixed to a constant, and all possibilities in active bytes are taken. Refer to Fig. 4, one fixes bytes in $M[\mathrm{SR}^{-1}(row(4,5,6,7))]$ to a constant and the rest of the bytes take all 2^{256} possibilities. In this way, the pairs from the same structure will have difference pattern in message automatically fulfilled. With help of this, the overall complexities of the attack can be reduced to 2^{465} for time, data and memory with 2^7 differential characteristics.

It is interesting to note that the equivalent key for Prefix MAC can be recovered in exactly the same way. For LPMAC, $K'_{eq} = \mathcal{CF}(IV, K \| \ell \| \mathtt{pad})$ can also be recovered in the same way too. However, recovering $K_{eq} = \mathcal{CF}(IV, K)$ requires inverting the compression function $\mathcal{CF}$ since $K'_{eq} = \mathcal{CF}(K_{eq}, \ell \| \mathtt{pad})$, which is currently unknown for Whirlpool reduced to 7 rounds. Hence, in case of LPMAC, we are able to launch universal forgery attack only.

5 Conclusion

Based on very recent advances in generic state recovery attacks on HMAC/NMAC, and meet-in-the-middle attacks against AES, we present equivalent key recovery attacks against HMAC/NMAC with Whirlpool reduced to 7 rounds. We also showed the application to Prefix-MAC, including LPMAC. This improves one round over the existing work on HMAC/NMAC-Whirlpool, and interestingly, the number attacked already exceeds that for collision and preimage attack against the Whirlpool hash function itself. One reason is that, collision attacks with complexity at or beyond the birthday bound $2^{n/2}$ did not attract much attention in history, and these collisions were found to reveal information about the internal state, and also the key material in MAC applications. Due to the full block key size, the security level is expected to be higher than that of collision resistance, these collisions become essentially helpful to carry out our attacks.

Future work. It is interesting to note that length of the message plays an important role in our attacks. The current generic attacks rely on queries of long message, from $2^{n/4}$ to $2^{n/2}$ blocks, or the complexities approach 2^n when the message length tends to 1 block. Some of the hash function such as current NIST standard SHA-256, supports messages with length up to 2^{64} bits, shorter than

$2^{n/4}$ blocks. This simple restriction may stop the attack or bring the attack complexity close to 2^n for many other hash functions as well. The second restriction is the length of the given message to be forged, in settings of forgery attacks. In many protocols using hash function based MAC, the message used can be very short, *e.g.,* one or two blocks. This won't stop our attacks since we focus on attacks against the compression function, which is a key advantage of our attack over the recent generic attacks. Progress in state recovery with low complexity and short messages will remove the dependency of our attack from the need of long message.

Our attacks do not apply to Sandwich MAC [41], *i.e.,* $\mathcal{H}(K\|p\|M\|p'\|K')$ with p, p' paddings, and Envelope MAC [42], *i.e.,* $\mathcal{H}(K\|p\|M\|p'\|K)$. One can still carry out the internal state recovery attack for long message between the two keys, however it is then not possible to convert it to a state recovery attack for short message, since the padding p' will be different. In our attacks, the recovered K_{in} and K_{out} play a role similar to key, while in Sandwich MAC, K' (or the second K in Envelope MAC) appears in message block, and could not be recovered since it plays a role as "plaintext", which is currently not explored in attacks against `AES`-like ciphers. For the same reason, we did not recover the master key of `HMAC`. It will be interesting to see any progress in this direction.

Acknowledgments. We would like to thank the organizers, Meiqin Wang and Hongbo Yu, of ASK 2013 workshop http://www.infosec.sdu.edu.cn/ask2013/ in China, without which the collaboration in this work could not be possible. Jian Guo and Lei Wang were supported by the Singapore National Research Foundation Fellowship 2012 (NRF-NRFF2012-06).

References

1. Rijmen, V., Barreto, P.S.L.M.: The WHIRLPOOL Hashing Function. Submitted to NISSIE, September 2000
2. NESSIE: New European Schemes for Signatures, Integrity, and Encryption. IST-1999-12324. http://cryptonessie.org/
3. Dai, W.: Crypto++ library. http://www.cryptopp.com/
4. Mendel, F., Rechberger, C., Schläffer, M., Thomsen, S.S.: The Rebound Attack: Cryptanalysis of Reduced Whirlpool and `Grøstl`. In: Dunkelman, O. (ed.) FSE 2009. LNCS, vol. 5665, pp. 260–276. Springer, Heidelberg (2009)
5. Lamberger, M., Mendel, F., Rechberger, C., Rijmen, V., Schläffer, M.: Rebound Distinguishers: Results on the Full Whirlpool Compression Function. In: Matsui, M. (ed.) ASIACRYPT 2009. LNCS, vol. 5912, pp. 126–143. Springer, Heidelberg (2009)
6. Lamberger, M., Mendel, F., Rechberger, C., Rijmen, V., Schläffer, M.: The Rebound Attack and Subspace Distinguishers: Application to Whirlpool. J. Cryptol. **28**(2), 257–296 (2009)
7. Sasaki, Y.: Meet-in-the-Middle Preimage Attacks on AES Hashing Modes and an Application to Whirlpool. In: Joux, A. (ed.) FSE 2011. LNCS, vol. 6733, pp. 378–396. Springer, Heidelberg (2011)

8. Wu, S., Feng, D., Wu, W., Guo, J., Dong, L., Zou, J.: (Pseudo) Preimage Attack on Round-Reduced Grøstl Hash Function and Others. In: Canteaut, A. (ed.) FSE 2012. LNCS, vol. 7549, pp. 127–145. Springer, Heidelberg (2012)
9. Sasaki, Y., Wang, L., Wu, S., Wu, W.: Investigating Fundamental Security Requirements on Whirlpool: Improved Preimage and Collision Attacks. In: Wang, X., Sako, K. (eds.) ASIACRYPT 2012. LNCS, vol. 7658, pp. 562–579. Springer, Heidelberg (2012)
10. Bellare, M., Canetti, R., Krawczyk, H.: Keying Hash Functions for Message Authentication. In: Koblitz, N. (ed.) CRYPTO 1996. LNCS, vol. 1109, pp. 1–15. Springer, Heidelberg (1996)
11. Preneel, B., van Oorschot, P.C.: On the Security of Two MAC Algorithms. In: Maurer, U.M. (ed.) EUROCRYPT 1996. LNCS, vol. 1070, pp. 19–32. Springer, Heidelberg (1996)
12. Peyrin, T., Sasaki, Y., Wang, L.: Generic Related-Key Attacks for HMAC. In: Wang, X., Sako, K. (eds.) ASIACRYPT 2012. LNCS, vol. 7658, pp. 580–597. Springer, Heidelberg (2012)
13. Naito, Y., Sasaki, Y., Wang, L., Yasuda, K.: Generic State-Recovery and Forgery Attacks on ChopMD-MAC and on NMAC/HMAC. In: Sakiyama, K., Terada, M. (eds.) IWSEC 2013. LNCS, vol. 8231, pp. 83–98. Springer, Heidelberg (2013)
14. Leurent, G., Peyrin, T., Wang, L.: New Generic Attacks Against Hash-based MACs. In: Sako, K., Sarkar, P. (eds.) ASIACRYPT 2013, Part II. LNCS, vol. 8270, pp. 1–20. Springer, Heidelberg (2013)
15. Contini, S., Yin, Y.L.: Forgery and Partial Key-recovery Attacks on HMAC and NMAC Using Hash Collisions. In: Lai, X., Chen, K. (eds.) ASIACRYPT 2006. LNCS, vol. 4284, pp. 37–53. Springer, Heidelberg (2006)
16. Fouque, P.-A., Leurent, G., Nguyen, P.Q.: Full Key-recovery Attacks on HMAC/NMAC-MD4 and NMAC-MD5. In: Menezes, A. (ed.) CRYPTO 2007. LNCS, vol. 4622, pp. 13–30. Springer, Heidelberg (2007)
17. Rechberger, C., Rijmen, V.: On Authentication with HMAC and Non-random Properties. In: Dietrich, S., Dhamija, R. (eds.) FC 2007 and USEC 2007. LNCS, vol. 4886, pp. 119–133. Springer, Heidelberg (2007)
18. Rechberger, C., Rijmen, V.: New Results on NMAC/HMAC When Instantiated With Popular Hash Functions. J. UCS **14**, 347–376 (2008)
19. Lee, E., Chang, D., Kim, J.-S., Sung, J., Hong, S.H.: Second Preimage Attack on 3-Pass HAVAL and Partial Key-recovery Attacks on HMAC/NMAC-3-Pass HAVAL. In: Nyberg, K. (ed.) FSE 2008. LNCS, vol. 5086, pp. 189–206. Springer, Heidelberg (2008)
20. Yu, H., Wang, X.: Full Key-recovery Attack on the HMAC/NMAC Based on 3 and 4-Pass HAVAL. In: Bao, F., Li, H., Wang, G. (eds.) ISPEC 2009. LNCS, vol. 5451, pp. 285–297. Springer, Heidelberg (2009)
21. Sasaki, Y., Wang, L.: Improved Single-key Distinguisher on HMAC-MD5 and Key Recovery Attacks on Sandwich-MAC-MD5. In: Lange, T., Lauter, K., Lisoněk, P. (eds.) SAC 2013. LNCS, vol. 8282, pp. 493–512. Springer, Heidelberg (2014)
22. European Union Agency for Network and Information Security: Algorithms, key sizes and parameters report, October 2013. http://www.enisa.europa.eu/
23. Guo, J., Sasaki, Y., Wang, L., Wu, S.: Cryptanalysis of `HMAC/NMAC-Whirlpool`. In: Sako, K., Sarkar, P. (eds.) ASIACRYPT 2013, Part II. LNCS, vol. 8270, pp. 21–40. Springer, Heidelberg (2013)
24. Derbez, P., Fouque, P.-A., Jean, J.: Improved Key Recovery Attacks on Reduced-Round `AES` in the Single-Key Setting. In: Johansson, T., Nguyen, P.Q. (eds.) EUROCRYPT 2013. LNCS, vol. 7881, pp. 371–387. Springer, Heidelberg (2013)

25. Dunkelman, O., Keller, N., Shamir, A.: Improved Single-Key Attacks on 8-Round AES-192 and AES-256 [43]. In: Abe, M. (ed.) ASIACRYPT 2010. LNCS, vol. 6477, pp. 158–176. Springer, Heidelberg (2010)
26. 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)
27. Knellwolf, S., Khovratovich, D.: New Preimage Attacks Against Reduced SHA-1. In: Safavi-Naini, R., Canetti, R. (eds.) CRYPTO 2012. LNCS, vol. 7417, pp. 367–383. Springer, Heidelberg (2012)
28. Krawczyk, H.: RFC: HMAC-Based Extract-and-Expand Key Derivation Function (HKDF), May 2010. https://tools.ietf.org/html/rfc5869.txt
29. U.S. Department of Commerce, National Institute of Standards and Technology: The Keyed-hash Message Authentication Code (HMAC) (Federal Information Processing Standards Publication 198), July 2008. http://csrc.nist.gov/publications/fips/fips198-1/FIPS-198-1_final.pdf
30. Bellare, M.: New Proofs for NMAC and HMAC: Security Without Collision-Resistance. In: Dwork, C. (ed.) CRYPTO 2006. LNCS, vol. 4117, pp. 602–619. Springer, Heidelberg (2006)
31. Tsudik, G.: Message Authentication with One-Way Hash Functions. SIGCOMM Comput. Commun. Rev. **22**(5), 29–38 (1992)
32. Aoki, K., Guo, J., Matusiewicz, K., Sasaki, Y., Wang, L.: Preimages for Step-Reduced SHA-2. In: Matsui, M. (ed.) ASIACRYPT 2009. LNCS, vol. 5912, pp. 578–597. Springer, Heidelberg (2009)
33. Guo, J., Ling, S., Rechberger, C., Wang, H.: Advanced Meet-in-the-middle Preimage Attacks: First Results on Full Tiger, and Improved Results on MD4 and SHA-2 [43]. In: Abe, M. (ed.) ASIACRYPT 2010. LNCS, vol. 6477, pp. 56–75. Springer, Heidelberg (2010)
34. Wei, L., Rechberger, C., Guo, J., Wu, H., Wang, H., Ling, S.: Improved Meet-in-the-middle Cryptanalysis of KTANTAN (Poster). In: Parampalli, U., Hawkes, P. (eds.) ACISP 2011. LNCS, vol. 6812, pp. 433–438. Springer, Heidelberg (2011)
35. Chaum, D., Evertse, J.-H.: Cryptanalysis of DES with a Reduced Number of Rounds. In: Williams, H.C. (ed.) CRYPTO 1985. LNCS, vol. 218, pp. 192–211. Springer, Heidelberg (1986)
36. Demirci, H., Selçuk, A.A.: A Meet-in-the-Middle Attack on 8-Round AES. In: Nyberg, K. (ed.) FSE 2008. LNCS, vol. 5086, pp. 116–126. Springer, Heidelberg (2008)
37. Gilbert, H., Minier, M.: A Collision Attack on 7 Rounds Rijndael. In: Third AES Candidate Conference (AES3), New York, pp. 230–241 (2000)
38. Daemen, J., Rijmen, V.: AES Proposal: Rijndael (1998)
39. Knudsen, L.R., Rijmen, V.: Known-Key Distinguishers for Some Block Ciphers. In: Kurosawa, K. (ed.) ASIACRYPT 2007. LNCS, vol. 4833, pp. 315–324. Springer, Heidelberg (2007)
40. Wei, Y., Lu, J., Hu, Y.: Meet-in-the-Middle Attack on 8 Rounds of the AES Block Cipher Under 192 Key Bits. In: Bao, F., Weng, J. (eds.) ISPEC 2011. LNCS, vol. 6672, pp. 222–232. Springer, Heidelberg (2011)
41. Yasuda, K.: Sandwich Is Indeed Secure: How to Authenticate a Message with Just One Hashing. In: Pieprzyk, J., Ghodosi, H., Dawson, E. (eds.) ACISP 2007. LNCS, vol. 4586, pp. 355–369. Springer, Heidelberg (2007)
42. Kaliski Jr., B.S., Robshaw, M.J.B.: Message Authentication with MD5. Technical report, CryptoBytes (1995)
43. Abe, M. (ed.): ASIACRYPT 2010. LNCS, vol. 6477. Springer, Heidelberg (2010)

Multiple Differential Cryptanalysis of Round-Reduced PRINCE

Anne Canteaut[1(✉)], Thomas Fuhr[2], Henri Gilbert[2], María Naya-Plasencia[1], and Jean-René Reinhard[2]

[1] Inria, Rocquencourt, France
anne.canteaut@inria.fr
[2] ANSSI, Paris, France

Abstract. PRINCE is a lightweight block cipher proposed by Borghoff *et al.* at Asiacrypt 2012. Due to its originality, novel design and low number of rounds, it has already attracted the attention of a large number of cryptanalysts. Several results on reduced versions have been published to date; the best one is an attack on 8 rounds out of the total number of 12. In this paper we improve this result by two rounds: we provide an attack on 10 rounds of the cipher with a data complexity of $2^{57.94}$ and a time complexity of $2^{60.62}$, corresponding to 118.56 security bits, instead of 126 for the generic attacks. Our attack uses multiple differentials and exploits some properties of PRINCE for recovering the whole key. PRINCE is defined as a member of a family of ciphers, differing by the choice of an Sbox among a distinguished set. We also show that the security offered by all the members of the family is not equivalent, by identifying an Sbox for which our attack can be extended up to 11 rounds with a data complexity of $2^{59.81}$ and a time complexity of $2^{62.43}$.

Keywords: Differential cryptanalysis · PRINCE · Multiple differentials · Key-recovery

1 Introduction

The area of lightweight primitives has drawn considerable attention over the last years, due to the need of low-cost cryptosystems for several emerging applications like RFID tags and sensor networks. The strong demand from industry has led to the design of a large number of lightweight block ciphers, with different implementation features. Among the best studied proposals are the ISO/IEC standards PRESENT [4] and CLEFIA [16], as well as LBlock [19], TWINE [18], LED [12] and KLEIN [11]. In this context, the need for a significant cryptanalysis effort is obvious. The demand from industry for clearly recommended lightweight ciphers requires that the large number of these potential candidates be narrowed down. Since the trade-off between the performance and the security is a major

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

C. Cid and C. Rechberger (Eds.): FSE 2014, LNCS 8540, pp. 591–610, 2015.
DOI: 10.1007/978-3-662-46706-0_30

issue for lightweight primitives, it is also very important to estimate the security margin of these ciphers, to determine for instance if some rounds need to be added, or if some can be omitted for achieving a given security level.

Recently, at Asiacrypt 2012, a new lightweight cipher named PRINCE has been proposed by Borghoff *et al.* [5]. This block cipher, which aims at low-latency encryption, has already received a lot of attention from the community, mainly due to its simplicity, originality and low number of rounds. These results include some small improvements upon the generic attack against the full cipher (that implies a reduction from 2^{127} to 2^{126} of the claimed security bound on the product $\mathsf{Data} \times \mathsf{Time}$ of the data and time complexities for any attack) [1,13]. Against round-reduced versions of PRINCE, the best attack published so far applies to 8 rounds (out of 12) [8] and an attack against 9 rounds has been described very recently [14]. Also some analysis of the building block PRINCE_{core} and some interesting results have been obtained on a variant of PRINCE using a chosen weak constant α [13,17]. However, this constant is not a parameter of the design of PRINCE. An attack in the multi-user setting has also been presented in [9].

In this paper, we propose a differential-type attack on round-reduced PRINCE that increases the number of analyzed rounds to 10 rounds, without modifying the constants or building-blocks in the cipher. It is a multiple differential attack, based on a principle similar to the one in [15], that we have combined with a sophisticated key recovery method specific for PRINCE. The fact that the linear layer in PRINCE is based on the same design strategy as the AES aims at making it resistant to classical differential attacks. In particular, due to the branch number of the linear transformation, any differential characteristic over four consecutive rounds has at least 16 active Sboxes, implying that the probability of any differential characteristic over the 12 rounds is at most 2^{-96}. Nevertheless, our attack exploits the following four properties coming from the main features of PRINCE:

- there exist many differentials for the round function with 4 active Sboxes and with an activity pattern having a particular shape (the active nibbles are the corners of a square);
- several of the characteristics obtained by iterating these round differentials have the same input and the same output differences, leading to some r-round differentials whose probability is much higher than the probability of a single characteristic;
- for a given pair of input and output activity patterns, we find several good differentials, which can be exploited together in a multiple differential attack, as proposed in [2,3];
- because of the particular shape of the activity patterns, these differentials can be extended by two rounds in each direction. Indeed, for some fixed input and output activity patterns, the active nibbles only depend on half of the bits of the plaintext and of the ciphertext, and on 66 of the 128 key bits.

Altogether, these four properties enable us to describe the first attack on 10-round PRINCE, which requires $2^{57.94}$ chosen plaintexts, with time complexity less that $2^{60.61}$ encryptions, leading to a product $\mathsf{Data} \times \mathsf{Time}$ around $2^{118.56}$.

Table 1. Summary of all attacks on PRINCE in the single-key setting, including the attacks presented in this paper.

Part of PRINCE	Source	Rounds	**D**ata	**T**ime	D × T	Memory	Attacks
PRINCE	[13]	6	2^{16}	2^{64}	2^{80}	2^{16}	Integral
	[8]	8	1	2^{123}	2^{123}	2^{20}	Sieve-in-the-middle
	[14]	9	2^{57}	2^{64}	2^{121}	$2^{57.3}$	Meet-in-the-middle
	Section 6	9	$2^{46.89}$	$2^{51.21}$	$2^{98.10}$	$2^{52.21}$	Multiple differential[a]
	Section 6	10	$2^{57.94}$	$2^{60.62}$	$2^{118.56}$	$2^{61.52}$	Multiple differential[a]
PRINCE(chosen α)	[13]	12	2^{41}	2^{41}	2^{82}	-	Boomerang
PRINCE_{core}	[13]	6	2^{16}	2^{30}	2^{46}	2^{16}	Integral
	[1]	12	2^{40}	$2^{62.72}$	$2^{102.72}$	2^{8}	Biclique
PRINCE-Family	Section 6	10	$2^{50.42}$	$2^{53.61}$	$2^{104.03}$	$2^{54.00}$	Multiple differential[a]
(modified Sbox)	Section 6	11	$2^{59.81}$	$2^{62.43}$	$2^{122.24}$	$2^{63.39}$	Multiple differential[a]

[a]Memory complexity figures can be slightly larger than time complexities due to the relative cost between an encryption and a memory access.

Another interesting issue is that, besides PRINCE, the designers have proposed a whole family of ciphers, named the PRINCE-Family, which differ in their 4 × 4 Sbox only. Since the Sbox of any member in the PRINCE-Family guarantees the same resistance to classical attacks, including differential attacks, all those ciphers seem to offer a similar security. Here, we show that it is not the case since all these Sboxes do not have the same behaviour regarding our attack. In particular, we exhibit a member of the PRINCE-Family for which up to 11 rounds can be attacked with data and time complexity satisfying $\mathsf{Data} \times \mathsf{Time} = 2^{122.24}$. The complexities of our attacks and a comparison with the previous results are given in Table 1.

The paper is organized as follows. After a description of PRINCE in Sects. 2 and 3 exhibits some differential paths with 4 active nibbles per round only, and gives a lower bound on the probabilities of some related differentials. Section 4 shows how these r-round differentials can be extended by two rounds at the beginning and by two rounds at the end, in a way such that some key bits can be recovered. Some experimental results supporting the previously made assumptions are presented in Sect. 5. Section 6 discusses and presents the results on 9 and 10 rounds of PRINCE and of some other element in the PRINCE-Family.

2 The PRINCE Block Cipher

PRINCE operates on 64-bit blocks and uses a 128-bit key composed of two 64-bit elements, K_0 and K_1. Its structure is depicted on Fig. 1.

PRINCE is based on the so-called FX-construction: two whitening keys $W_{in} = (K_0 \oplus K_1)$ and $W_{out} = (K_0' \oplus K_1)$ are XORed respectively to the input and to the output of a 12-round core cipher, named PRINCE_{core}, parametrized by K_1 only. The value of K_0' involved in the post-whitening key is derived from K_0 by $K_0' = (K_0 \ggg 1) \oplus (K_0 \gg 63)$.

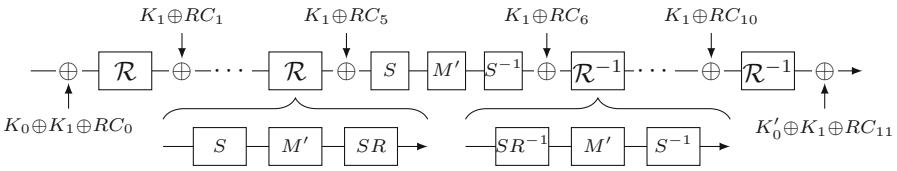


Fig. 1. Structure of PRINCE.

The internal state of the cipher is represented by a 4×16 binary matrix. The lower right corner of this matrix is the least significant bit of the input, while the upper left corner is the bit of index 63. This 4×16 binary matrix can also be seen as a 4×4 matrix of nibbles. In this case, the nibbles are numbered by their positions in the matrix, where the rows (resp. the columns) are numbered from top to bottom (resp. from left to right). The precise numbering of bit positions and nibbles is depicted on Fig. 2. We adopt the following notation. For a 64-bit state, plaintext or ciphertext value W, when W is seen as a matrix of nibbles, W^j designates column j of W, and $W^{i,j}$ designates the nibble at row i and column j. W_i designates the bit at position i. For a 64-bit key value K, K^i designates the bit of K at position i. For a general 64-bit value W and a set of bit indexes E, W^E designates the set of bits of W at bit positions in the set E.

The round function is then composed of:

- a non-linear layer S corresponding to 16 parallel applications of a 4×4 Sbox σ, which operates on the 16 nibbles of the internal state.
- a linear layer $SR \circ M'$, where M' is the parallel application of 4 involutive MixColumn transformations, each operating on 16 bits. The same transformation is applied on first and last (resp. on second and third) columns of the state. This transformation is then followed by a permutation SR of the 16 nibbles which is similar to the AES ShiftRows operation: in the 4×4 matrix of nibbles, the row of index i is rotated by i positions to the left.
- the addition of a round constant RC_i and of the subkey K_1.

The first 5 rounds in PRINCE correspond to the previously described round permutation $\mathcal{R}$, while the last 5 rounds are defined by the inverse permutation $\mathcal{R}^{-1}$.

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

Bits numbering

0,0	0,1	0,2	0,3
1,0	1,1	1,2	1,3
2,0	2,1	2,2	2,3
3,0	3,1	3,2	3,3

Nibbles numbering

Fig. 2. Numbering of nibbles and bits of the internal state.

The middle round corresponds to the successive applications of S, M' and S^{-1} (see Fig. 1).

One of the main features of PRINCE$_{core}$ is that decryption can be implemented by the same circuit as encryption, but under another key. This comes from the fact that the 12 round constants satisfy $RC_i \oplus RC_{11-i} = \alpha$ for $0 \leq i \leq 11$, where α is a fixed constant, implying that decryption under key K_1 corresponds to encryption under key $K_1 \oplus \alpha$.

Round-Reduced Versions of PRINCE. The number of rounds in PRINCE corresponds to the number of nonlinear layers. There are 12 rounds for the full cipher. Round-reduced versions are defined in a similar way: if the overall number of rounds r is even, the numbers of rounds before and after the middle transformation are the same, while they differ by 1 when r is odd.

Some Observations on the Linear Layer. We note that M' can be expressed as the parallel application of 16 independent transformations operating on one column of bits of the internal state. These 4-bit transformations have the following form:

$$x = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} \mapsto \begin{pmatrix} wt(x) & \mod 2 \\ wt(x) & \mod 2 \\ wt(x) & \mod 2 \\ wt(x) & \mod 2 \end{pmatrix} \oplus \mathsf{Rot}_{n_i} \begin{pmatrix} x_4 \\ x_3 \\ x_2 \\ x_1 \end{pmatrix},$$

where $wt(x)$ is the Hamming weight of x and Rot_{n_i} denotes some rotation by n_i positions to the top. In other words, M' is the composition of three simple columnwise operations: the inversion of the order of the four bits, followed by a bitwise rotation, completed by the addition of the parity of the column to each bit of the column. The whole linear layer when the weight of each column is even is depicted in the appendix of [7].

3 Differentials with 4 Active Nibbles for PRINCE$_{core}$

We now study some differentials for PRINCE$_{core}$ with particular activity patterns, and we compute a lower bound on their probabilities.

In the following, we denote by R the permutation corresponding to $S \circ SR \circ M'$. Evaluating the difference propagation through R enables to evaluate the difference propagation through one round $\mathcal{R}$ of PRINCE$_{core}$ since key and round constant additions do not alter the differences. To assess the probability of characteristics over several rounds, we will consider that the transition probabilities in different rounds are independent. Due to the absence of key addition around the middle M' layer, we apply a specific analysis to the surrounding non-linear layers. We denote by S_{sbox} the permutation $S^{-1} \circ M' \circ S$, which covers one Super Sbox, as defined in [10]. Here, we study some differentials for the function

$$\mathcal{F}_{r_1+r_2+2} = \left(R^{-1}\right)^{r_2} \circ M' \circ SR^{-1} \circ S_{sbox} \circ SR \circ M' \circ R^{r_1},$$

which should also be good differentials for $\mathcal{F}^{K_1}_{r_1+r_2+2}$, the function obtained by considering $(r_1 + r_2 + 2)$ rounds of PRINCE$_{core}$, with constants, key additions, and an additional linear layer at the beginning and at the end.

In the following, the internal state is seen as a 4×4 matrix of nibbles numbered according to Fig. 2. Since any differential characteristic over four consecutive rounds has at least 16 active Sboxes [6, Theorem 2], we here focus on differential characteristics with four active Sboxes in each round. We show that several such characteristics can be built, and we additionally exhibit such characteristics sharing their input and output differences. In other words, we are able to find a lower bound on the probability of some *differentials* which include several differential characteristics with the lowest possible number of active Sboxes.

The crucial observation is that the diffusion in the linear layer is ensured by the addition of the parity in M'. In order to build some characteristics with a low number of active Sboxes, we focus on the differences with four active nibbles located in two columns, such that the two active nibbles in the same column are identical. This ensures that no diffusion of the differences takes place since the corresponding parity bits are inactive. We further restrict the differences considered to those whose active nibbles are the corners of a 3×3 *square*, *i.e.*, the positions of the four active nibbles in the 4×4 internal state are of the form (i, c), $(i+2, c)$, $(i, c+2)$ and $(i+2, c+2)$ for some i and c in $\{0, 1\}$. A difference satisfying these properties will be called *valid*.

There are exactly four square activity patterns and each of them is denoted by $[i, c]$ with the minimal values of i and c. In the following, any valid difference is then characterized by its activity pattern $[i, c]$ and by the two differences (δ_1, δ_2) where δ_1 (resp. δ_2) denotes the difference which appears in Column c (resp. in Column $(c + 2)$).

3.1 Computing the Transition Probabilities

There are exactly $15^2 = 225$ valid differences for each activity pattern. We now determine when a valid difference can be mapped into another valid difference by $R = S \circ SR \circ M'$. Since S operates on the nibbles independently, it does not affect the activity pattern. Then, we need to determine when a square activity pattern can be transformed into another square activity pattern by the linear layer $SR \circ M'$. It can be easily checked that this situation occurs if and only if the nibble differences (δ_1, δ_2) belong to the 18 pairs of the form

$$(\delta_1, \delta_2) \in (\Delta_1 \times \Delta_2) \cup (\Delta_2 \times \Delta_1) \text{ with } \Delta_1 = \{1, 4, 5\} \text{ and } \Delta_2 = \{2, 8, 10\}.$$

Then, there are exactly $4 \times 2 \times 9 = 72$ nonzero valid differences which are mapped by $SR \circ M'$ to a difference with a square activity pattern. The resulting differences have equal nibble differences on the square pattern diagonals. An example of such a propagation is depicted on Fig. 3. Therefore, a valid input difference with active nibbles (δ_1, δ_2) can be mapped by R to a valid difference with active nibbles (δ'_1, δ'_2) if and only if the following four transitions for the Sbox are valid: $\delta_1 \mapsto \delta'_1$, $\delta_2 \mapsto \delta'_2$, $\delta_1 \mapsto \delta'_2$ and $\delta_2 \mapsto \delta'_1$. Then, the probability

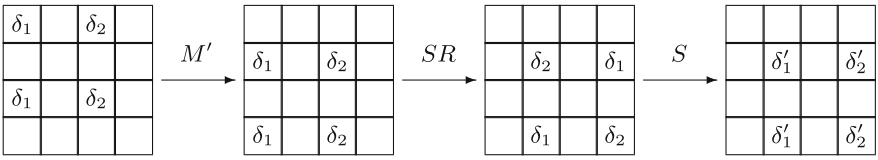


Fig. 3. Propagation of a valid input difference over one round R when $(\delta_1, \delta_2) \in \Delta_1 \times \Delta_2$ (if $(\delta_1, \delta_2) \in \Delta_2 \times \Delta_1$, the above sequence $[0,0] \rightarrow [1,0] \rightarrow [1,1] \rightarrow [1,1]$ of square activity patterns would be replaced by $[0,0] \rightarrow [0,0] \rightarrow [0,0] \rightarrow [0,0]$).

of the transition $(\delta_1, \delta_2) \mapsto (\delta'_1, \delta'_2)$ for R does not vary when the roles of δ_1 and δ_2 are inverted, or when the roles of δ'_1 and δ'_2 are inverted. This important property will be exploited in the attack. Thus, we consider the 9×9 transition matrix μ whose entry at the intersection of Row $(\delta_1, \delta_2) \in \Delta_1 \times \Delta_2$ and Column $(\delta'_1, \delta'_2) \in \Delta_1 \times \Delta_2$ is the product of the four probabilities:

$$p(\delta_1, \delta'_1)p(\delta_2, \delta'_1)p(\delta_1, \delta'_2)p(\delta_2, \delta'_2),$$

where

$$p(\delta, \delta') = \Pr_X[\sigma(X \oplus \delta) \oplus \sigma(X) = \delta']$$

and σ is the 4×4 Sbox used in PRINCE. Now, for a given input activity pattern, all 9 (δ_1, δ_2) in $\Delta_1 \times \Delta_2$ lead to the same output activity pattern, while all 9 (δ_1, δ_2) in $\Delta_2 \times \Delta_1$ lead to another one. Then, the whole 72×72 transition matrix can be written as the Kronecker product[1] $A \otimes \mu$, where the 8×8 matrix A encodes the transition between the 4 square activity patterns, together with the fact that δ belongs either to $\Delta_1 \times \Delta_2$ or to $\Delta_2 \times \Delta_1$, and the 9×9 matrix μ encodes transition between values of $\delta \in \Delta_1 \times \Delta_2$. The values of these two matrices are given in [7]. Thus, the transition matrix corresponding to r iterations of R is given by

$$(A \otimes \mu)^r = (A^r \otimes \mu^r).$$

In the same way, we can define the matrices B and ν that correspond to the middle round $M' \circ SR^{-1} \circ S_{sbox} \circ SR \circ M'$. Note that due to the involutivity of S_{sbox}, ν is symmetric. Obviously, the transition matrix for R^{-1} is the transpose of the transition matrix for R, *i.e.*,

$$(A \otimes \mu)^T = (A^T \otimes \mu^T).$$

It eventually follows that the transition matrix for $\mathcal{F}_{r_1+r_2+2}$ is

$$\left(A^{r_1} B (A^{r_2})^T \otimes \mu^{r_1} \nu (\mu^{r_2})^T\right).$$

[1] The Kronecker product of an $m \times n$ matrix A and a $p \times q$ matrix B is the $mp \times nq$ block matrix whose block at Position (i, j) equals $A_{i,j}B$.

It can be checked that $AB = B(A)^T = J$, where J is the matrix with all entries equal to 1. Since all rows and all columns of A have weight 2, we deduce that $AJ = 2J$ and $JA^T = 2J$, implying that, for any $r_1 + r_2 \geq 1$,

$$A^{r_1} B (A^{r_2})^T = 2^{r_1+r_2-1} J.$$

It follows that, for any valid input difference defined by $\Delta_{in} = (\delta_1, \delta_2)$ and with any given input activity pattern, the probability that the output difference is a valid difference $\Delta_{out} = (\delta_1', \delta_2')$ with a given output activity pattern is

$$2^{r_1+r_2-1} \left(\mu^{r_1} \nu (\mu^{r_2})^T\right)_{i,j}$$

where i and j are the row and column indices corresponding to Δ_{in} and Δ_{out}. It is worth noticing that this probability depends on $(\Delta_{in}, \Delta_{out})$ only, and is independent from the input and output square activity patterns.

3.2 Results for the Sbox Used in PRINCE

By computing $\mu^{r_1} \nu (\mu^{r_2})^T$, we get the following results for the Sbox used in PRINCE (the transition matrices are given in the appendix of [7]).

- For $r_1 = r_2 = 2$, i.e., for six rounds, the highest coefficient of the matrix $\mu^2 \nu (\mu^2)^T$ is $2^{-70} \times 1536$. Then, the best differential has probability

$$2^{-70} \times 1536 \times 2^3 \approx 2^{-56.42}.$$

  This probability is obtained for four differentials, namely $(\Delta_{in}, \Delta_{out}) \in \{(1,2), (2,1)\} \times \{(1,2), (2,1)\}$, and for any fixed input and output activity patterns.
- For $r_1 = 1$ and $r_2 = 2$, the probability of the best differential is $2^{-47.42}$. For $r_1 = 2$ and $r_2 = 1$, the transition matrix $\mu^2 \nu (\mu)^T$ is obviously the transpose of the previous one, leading to the same maximal transition probability.

3.3 Results for Another Sbox Used in the PRINCE-Family

The PRINCE-family consists of all ciphers defined as PRINCE but with an Sbox σ which can be chosen among eight 4×4 Sboxes and all the affinely equivalent transformations (see Appendix B in [6]). All these Sboxes are expected to have the same properties regarding classical differential attacks. However, when we focus on the valid differentials with square activity patterns, these Sboxes do not have the same behaviour. We have searched for some Sbox minimizing the complexity of the attack described in the next section. An optimal Sbox in that sense, linearly equivalent to Sbox S_5 from [6], is

x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$\sigma[x]$	0	A	6	5	8	D	3	4	7	C	2	E	9	F	B	1

For this Sbox, we get the following results (the corresponding transition matrices are given in [7]).

– For $r_1 = r_2 = 2$, i.e., for six rounds, the highest probability is obtained for four differentials, namely when $(\Delta_{in}, \Delta_{out}) \in \{(5,2),(2,5)\} \times \{(5,2),(2,5)\}$. The corresponding probability is 2^{-50}.
– For $r_1 = 2$ and $r_2 = 3$, i.e., for seven rounds, the probability of the best differential is 2^{-58}.

4 Key Recovery on Round Reduced Versions of PRINCE

In this section we show how the previously described differentials can be extended by up to 4 full rounds, in order to recover the key of reduced variants of PRINCE, in a chosen-plaintext scenario.

4.1 General Principle

Our attack applies to r rounds of PRINCE with $r > 4$. We call *reduced cipher*, the cipher derived from r rounds of PRINCE$_{core}$ by removing, both at the beginning and at the end of the cipher, one full round and an additional nonlinear layer. Therefore, the reduced cipher corresponds to the $(r-4)$-round function defined in the previous section $\mathcal{F}^{K_1}_{r_1+r_2+2}$ with $r_1 + r_2 + 2 = r - 4$ and $|r_1 - r_2| \leq 1$. Our attack exploits together several differentials with the same input and output activity patterns for the reduced cipher, as proposed in [2,3]. Indeed, there exist several differentials with the same square activity pattern which have similar probabilities. In particular, if $(\delta_1, \delta_2) \mapsto (\delta'_1, \delta'_2)$ holds with probability p, so do $(\delta_2, \delta_1) \mapsto (\delta'_1, \delta'_2)$, $(\delta_1, \delta_2) \mapsto (\delta'_2, \delta'_1)$ and $(\delta_2, \delta_1) \mapsto (\delta'_2, \delta'_1)$.

These square differentials for $(r-4)$ rounds of the reduced cipher can be exploited to perform a key-recovery attack on r rounds of PRINCE. Actually, due to the very simple key schedule of PRINCE, the knowledge of 66 key bits only (out of 128) allows to determine whether a given plaintext-ciphertext pair corresponds to a pair of input and output of the reduced cipher which follows one of the considered differentials. Moreover, there is an efficient procedure for deriving, from each plaintext-ciphertext pair, the list of all partial key candidates which lead to one of the expected differentials for the reduced cipher. Assuming the considered differentials have good probabilities, the correct partial key is then suggested with higher probability than the other ones. Our attack then follows the general principle of statistical attacks on block ciphers: the first part is a distillation phase which counts how many times the partial keys are suggested by the available plaintext-ciphertext pairs. Then the analysis phase selects the partial keys suggested by at least τ pairs, where the threshold τ is a parameter of the attack. The search phase eventually consists in finding the key of the cipher from the identified list of partial keys.

For the sake of clarity, we first focus on input and output differences corresponding to the $[0,0]$-activity pattern. From now on, we denote by $\Delta = (\Delta_{in}, \Delta_{out})$ a differential with input and output having the $[0,0]$-activity pattern, and by $p_{true}(\Delta)$ its transition probability. We consider a set Σ of D such differentials. Σ_{in} denotes the set of all input differences Δ_{in} such that $(\Delta_{in}, \Delta_{out})$

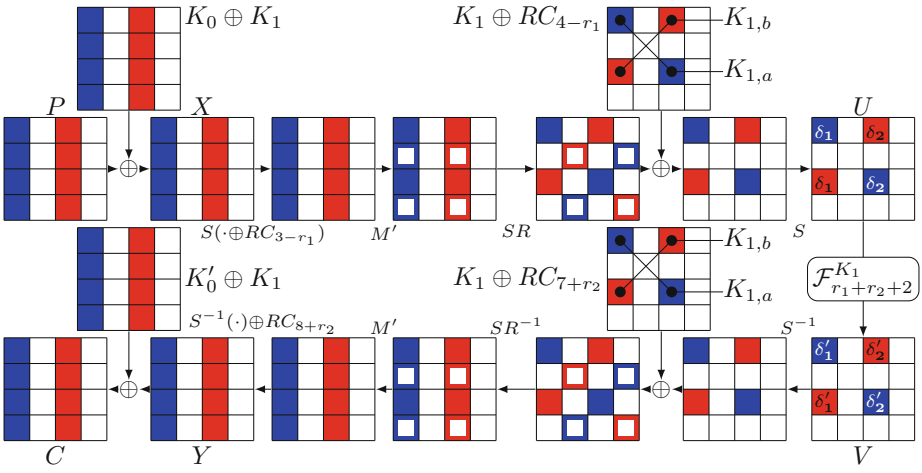


Fig. 4. Extension of the reduced cipher square differentials. The blue (resp. red) nibbles correspond to plaintext/ciphertext column 0 (resp. 2). The colored nibbles of the states can be recovered from the corresponding colored nibbles of the key material, plaintext and ciphertext. In the cipher states, full colored nibbles indicate active nibbles, emptied colored nibbles and empty nibbles indicate no difference (Color figure online).

belongs to Σ for some Δ_{out}. Similarly, Σ_{out} is the set of all output differences in Σ. The sizes of these sets are denoted by D_{in} and D_{out} respectively.

4.2 Extension of the Reduced Cipher Square Differentials

Construction of Structures. Our attack makes use of structures of plaintexts. A structure S_{P^1,P^3} is a set of 2^{32} pairs of plaintexts and ciphertexts, defined as follows. Columns 1 and 3 of all the plaintexts in the structure share the same values P^1 and P^3, while Columns 0 and 2 take all the $(2^{16})^2 = 2^{32}$ possible values. Building such a structure requires 2^{32} encryption queries.

Let us consider any differential $\Delta = (\Delta_{in}, \Delta_{out})$ in Σ. For any plaintext P in the structure, we denote by X the (unknown) value obtained during the encryption of P after addition of the whitening key W_{in}, and by U the (unknown) value at the beginning of the reduced cipher. Similarly, with any ciphertext C we associate the value Y before the addition of the whitening key W_{out} and the value V at the end of the reduced cipher (see Fig. 4).

Then, the plaintext P_Δ corresponding to $U \oplus \Delta_{in}$ has the same value as P on Columns 1 and 3. Indeed these columns do not depend on the active nibbles of Δ_{in} when computing backwards as shown on Fig. 4. Therefore, P_Δ lies in the same structure as P. The subset of all $2^{31}(2^{32}-1)$ pairs of distinct plaintexts obtained from S_{P^1,P^3} can be partitioned into subsets of 2^{31} pairs, each corresponding to a given difference at the beginning of the reduced cipher. As a consequence, there are exactly 2^{31} pairs of distinct plaintexts from S_{P^1,P^3}

which lead to any given input difference $\Delta_{in} \in \Sigma_{in}$ for the reduced cipher. We can build N_s such structures for a cost of $N = 2^{32}N_s$ chosen plaintexts.

Right Pairs and Candidate Pairs. When considering the target key, the pairs that follow one of the differentials in Σ are called *right pairs*. Their number is about

$$N_{true} = 2^{31}N_s \sum_{\Delta \in \Sigma} p_{true}(\Delta) = \frac{N}{2} p_{true} \text{ where } p_{true} = \sum_{\Delta \in \Sigma} p_{true}(\Delta), \quad (1)$$

i.e., p_{true} is the sum of the probabilities of all the differentials in Σ.

For the right pairs, the difference at the end of the reduced cipher has the $[0, 0]$-activity pattern. Then, it can be seen on Fig. 4 that this difference can propagate only to Columns 0 and 2 of the ciphertexts. Thus, our criterion to detect potential right pairs is a collision on Columns 1 and 3 of the ciphertexts. All pairs with ciphertexts colliding on Columns 1 and 3 are named *candidate pairs*. Obviously, all right pairs belong to the set of candidate pairs. However, about $2^{63}N_s$ possible pairs can be obtained from the N_s structures, and each of those fulfills the criterion with probability 2^{-32}. Therefore we have about $F = 2^{31}N_s$ candidate pairs.

Guessing Key Bits. For any given candidate pair (P, P', C, C'), it can be easily seen that the difference $U \oplus U'$ at the beginning of the reduced cipher can be recovered from the plaintexts and from Columns 0 and 2 of the pre-whitening key $W_{in} = K_0 \oplus K_1$ and from four nibbles of K_1 (the colored nibbles in Fig. 4), namely the bits $K_1^{[20;23]\cup[28;31]\cup[52;55]\cup[60;63]}$. Similarly, the difference at the end of the reduced cipher depends on the ciphertexts, on the same four nibbles of K_1 and also on Columns 0 and 2 of the post-whitening key $W_{out} = K'_0 \oplus K_1$.

Since the two whitening keys are related by some simple formula deduced from the fact that $K'_0 = (K_0 \ggg 1) \oplus (K_0 \gg 63)$, the following property can be easily proved by induction.

Property 1. Let j and ℓ be two indices with $0 \leq j \leq \ell \leq 63$. From the knowledge of K_1^{ℓ} (or $K_0^{\ell+1}$, with $K_0^{64} = K_0^0$) and $W_{in}^{[j;\ell]}$ and $W_{out}^{[j;\ell]}$, one can derive linearly the values of $K_0^{[j;\ell]}$ and $K_1^{[j;\ell]}$ (with a perturbation by K_0^{63} at bit position 0).

Using Property 1, it is easy to establish that the key material required to ensure a candidate pair follows a differential corresponds to 66 information bits of the key: bits $K_1^{[16;31]\cup[48;63]}$ and bits $K_0^{0\cup[16;32]\cup[48;63]}$.

Distillation Phase Overview. Taking into account the structure used by the attack, the distillation phase basic principle is given in Algorithm 1.

We show in the following subsections how the partial keys corresponding to candidate pairs can be identified efficiently during the distillation phase. We remark that a second distillation phase can be performed by considering another

Algorithm 1. Distillation phase of the attack.

for all N_s structures of plaintexts **do**
  for all pairs (P, P') in the structure such that $(C^1 \oplus C'^1 = C^3 \oplus C'^3) = 0$ **do**
    for all 66-bit partial keys k which lead to $(U \oplus U', V \oplus V') \in \Sigma$ **do**
      Increment the counter N_k

input-output activity pattern for the differentials of the reduced cipher, and we study the distribution of the counters N_k as a function of the data complexity and a threshold parameter τ.

4.3 Identification of Key Candidates in the Distillation Phase

Overview. The attack described above can be performed only if, for each candidate pair, the partial keys which lead to a differential in Σ for the reduced cipher can be efficiently determined. Finding these key candidates is not a trivial task in the general case. In the following, we describe a method to achieve it with a time complexity close to D encryptions per candidate pair, where D is the number of differentials considered in the attack.

First, let us remark that the first (resp. last) two rounds of the cipher, before the beginning (resp. the end) of the reduced cipher $\mathcal{F}^{K_1}_{r_1+r_2+2}$, act independently on the different columns (or diagonals) of the state, as depicted on Fig. 4. Thus, for every differential $\Delta \in \Sigma$, for column $i \in \{0, 2\}$ and for every partial values of K_1 ($K_{1,a}$ or $K_{1,b}$), one can precompute all compatible input-output column pairs $(X^i, X^i \oplus \Delta X^i, Y^i, Y^i \oplus \Delta Y^i)$ for the cipher without its pre- and post-whitening, as detailed in the precomputation step paragraph below.

Then, the processing step successively examines all candidate pairs of plaintext-ciphertext for all considered differentials of the reduced cipher. Since the whitening keys satisfy $W_{in} = P \oplus X$ and $W_{out} = C \oplus Y$, the precomputed tables enable us to determine all partial values of the whitening keys which can lead to a differential in Σ for the reduced cipher. Among these whitening keys, further filtering is required to ensure constraints stemming from the key schedule are satisfied.

Now, we describe in details how lists can be precomputed and carefully merged in order to reduce the complexity of the attack. The list elements are tuples of values. We will assume that the lists are sorted lexicographically after being generated.

Precomputation Step. At the input of the reduced cipher, the four active nibbles can be divided into two pairs. Indeed, referring to Fig. 4, column X^0 (resp. X^2) of X can be computed from the two blue (resp. red) active nibbles of U, the two blue (resp. red) nibbles of K_1 and two additional inactive nibbles in the blue (resp. red) column after the M' layer. Therefore, we precompute the following lists: for all $\Delta_{in} = (\delta_1, \delta_2) \in \Sigma_{in}$, $\mathcal{L}^{in,a}_{\Delta_{in}}$ is composed of triples $(K_{1,a}, \Delta X^0, X^0)$ where X^0 and ΔX^0 are two 16-bit elements and $K_{1,a}$ is an 8-bit part of K_1 corresponding to its two blue nibbles on Fig. 4. The values X^0

and $(X^0 \oplus \Delta X^0)$ are the values on Column 0 of some internal states X and X' which lead to Δ_{in} on the blue nibbles at the input of the reduced cipher given $K_{1,a}$. Similarly, $\mathcal{L}^{in,b}_{\Delta_{in}}$ contains triples $(K_{1,b}, \Delta X^2, X^2)$ where $K_{1,b}$ corresponds to the two red nibbles of K_1, and X^2 and $(X^2 \oplus \Delta X^2)$ are the values on Column 2 of some X and X' which lead to Δ_{in} on the red nibbles at the input of the reduced cipher. The detailed algorithm for computing $\mathcal{L}^{in,a}_{\Delta_{in}}$ is given in Algorithm 2.

Algorithm 2. Precomputation step of the distillation phase

$\mathcal{L} \leftarrow \{(z_0, z_2, z'_0, z'_2) \in (\mathbf{F}_2^4)^4 \ : \sigma(z_0) \oplus \sigma(z'_0) = \delta_1$ and $\sigma(z_2) \oplus \sigma(z'_2) = \delta_2$ with $(\delta_1, \delta_2) = \Delta_{in}\}$
for all (z, z') in $\mathcal{L}$ **do**
  for all $K_{1,a} = (k_{0,0}, k_{2,2})$ corresponding to the two blue nibbles of K_1 **do**
    for all pairs of nibbles (z_1, z_3) **do**
      $X^0 \leftarrow S^{-1}(M'(z_0 \oplus k_{0,0} \oplus RC^{0,0}_{4-r_1}, z_1, z_2 \oplus k_{2,2} \oplus RC^{2,2}_{4-r_1}, z_3)) \oplus RC^0_{3-r_1}$
      $\Delta X^0 \leftarrow X_0 \oplus S^{-1}(M'(z'_0 \oplus k_{0,0} \oplus RC^{0,0}_{4-r_1}, z_1, z'_2 \oplus k_{2,2} \oplus RC^{2,2}_{4-r_1}, z_3)) \oplus RC^0_{3-r_1}$
      Add $(K_{1,a}, \Delta X^0, X^0)$ to $\mathcal{L}^{in,a}_{\Delta_{in}}$

The first list $\mathcal{L}$ has size exactly 2^8. Then, each list $\mathcal{L}^{in,a}_{\Delta_{in}}$ contains 2^{24} elements, as well as the lists $\mathcal{L}^{in,b}_{\Delta_{in}}$. Similarly, we compute the $2D_{out}$ lists $\mathcal{L}^{out,a}_{\Delta_{out}}$ and $\mathcal{L}^{out,b}_{\Delta_{out}}$, each of size 2^{24} elements, which correspond to the possible pairs of values for Y and Y' on Column 0 and Column 2, respectively.

For a PRINCE computation, the eight bits of K_1 involved in $\mathcal{L}^{out,a}_{\Delta_{out}}$ (resp. in $\mathcal{L}^{out,b}_{\Delta_{out}}$) are the same as the ones involved in $\mathcal{L}^{in,a}_{\Delta_{in}}$ (resp. in $\mathcal{L}^{in,b}_{\Delta_{in}}$). Therefore, the lists sharing the same key bits can be merged. For any $\Delta = (\Delta_{in}, \Delta_{out})$, the two lists $\mathcal{L}^{in,a}_{\Delta_{in}}$ and $\mathcal{L}^{out,a}_{\Delta_{out}}$ then lead to a list $\mathcal{L}^a_\Delta$ of size 2^{40} composed of tuples $(\Delta X^0, \Delta Y^0, T, K_{1,a}, X^0, Y^0)$. T is a 3-bit linear tag appended to each element of the list, whose role will be explained later. The overall time complexity of this merging process is proportional to $2^{40}D$. The memory required for storing the D lists $\mathcal{L}^a_\Delta$ corresponds to $2^{40}D$ elements of 75 bits. Similarly, the lists $\mathcal{L}^{in,b}_{\Delta_{in}}$ and $\mathcal{L}^{out,b}_{\Delta_{out}}$ are merged into D lists $\mathcal{L}^b_\Delta$ of size 2^{40}.

Processing Step. We now explain how, for each candidate pair, (P, P', C, C'), we compute all partial keys which lead to a differential in Σ for the reduced cipher. Since all possible values for Columns 0 and 2 of (X, X', Y, Y') have been precomputed and $P \oplus P' = X \oplus X', C \oplus C' = Y \oplus Y'$, the precomputed lists provide us with the list of all possible values for the whitening keys $W_{in} = K_0 \oplus K_1$ and $W_{out} = K'_0 \oplus K_1$ on Columns 0 and 2, along with co-determined values of some bits of Columns 0 and 2 of K_1.

Moreover, since for all $i \neq 0$ we have $X_i \oplus P_i = K^i_0 \oplus K^i_1$ and $Y_i \oplus C_i = K^{i+1}_0 \oplus K^i_1$, we also deduce the following relation between P, C, X, Y and K_1:

$$P_i \oplus C_{i-1} = X_i \oplus Y_{i-1} \oplus K^i_1 \oplus K^{i-1}_1. \tag{2}$$

For each element in one of the lists $\mathcal{L}^a_\Delta$, the right-hand term in the previous relation is known for $61 \leq i \leq 63$ since the nibble in Position $(0,0)$ of K_1 belongs to $K_{1,a}$. This is the 3-bit value, denoted by T, included in the tuples in $\mathcal{L}^a_\Delta$. Then, for each candidate pair examined in the attack, the corresponding value of $(P_i \oplus C_{i-1})$, $61 \leq i \leq 63$ is compared with T. The number of candidates for the value of W^0_{in} is then divided by a factor 8, on average. Similarly, the 3-bit value corresponding to the right-hand side of (2) for $28 \leq i \leq 31$ is included in each element of $\mathcal{L}^b_\Delta$. The 66-bit keys corresponding to a given candidate pair are then computed through Algorithm 3. The principle of this algorithm is to generate lists $\mathcal{L}^i$, $i \in \{0,2\}$ containing partial key values affecting mainly Column i and compatible with Column i of the considered candidate pair, to compute the key bits shared by elements in both lists, and finally to merge the two lists according to the shared bit values.

Algorithm 3. Processing step of the distillation phase

Input: a candidate pair (P, P', C, C')
$B \leftarrow (P_i \oplus C_{i-1}, 61 \leq i \leq 63)$; $B' \leftarrow (P_i \oplus C_{i-1}, 29 \leq i \leq 31)$.
for all $\Delta \in \Sigma$ **do**
  $\mathcal{L}^0 = \emptyset, \mathcal{L}^2 = \emptyset$
  for all elements in $\mathcal{L}^a_\Delta$ of the form $(P^0 \oplus P'^0, C^0 \oplus C'^0, B, K_{1,a}, X^0, Y^0)$ **do**
    $W^0_{in} \leftarrow X^0 \oplus P^0$; $W^0_{out} \leftarrow Y^0 \oplus C^0$.
    Compute $K_1^{[52;55]}$ with Property 1 starting from $K_1^{60} \in K_{1,a}$.
    Add $(K_1^{[52;55]}, K_1^{[20;23]}, W^0_{in}, W^0_{out}, K_1^{63})$ to $\mathcal{L}^0$ ($K_1^{[20;23]}$ is a nibble of $K_{1,a}$).
  for all elements in $\mathcal{L}^b_\Delta$ of the form $(P^2 \oplus P'^2, C^2 \oplus C'^2, B', K_{1,b}, X^2, Y^2)$ **do**
    $W^2_{in} \leftarrow X^2 \oplus P^2$; $W^2_{out} \leftarrow Y^2 \oplus C^2$.
    Compute $K_1^{[20;23]}$ with Property 1 starting from $K_1^{28} \in K_{1,b}$.
    Add $(K_1^{[52;55]}, K_1^{[20;23]}, W^2_{in}, W^2_{out}, K_1^{31})$ to $\mathcal{L}^2$ ($K_1^{[52;55]}$ is a nibble of $K_{1,b}$).
  for all pairs of elements in $\mathcal{L}^0, \mathcal{L}^2$ matching on their first two entries **do**
    Increment the counter N_k, with $k = (W^0_{in}, W^2_{in}, W^0_{out}, W^2_{out}, K_1^{63}, K_1^{31})$

For $\Delta \in \Sigma$, the number of elements in lists $\mathcal{L}^a_\Delta$ and $\mathcal{L}^b_\Delta$, which take a given value on the first 3 values ($\Delta X, \Delta Y$ and T) of their tuple, amounting to 35 bits, is 2^5 on average. For each of these elements, we need to compute 4 bits by a simple linear relation. The final merging step between $\mathcal{L}^0$ and $\mathcal{L}^2$ leads to 2^2 key candidates on average. It requires to sort the two lists according to their first two values, which can be done in linear time, for a cost of 2^6.

Thus the average complexity of Algorithm 3 is $2^6 D$ elementary operations and the average number of partial key candidates found for each candidate pair is $2^{-8} \times (2^5)^2 D = 4D$.

4.4 Iterating the Distillation Phase for Recovering the Whole Key

We have shown how to perform a distillation phase leveraging differentials with input and output $[0,0]$-activity patterns. It extracts information about the number of values taken by 66 key bits from structures of plaintext-ciphertext pairs.

We now remark that a second distillation phase can be performed, by adapting the procedure described in previous subsections to the case of $[0,1]$-activity pattern instead of the $[0,0]$-activity pattern. Then, the bits of the key involved in the computation of the active nibbles cover mainly Columns 1 and 3 (instead of 0 and 2).

In this second distillation phase, the conformity of a plaintext-ciphertext pair to one of the differentials in Σ can also be checked by guessing key nibbles and whitening keys columns. They can be shown to be equivalent to 66 bits of key information: bits $K_1^{[1;15]\cup[32;47]}$, $K_0^{[1;16]\cup[32;48]}$, $K_0^0 \oplus K_0^{63}$, and $K_1^0 \oplus K_0^{63}$.

Then, each of the 128 key bits is involved in at least one of the distillation phases. Each distillation phase restricts the value taken by about half of the key.

4.5 Complexity Analysis

Distillation Phase Complexity. For reasons of symmetry, the number of differentials D that we take into account and the corresponding probabilities p_{true} are the same in both parts. We also take in both cases the same value for the number of structures N_s. In each distillation step, Algorithm 3 is applied once for each candidate pair. Since the time complexity of the distillation phases is assessed in elementary operations (xor on 4 bits), the comparison with the complexity of the generic attack is not easy. We argue that the search for all partial key candidates for each candidate pair and each differential, which has been estimated to be 2^6 operations, is less costly than one full encryption. Indeed, each of the 11 applications of the linear layer M' required by a single encryption consists of four linear transformations on 16 bits. This obviously corresponds to more than 2^8 binary xors. Then, the time complexity of the distillation phase satisfies

$$\mathsf{Time}_1 = 2 \times 2^{31} N_s \times 2^6 D \text{ elementary operations } \leq \frac{2N_{true}}{p_{true}} D \text{ encryptions.}$$

Since each distillation phase requires specific structures, we have

$$\mathsf{Data} = 2 \times 2^{32} N_s = \frac{4N_{true}}{p_{true}}.$$

The memory complexity of our attack comes essentially from the storage of partial key candidates counters during the distillation phase. It is worth noticing that the total number of partial key candidates considered during the distillation is bounded by $4D \times 2^{31} N_s$. Using an appropriate data structure to access the counters, the memory required is then bounded by

$$\mathsf{Memory} \leq 4D \times \frac{N_{true}}{p_{true}}.$$

Search Phase Complexity. We now want to determine to what extent the statistic on the keys resulting from the distillation phases reduces the size of the search space.

All counters N_k correspond to the sum of some basic counters $N_k(P, P', C, C')$, one for each candidate pair. These basic counters are defined as

$$N_k(P, P', C, C') = \begin{cases} 1 & \text{if } (U \oplus U', V \oplus V') \in \Sigma \\ 0 & \text{otherwise.} \end{cases}$$

The basic counters for wrong keys follow a Bernoulli distribution. Indeed, let us consider any wrong key k. For each candidate pair, columns P^1, P'^1, P^3 and P'^3 are chosen according to the structure and $C^1 = C'^1$ and $C^3 = C'^3$. The values of columns 0 and 2 of P, P', C and C' are distributed uniformly among pairs that fulfill $P \neq P'$ and $C \neq C'$. The values of active diagonals at the beginning and at the end of the reduced cipher when encrypting P, P' and decrypting C, C' under key k follow the same distribution. The counter N_k is incremented if one of the D differences in Σ occurs at the beginning and at the end of the reduced cipher, which happens with probability $(2^{32} - 1)^{-2} D \approx 2^{-64} D = p_{false}$.

For the right key guess, the average value of N_k is N_{true}. Let us choose a threshold $\tau = N_{true}/\eta$. If τ decreases, the success probability of the attack increases, but so does the number of wrong keys reaching the threshold. Theorem 3 of [2] gives an estimation of the probability that a wrong key reaches this threshold, using accurate bounds for the tail distribution of the counter values for wrong keys. Using our notation, this estimation leads to $\Pr[N_k \geq \tau] \approx G(p_{true}/\eta D_{in}, p_{false}/D_{in}, \tau)$, where

$$G(x, y, t) = e^{\frac{-tD(x||y)}{x}} \left[\frac{x(1-y)}{(x-y)\sqrt{2\pi t(1-x)}} + \frac{1}{\sqrt{8\pi t}} \right],$$

and $D(x||y) = x \log\left(\frac{x}{y}\right) + (1-x) \log\left(\frac{1-x}{1-y}\right)$ is the Kullback-Leibler divergence.

Taking into account that $p_{true} \ll 1$ and $p_{false} \ll 1$, and denoting by ρ the ratio p_{false}/p_{true}, it can be shown that

$$\Pr[N_k \geq N_{true}] \approx (\eta\rho)^\tau e^{\tau(1-\eta\rho)} \frac{1}{\sqrt{2\pi\tau}} \left(\frac{1}{2} + \frac{1}{1-\eta\rho} \right) \approx \frac{3(e \cdot \eta\rho)^\tau}{\sqrt{8\pi\tau}},$$

as in most cases, $p_{false} \ll p_{true}/\eta$.

After the two distillation phases, the whole key is eventually recovered by a search phase which only examines the keys having their two 66-bit restrictions above the threshold. These keys can be easily found by merging the two lists of partial keys which share the same value on the following 4 bits: $K_0^{48}, K_0^{32}, K_0^{16}$ and $(K_0^0 \oplus K_0^{63})$.

The complexity, Time_2, of the search phase corresponds to one encryption under each key whose score reaches the threshold in both distillation phases. There are approximately 2×2^{62} wrong keys that collide with the right key on all the 66 key bits involved in one of the distillation phases, therefore

$$\mathsf{Time}_2 \approx 2^{128} \left(\frac{3(e \cdot \eta\rho)^\tau}{\sqrt{8\pi\tau}} \right)^2 + 2^{63} \frac{3(e \cdot \eta\rho)^\tau}{\sqrt{8\pi\tau}} \approx 2^{125} \frac{9(e \cdot \eta\rho)^{2\tau}}{\pi\tau}.$$

Overall Complexity. Taking into account the complexity of distillation and search phases, we get

$$\mathsf{Data} \times (\mathsf{Time}_1 + \mathsf{Time}_2) = \frac{8N_{true}^2 D}{p_{true}^2} + \frac{9 \times 2^{127} N_{true}}{\pi \cdot p_{true} \cdot \tau} \left(\frac{e \cdot p_{false} \cdot N_{true}}{p_{true} \cdot \tau} \right)^{2\tau}.$$

Memory. The distillation phase is the only phase having a large memory requirement, thus the memory complexity is

$$\mathsf{Memory} = 4D \frac{N_{true}}{p_{true}} = D \cdot \mathsf{Data}.$$

4.6 Success Probability of the Attack

We now estimate the success probability of our attack. For each distillation phase, there are $t = 2^{63} p_{true}$ right pairs, that might follow one of the differentials in Σ. Provided t does not exceed 2^{16}, these pairs all belong to different structures with a high probability. Therefore, t is a good estimation of the number of structures (out of the 2^{32} possible choices) that contain at least one valid pair for each of the distillation phases.

During the attack, the adversary chooses N_s structures out of 2^{32} in both phases, and succeeds if each set of N_s structures contains at least τ right pairs. The probability that a set of N_s structures contains exactly u right pairs can be approximated by the probability that it contains exactly u structures chosen among those containing a right pair. By summing for $u \geq \tau$, and assuming independence of the two phases, this leads to the following estimation of the probability of success:

$$P(N_s, \tau) = \left(\sum_{u \geq \tau} \frac{\binom{t}{u}\binom{2^{32}-t}{N_s-u}}{\binom{2^{32}}{N_s}} \right)^2 = \left(\sum_{u \geq \tau} \frac{\binom{N_s}{u}\binom{2^{32}-N_s}{t-u}}{\binom{2^{32}}{t}} \right)^2. \tag{3}$$

5 Experimental Estimation of p_{true}

In this section we display some experimental results that validate our attack strategy. Indeed, the complexity evaluation of our attack heavily relies on the estimation of the value of p_{true}. However, this estimation highly relies on the assumption that the differential transitions between the rounds are independent. In the case of PRINCE, as the round keys are all identical, this assumption is questionable. Therefore we have validated our approach by the following experiments.

In the next section, we show that, based on theoretical estimations of p_{true} and of the time and memory complexities, the best choice of parameters for attacking 10 rounds of PRINCE are $N_{true} = 6$ and $D = 12$. This attack then makes use of 24 differentials over 6 rounds, 12 in each distillation phase. Each of

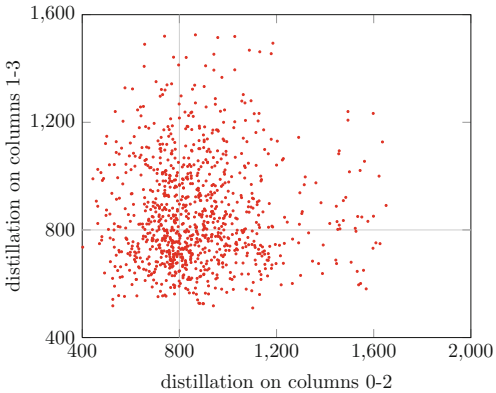


Fig. 5. Number of message pairs following one of the 16 differentials for both distillation phases for the 1000 keys tested

these differentials have been determined by aggregating several differential paths with a high probability over 6 rounds.

We have implemented an algorithm that computes exhaustively the pairs of messages which follow one of these differential paths, given one value of the key. As a result, we obtain a lower bound on the number of pairs that follow one of the 12 differentials for each distillation phase.

We ran our program on 1000 randomly chosen keys. The results are depicted on Fig. 5, where the x-coordinate (resp. the y-coordinate) represents the number of pairs following one of the differentials involved in the first (resp. the second) distillation phase. Following our theoretical estimation, the average number of pairs should be $\sum 2^{63} p_{true}(\Delta) = p_{true} 2^{63} = 800$ for each phase. In our experiments we observe an average value of 869 pairs for the first phase and 861 pairs for the second phase. This tends to show that the differentials we have identified are followed with probabilities slightly higher than estimated.

6 Results

In this section we apply the previously described attack to several variants of PRINCE. Using the study of the differential properties of PRINCE displayed in Sect. 3, we still have to select some parameters for our attack, namely D (the number of differentials we take into account), τ (the threshold score for key candidates), and N_{true} (the estimated score of the right keys). By choosing $\tau = N_{true}$, the right key reaches τ in each distillation phase with probability close to 0.5, leading to an almost constant success probability of 0.25.

Then, our goal is to find the values of D and N_{true} that minimize the product $\mathsf{Comp} = \mathsf{Data} \times (\mathsf{Time}_1 + \mathsf{Time}_2)$.

The following table summarizes our most significant results. We focus on 9-round and 10-round versions of PRINCE with the original Sbox. We also

emphasize that some members of the PRINCE-family are more vulnerable by presenting results obtained with the modified Sbox defined in Sect. 3. It is worth noticing that, in all cases, p_{true} is much higher than the false alarm probability $p_{false} = 2^{-64}D$.

Cipher	Rounds	D	p_{true}	N_{true}	Data	Time_1	Time_2	Comp
PRINCE original Sbox	9	40	$2^{-43.30}$	3	$2^{46.89}$	$2^{51.21}$	$2^{41.34}$	$2^{98.10}$
PRINCE original Sbox	10	12	$2^{-53.36}$	6	$2^{57.94}$	$2^{60.53}$	$2^{56.54}$	$2^{118.56}$
PRINCE modified Sbox	10	12	$2^{-46.83}$	3	$2^{50.42}$	2^{53}	$2^{52.08}$	$2^{104.03}$
PRINCE modified Sbox	11	12	$2^{-54.81}$	8	$2^{59.81}$	$2^{62.40}$	$2^{56.92}$	$2^{122.24}$

Our choice of parameters has been optimized for $\tau = N_{true}$. If we increase the amount of data, keeping the same threshold value would highly increase the complexity of the search phase. A better strategy may then consist in increasing the threshold but keeping it less than N_{true}. In this case, we increase the success probability of the attack. For example, with the previous parameters, if the amount of available data is $\mathsf{Data} = 2^{59.7}$ (instead of $2^{57.94}$), N_{true} increases from 6 to 20. Then, choosing $\tau = 7$ leads to an overall complexity of $\mathsf{Comp} = 2^{120}$, and a success probability of 0.85 as given by Eq. (3).

References

1. Abed, F., List, E., Lucks, S.: On the Security of the Core of PRINCE Against Biclique and Differential Cryptanalysis. IACR Cryptology ePrint Archive, Report 2012/712 (2012). http://eprint.iacr.org/2012/712
2. Blondeau, C., Gérard, B.: Multiple differential cryptanalysis: theory and practice. In: Joux, A. (ed.) FSE 2011. LNCS, vol. 6733, pp. 35–54. Springer, Heidelberg (2011)
3. Blondeau, C., Gérard, B., Nyberg, K.: Multiple differential cryptanalysis using LLR and χ^2 statistics. In: Visconti, I., De Prisco, R. (eds.) SCN 2012. LNCS, vol. 7485, pp. 343–360. Springer, Heidelberg (2012)
4. Bogdanov, A.A., Knudsen, L.R., Leander, G., Paar, C., Poschmann, A., Robshaw, M., Seurin, Y., Vikkelsoe, C.: PRESENT: an ultra-lightweight block cipher. In: Paillier, P., Verbauwhede, I. (eds.) CHES 2007. LNCS, vol. 4727, pp. 450–466. Springer, Heidelberg (2007)
5. Borghoff, J., Canteaut, A., Güneysu, T., Kavun, E.B., Knezevic, M., Knudsen, L.R., Leander, G., Nikov, V., Paar, C., Rechberger, C., Rombouts, P., Thomsen, S.S., Yalçın, T.: PRINCE – a low-latency block cipher for pervasive computing applications. In: Wang, X., Sako, K. (eds.) ASIACRYPT 2012. LNCS, vol. 7658, pp. 208–225. Springer, Heidelberg (2012)
6. Borghoff, J., Canteaut, A., Güneysu, T., Kavun, E.B., Knezevic, M., Knudsen, L.R., Leander, G., Nikov, V., Paar, C., Rechberger, C., Rombouts, P., Thomsen, S.S., Yalçin, T.: PRINCE - a low-latency block cipher for pervasive computing applications (full version). IACR Cryptology ePrint Archive, Report 2012/529 (2012). http://eprint.iacr.org/2012/529

7. Canteaut, A., Fuhr, T., Gilbert, H., Naya-Plasencia, M., Reinhard, J.-R.: Multiple differential cryptanalysis of round-reduced PRINCE (Full version). IACR Cryptology ePrint Archive, Report 2014/089 (2014). http://eprint.iacr.org/2014/089
8. Canteaut, A., Naya-Plasencia, M., Vayssière, B.: Sieve-in-the-middle: Improved MITM attacks. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013, Part I. LNCS, vol. 8042, pp. 222–240. Springer, Heidelberg (2013)
9. Fouque, P. -A., Joux, A., Mavromati, C.: Multi-user collisions: Applications to Discrete Logs, Even-Mansour and Prince. IACR Cryptology ePrint Archive, Report 2013/761 (2013). http://eprint.iacr.org/2013/761
10. Gilbert, H., Peyrin, T.: Super-Sbox Cryptanalysis: improved attacks for AES-like permutations. In: Hong, S., Iwata, T. (eds.) FSE 2010. LNCS, vol. 6147, pp. 365–383. Springer, Heidelberg (2010)
11. Gong, Z., Nikova, S., Law, Y.W.: KLEIN: a new family of lightweight block ciphers. In: Juels, A., Paar, C. (eds.) RFIDSec 2011. LNCS, vol. 7055, pp. 1–18. Springer, Heidelberg (2012)
12. Guo, J., Peyrin, T., Poschmann, A., Robshaw, M.: The LED block cipher. In: Preneel, B., Takagi, T. (eds.) CHES 2011. LNCS, vol. 6917, pp. 326–341. Springer, Heidelberg (2011)
13. Jean, J., Nikolić, I., Peyrin, T., Wang, L., Wu, S.: Security analysis of PRINCE. In: Moriai, S. (ed.) FSE 2013. LNCS, vol. 8424, pp. 92–111. Springer, Heidelberg (2014)
14. Li, L., Jia, K., Wang, X.: Improved meet-in-the-middle attacks on AES-192 and PRINCE. IACR Cryptology ePrint Archive, Report 2013/573 (2013)
15. Minier, M., Gilbert, H.: Stochastic cryptanalysis of Crypton. In: Schneier, B. (ed.) FSE 2000. LNCS, vol. 1978, pp. 121–133. Springer, Heidelberg (2000)
16. Shirai, T., Shibutani, K., Akishita, T., Moriai, S., Iwata, T.: The 128-bit blockcipher CLEFIA (Extended Abstract). In: Biryukov, A. (ed.) FSE 2007. LNCS, vol. 4593, pp. 181–195. Springer, Heidelberg (2007)
17. Soleimany, H., Blondeau, C., Yu, X., Wu, W., Nyberg, K., Zhang, H., Zhang, L., Wang, Y.: Reflection cryptanalysis of PRINCE-like ciphers. In: Moriai, S. (ed.) FSE 2013. LNCS, vol. 8424, pp. 71–91. Springer, Heidelberg (2013)
18. Suzaki, T., Minematsu, K., Morioka, S., Kobayashi, E.: TWINE: a lightweight block cipher for multiple platforms. In: Knudsen, L.R., Wu, H. (eds.) SAC 2012. LNCS, vol. 7707, pp. 339–354. Springer, Heidelberg (2012)
19. Wu, W., Zhang, L.: LBlock: a lightweight block cipher. In: Lopez, J., Tsudik, G. (eds.) ACNS 2011. LNCS, vol. 6715, pp. 327–344. Springer, Heidelberg (2011)

Advanced Constructions

Efficient Fuzzy Search on Encrypted Data

Alexandra Boldyreva[1(✉)] and Nathan Chenette[2]

[1] Georgia Institute of Technology, Atlanta, Georgia
sasha@gatech.edu
[2] Clemson University, Clemson, USA
nchenet@clemson.edu

Abstract. We study the problem of efficient (sub-linear) fuzzy search on encrypted outsourced data, in the symmetric-key setting. In particular, a user who stores encrypted data on a remote untrusted server forms queries that enable the server to efficiently locate the records containing the requested keywords, even though the user may misspell keywords or provide noisy data in the query. We define an appropriate primitive, for a general *closeness* function on the message space, that we call *efficiently fuzzy-searchable encryption* (*EFSE*). Next we identify an optimal security notion for EFSE. We demonstrate that existing schemes do not meet our security definition and propose a new scheme that we prove secure under basic assumptions. Unfortunately, the scheme requires large ciphertext length, but we show that, in a sense, this space-inefficiency is unavoidable for a general, optimally-secure scheme. Seeking the right balance between efficiency and security, we then show how to construct schemes that are more efficient and satisfy a weaker security notion that we propose. To illustrate, we present and analyze a more space-efficient scheme for supporting fuzzy search on biometric data that achieves the weaker notion.

1 Introduction

Motivation and Related Work. Cloud storage, which is a remote storage accessed over a network, has moved from hype to reality and is currently experiencing explosive growth. One of the major challenges in cloud storage adoption is providing security against the untrusted server without compromising functionality and efficiency. Numerous works have addressed the problem of symmetric searchable encryption in recent years, e.g. [2,9,12,13,23]. The solutions differ in the level of security and efficiency they provide, however most of them only support exact-match queries.

These solutions, however, are not suitable for practical situations where queried keywords differ slightly from those corresponding to stored encrypted

A. Boldyreva—Supported by NSF: CNS-0831184 and CNS-1318511 awards.
N. Chenette—Part of the work done while at the Georgia Institute of Technology. Partially supported by NSF: CNS-0831184 grant of the first author.

C. Cid and C. Rechberger (Eds.): FSE 2014, LNCS 8540, pp. 613–633, 2015.
DOI: 10.1007/978-3-662-46706-0_31

data. A user can use different spellings over time, such as "1 800 555-66-77" and "1(800)555 66 77". Google queries can tolerate typos, but such functionality is much more challenging to support when the data is encrypted. Moreover, data can be inherently noisy, e.g. for biometric identification: investigators querying a criminal database using data from a crime scene should allow for "fuzziness" in fingerprint readings and witness description of the suspect. In this work we consider the problem of efficient (sub-linear) search on encrypted data that supports fuzzy search queries. Sub-linear and, in particular, logarithmic-time search is essential because a linear scan of the whole data is unacceptable for any application dealing with large databases. Typically, this requirement for efficient search is irreconcilable with achieving a conventional "strong" security notion. But practitioners are willing to compromise security for functionality and thus it is important to identify suitable (possibly "weak") levels of security and provide provably-secure solutions.

Several recent papers pertain to fuzzy searchable encryption. The scheme from [18] is designed to address the general problem, though it lacks formal security analysis and we later show that, in spite of being space-inefficient, its security is not strong enough. The construction from [1], as well as the related schemes for the public-key setting [10,11] and the recent work [16] for the symmetric-key setting require the user to know all the data in advance, analyze the entire data and pre-compute the index before data outsourcing. This requirement is unsuitable for many broad applications, such as when data is frequently updated or streaming. The paper [25] motivates and discusses the problem of fuzzy search, but does not provide any solutions. Fully homomorphic encryption [14,24] could be used to implement fuzzy search queries; however, even a (future) computationally efficient FHE scheme would require search time linear in the length of the database. Hence the task of finding a provably-secure efficient (sub-linear) fuzzy encryption scheme supporting on-the-fly encryption has been open prior to our work.

The major contribution of this work is to initiate the study of a highly relevant problem, efficient fuzzy encryption, from a cryptographic (provable-security) standpoint. It should be viewed as a "first step" in this effort and should not be considered a complete treatment of the subject, which has strong possibilities for future directions of research. Nevertheless, our work provides the foundations for the study of the subject, including basic definitions, impossibility results, and basic schemes. Our work continues a line of recent research on studying encryption schemes providing more functionality while satisfying weaker security notions, such as deterministic, order-preserving, format-preserving, property-preserving, predicate, and functional encryption [3,6,9,15,17,21].

We now give an overview of our results.

Defining closeness. To even define our problem, we first need to establish what "close" means for messages. At its core, closeness is a function assigning a value ("close", "far," or "near") to any pair of messages from a space. Thus, we introduce the concept of a *closeness domain* which consists of a domain along with a closeness function.

Efficiently fuzzy-searchable encryption and its security. Next we define the central primitive, *efficiently fuzzy-searchable encryption (EFSE)*, defined on a closeness domain. In addition to the standard functions of a symmetric encryption scheme, an EFSE scheme should provide a public function that takes a ciphertext and returns all ciphertexts in a database that are equal or close to (but none that are far from) the queried ciphertext. We also allow for optional false-positives, i.e. the function may return ciphertexts of some near messages. Furthermore, this function should be sub-linearly efficient. We then discuss the details of how a user and the server perform search using an EFSE scheme. We note that an EFSE scheme leaks equality and "closeness" of queried messages in order to provide efficient exact-match and fuzzy search. Thus, an optimal security notion for EFSE would be a natural relaxation of the standard IND-CPA security definition prohibiting queries that trivially exploit this leakage of closeness and equality—we call this optimal security *indistinguishability under same-closeness-pattern chosen-plaintext attacks* (IND-CLS-CPA) and define it formally.

Template EFSE construction and its security. For generality and convenience, we propose a general template EFSE construction providing the basis of all specific EFSE constructions that we discuss later. The template construction, which is inspired by the scheme from [18], formalizes and extends their construction by building an EFSE scheme from three elements, listed with security notions as follows.

1. An *efficient searchable encryption* (ESE) scheme, which was defined in [2] and is essentially a symmetric encryption that leaks equality, and is thus is a generalization of deterministic encryption; the relevant security notion is *indistinguishability under distinct chosen-plaintext attack* or IND-DCPA [4].
2. A *closeness-preserving tagging function* that maps domain elements to "tags" so that only close messages map to overlapping tags; the relevant security condition is called *consistency*.
3. A *batch-encoding family*, each instance of which maps batches of elements according to a deterministic function from domain to range; the relevant security notion is *privacy-preserving under chosen batch attacks* (PP-CBA) and is related to IND-DCPA.

Note that the latter two primitives and their security notions are novel.

The template scheme works as follows: a ciphertext contains an ESE-encryption of the message, as well as a batch-encoding of all of the message's "tags," as defined by the closeness-preserving tagging function. The ESE-encryption leaks equality, and the batch-encoded tags leak closeness. We show that a scheme based on the template is secure if the ESE scheme is IND-DCPA-secure, the batch-encoding family is collision-free and PP-CBA-secure, and the tagging function is consistent. We also suggest how to instantiate an IND-DCPA-secure ESE scheme and a PP-CBA-secure batch-encoding family out of blockciphers for use in constructions, leaving the remaining task of finding a consistent tagging function (discussed later, individually for each particular scheme.)

Analysis of scheme from [18]. Next, we present the first cryptographic security analysis of the scheme from [18] (which was missing a formal definition of security and proof.) We first define a scheme based on our template construction that is essentially equivalent, in that the scheme's core component is a tagging function that for a message outputs its "neighbors," i.e. the other messages in the message space that are close to a message. However, this tagging function is *not* consistent in general, which means that this construction is not IND-CLS-CPA-secure in general: to prove this, we present a simple efficient adversary with high advantage. The attack exploits a simple observation that looking at two encoded tags one can with high probability tell *how many* neighbors the associated messages share. Leaking such information is not required for the functionality of EFSE and hence is a security breach according to our definition. We also note that the scheme from [18], besides being IND-CLS-CPA-insecure, is not very efficient in terms of ciphertext length. The constructions we propose target either strong security with the same efficiency, or much improved efficiency (with a necessarily weaker security guarantee.)

New optimally-secure construction. We propose a new general EFSE scheme. It relies on the notion of the *closeness graph*, whose vertices are the unique elements of the message space, and edges indicate closeness between elements. Defined according to the template model, the tagging function for this scheme sends a message to its set of incident edges (rather than neighboring vertices á la [18]) in the closeness graph. This tagging function is consistent, and so the scheme is IND-CLS-CPA-secure assuming the other components of the scheme satisfy the appropriate security notions.

One might worry that our construction is rather inefficient in the ciphertext length, which is linear in the maximum degree of the closeness graph. However we show that an EFSE scheme that works on general closeness domains (i.e. the scheme's algorithms do not depend on the structure of the closeness domain) must, in fact, require ciphertext length linear in the maximum degree of the closeness graph. The argument is information theoretic and relies on the functionality, rather than security, of the primitive. Thus, in achieving EFSE on arbitrarily-defined closeness domains the new IND-CLS-CPA-secure construction is (asymptotically) space-optimal, and moreover optimally secure.

Constructions with improved efficiency. In many (even most?) practical applications, vertices of the closeness graph have massive degrees. Degrees can even be infinite, e.g. on continuous spaces—consider, for example, searching a massive database of website access-records for one that accessed a webpage at approximately 6:59:59.95 PM on May 20, 2012 (where the time query must be fuzzy to account for inherent lag-time in the network)—here, depending on the granularity of measurements and the closeness tolerance, there could be a huge number of neighbors. This situation can grow even worse for multi-dimensional spaces, as the number of "close neighbors" increases exponentially with dimensionality for closeness defined on a metric. Consider, for example, querying a criminal database with a large array of biometric measurements taken from a crime scene, in an attempt to find suspects—here, multi-dimensional closeness

(closeness in every measurement) is needed, and if there are (say) a few dozen measurements, and even a narrow definition of closeness in each, the number of neighbors could again be huge. In such situations our optimally-secure scheme, as well as the less-secure scheme from [18], are unacceptably inefficient—and the aforementioned lower bound result shows that we cannot expect to do better for arbitrary domains.

We seek the right balance between the desired efficiency and security of EFSE, and look at closeness domains with a well-defined structure. We argue that IND-CLS-CPA-security is too strong to be useful in characterizing EFSEs on "non-rigid" closeness domains (where near messages could be encrypted to either close or far ciphertexts), and so to do this we introduce a new security definition. The new definition requires schemes to hide all information about plaintexts except nearness and a certain aspect of "local structure"—essentially, messages' offsets from a predetermined fixed regular lattice $\mathcal{L}$ on the space. Importantly, this implies that no major relative information (i.e., nothing above the least-significant-bit level) is leaked about a pair of "disconnected messages," that is, messages that cannot be connected through a chain of near known corresponding ciphertext pairs. Hence, we call this notion *macrostructure-security*. Note that this security may be useful in applications such as the website access-record and biometric matching examples above, where it is not a big deal to reveal aspects of local structure (does it matter if an adversary knows, say, the least significant bits relating to biometric measurements in the criminal database?) but it is important to hide large differences between messages.

Our security definition and construction strategy focus on a practical choice of domains with associated metric and close, near, and far distance thresholds, that we call *metric closeness domains*; in particular, we consider real multidimensional space. Critically, on these domains, closeness is defined in a "regular" manner across the space—namely, for any regular lattice in the space, closeness is invariant under translation by a lattice vector. The security definition is then defined in terms of a fixed lattice, demanding that nothing is leaked except "local structure" of near clusters of messages with respect to the lattice. To provide a blueprint for building specific schemes, we introduce the concept of an "anchor radius" for a metric closeness domain and a lattice, and use it to construct a tagging function to build an EFSE via our usual template. We show that a valid anchor radius implies an EFSE construction that is macrostructure-secure. Then, to enhance understanding, we present a practical example, filling in details of the blueprint to build a (relatively) space-efficient, macrostructure-secure EFSE scheme supporting fuzzy search on fingerprint data. Finally, in the full version [8], we observe that an efficient scheme that probabilistically acts like an EFSE scheme can be constructed out of locality-sensitive hash (LSH) functions. But the theory behind these schemes and their security is beyond the scope of this work.

Future work. Our work provides the basis for cryptographic study of fuzzy-searchable encryption. Our template constructions invite exploration of more efficient schemes that will automatically satisfy our security notions. In addition, future studies might achieve more efficient and secure schemes—circumventing

our impossibility result by defining closeness and EFSE primitives in a different manner. For instance, one could consider only closeness domains with certain natural structure, or closeness could be defined quantitatively or probabilistically.

2 Preliminaries

We let $\mathcal{LR}$ (left-or-right) denote the "selector" that on input m_0, m_1, b returns m_b. For $x \in \mathbb{Z}$, the notation $[x]$ denotes the set $\{1, 2, \ldots, x\}$. In some of the algorithm descriptions, for ease and clarity of analysis, we use abstract set notation. In a practical implementation, the sets can be implemented by some specialized data structure, or by vectors/lists with a common predetermined order (e.g., numerical order.) We recall the syntax and security for symmetric encryption in the full version of the paper [8]. We wait until Sect. 4 to define efficiently searchable encryption, privacy-preserving batch-encoding, and closeness-preserving tagging functions. Here, we introduce a metric space, closeness domains and associated graph-theoretical concepts.

Metric spaces. $(\mathcal{D}, d)$ is a *metric space* if $\mathcal{D}$ is a set and d (the *metric*) is a real-valued function on $\mathcal{D} \times \mathcal{D}$ such that for all $x, y, z \in \mathcal{D}$,

$$d(x,y) \geq 0 \qquad d(x,y) = 0 \text{ iff } x = y$$
$$d(x,y) = d(y,x) \qquad d(x,z) \leq d(x,y) + d(y,z).$$

Closeness domain. We refer to the pair $\Lambda = (\mathcal{D}, \mathrm{Cl})$ as a *closeness domain* if

1. $\mathcal{D}$ is a (finite or infinite) set, called the *domain* or *message space*;
2. Cl is the *closeness function* that takes a pair of messages and outputs a member of $\{\mathtt{eq}, \mathtt{close}, \mathtt{near}, \mathtt{far}\}$, so that Cl is symmetric (i.e., $\mathrm{Cl}(m, m') = \mathrm{Cl}(m', m)$ for all $m, m' \in \mathcal{D}$) and $\mathrm{Cl}(m, m') = \mathtt{eq}$ if and only if $m = m'$.

According to the output of Cl, we say a pair of messages is *equal*, *close*, *near*, or *far*. Note that a closeness domain can be defined by describing which distinct message pairs of a domain $\mathcal{D}$ are close and which are far (the rest are then near.) For convenience, we say Λ is *rigid* if $\mathrm{Cl}(m, m') \in \{\mathtt{close}, \mathtt{far}\}$ for all $m \neq m' \in \mathcal{D}$. When these quantities exist, the *degree* of a message m in Λ is $\Delta_m = |\{m' \in \mathcal{D} \mid \mathrm{Cl}(m, m') = \mathtt{close}\}|$, and the *max degree* of Λ is $\Delta = max_{m \in \mathcal{D}} \Delta_m$.

As a special case, let d be a metric[1] on domain $\mathcal{D}$, and let $\delta > 0$. The *metric* closeness domain $\left(\mathcal{D}, \mathcal{M}_d^{\delta^{\mathtt{C}}, \delta^{\mathtt{F}}}\right)$ on domain $\mathcal{D}$ with respect to metric d, *close threshold* $\delta^{\mathtt{C}} \geq 0$, and *far threshold* $\delta^{\mathtt{F}} \geq \delta^{\mathtt{C}}$, has the following closeness function: for distinct $m, m' \in \mathcal{D}$, $\mathcal{M}_d^{\delta^{\mathtt{C}}, \delta^{\mathtt{F}}} = \begin{cases} \mathtt{close} & \text{if } d(m,m') \leq \delta^{\mathtt{C}}; \\ \mathtt{far} & \text{if } d(m,m') > \delta^{\mathtt{F}}. \end{cases}$ For instance,

[1] So in particular, d obeys the triangle inequality.

$\left(\{0,1\}^{80}, \mathcal{M}^{1,2}_{\text{Ham}}\right)$, where Ham is Hamming distance, is a closeness domain of all length-80 strings where strings differing in 1 bit are close, differing in 2 bits are near, and differing in more than 2 bits are far.

Closeness and nearness graph, induced subgraph. Let $\Lambda = (\mathcal{D}, \text{Cl})$ be a closeness domain, $\mathcal{V}_\Lambda = \mathcal{D}$ and

$$\mathcal{E}^{\mathtt{C}}_\Lambda = \{\{u,v\} \mid u \neq v \in \mathcal{V}_\Lambda \text{ and } \text{Cl}(u,v) = \mathtt{close}\};$$
$$\mathcal{E}^{\mathtt{N}}_\Lambda = \{\{u,v\} \mid u \neq v \in \mathcal{V}_\Lambda \text{ and } \text{Cl}(u,v) \in \{\mathtt{close}, \mathtt{near}\}\}.$$

Then $\mathcal{G}^{\mathtt{C}}_\Lambda = (\mathcal{D}, \mathcal{E}^{\mathtt{C}}_\Lambda)$ is the *closeness graph* and $\mathcal{G}^{\mathtt{N}}_\Lambda = (\mathcal{D}, \mathcal{E}^{\mathtt{N}}_\Lambda)$ is the *nearness graph* of Λ. For graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ and $H \subseteq \mathcal{V}$ let $\mathcal{G}(H) = (H, \mathcal{E}(H))$ be the *subgraph induced by* H where $\mathcal{E}(H) = \{\{u,v\} \in \mathcal{E} \mid u, v \in H\}$.

3 Efficiently Fuzzy-Searchable Symmetric Encryption

We now define our main primitive and show how can it be used for efficient search. Following that, we formulate the optimal level of security for EFSE schemes.

Defining Efficiently Fuzzy-Searchable Encryption. $\mathtt{FSE} = (\mathcal{K}, \mathcal{E}nc, \mathcal{D}ec, \mathtt{makeDS}, \mathtt{fuzzyQ})$ is a *structured fuzzy-searchable symmetric encryption* (StructFSE) scheme on closeness domain $\Lambda = (\mathcal{D}, \text{Cl})$ if $(\mathcal{K}, \mathcal{E}nc, \mathcal{D}ec)$ is a symmetric encryption scheme on $\mathcal{D}$, and for any key K output by $\mathcal{K}$,

- $\mathtt{makeDS}$ takes a set of ciphertexts $\mathbf{C}$ (the *database*) encrypted under K and outputs a data structure $\mathtt{DS}_{\mathbf{C}}$;
- $\mathtt{fuzzyQ}$, given database $\mathbf{C}$, data structure $\mathtt{DS}_{\mathbf{C}}$, and query ciphertext c, outputs two subsets $\mathbf{E}, \mathbf{F}$ of $\mathbf{C}$ such that

$$\mathbf{E} = \mathbf{C}_{\mathtt{eq}}(c) \qquad \text{and} \qquad \mathbf{C}_{\mathtt{close}}(c) \subseteq \mathbf{F} \subseteq \mathbf{C}_{\mathtt{near}}(c),$$

where for $m = \mathcal{D}ec(K, c)$, $m' = \mathcal{D}ec(K, c')$,

$$\begin{cases} \mathbf{C}_{\mathtt{eq}}(c) & = \{c' \in \mathbf{C} \mid \text{Cl}(m, m') = \mathtt{eq}\} \\ \mathbf{C}_{\mathtt{close}}(c) & = \{c' \in \mathbf{C} \mid \text{Cl}(m, m') = \mathtt{close}\}. \\ \mathbf{C}_{\mathtt{near}}(c) & = \{c' \in \mathbf{C} \mid \text{Cl}(m, m') \in \{\mathtt{close}, \mathtt{near}\}\}. \end{cases}$$

One could easily relax the above syntax to not require the returned ciphertexts to equal those from the database. This would allow one to consider, for example, schemes based on homomorphic encryption. We stick with a stricter definition for simplicity. To ease discussion, for implicit fixed key K we say that ciphertexts c and c' are close (respectively, far) if their decryptions $m = \mathcal{D}ec(K, c)$ and $m' = \mathcal{D}ec(K, c')$ are close (far). Notice that in a StructFSE scheme, $\mathtt{fuzzyQ}(\mathbf{C}, \mathtt{DS}_{\mathbf{C}}, c)$ returns all ciphertexts in $\mathbf{C}$ close to c and no ciphertexts far from c. Any near ciphertext may be returned as well—these can be thought of as "legal false positives" in a fuzzy search query. In this sense, FSE on a rigid closeness domain

cannot have any false positives. But of course, even on a non-rigid domain, we must limit false positives to ensure efficiency.

We say StructFSE scheme $\mathtt{FSE} = (\mathcal{K}, \mathcal{E}nc, \mathcal{D}ec, \mathtt{makeDS}, \mathtt{fuzzyQ})$ is an *efficiently fuzzy searchable symmetric encryption* (EFSE) scheme if for any (sufficiently large) database $\mathbf{C}$, data structure $\mathtt{DS}_{\mathbf{C}}$, key K generated by $\mathcal{K}$, and query ciphertext c with $|\mathbf{C}_{\mathtt{close}}(c)|$ sub-linear in the size of $\mathbf{C}$, the running time of $\mathtt{fuzzyQ}_{\mathbf{C},\mathtt{DS}_{\mathbf{C}}}(c)$ is sub-linear in the size of $\mathbf{C}$. Notice this condition on the running time limits the number of false positives for a fuzzy query.

We note that EFSE defined for rigid domains makes a special case of property-preserving encryption from [21] (for the property of "closeness"), but the general case of EFSE does not seem to fit the class of schemes from [21].

Using an EFSE scheme. Let $\mathtt{FSE} = (\mathcal{K}, \mathcal{E}nc, \mathcal{D}ec, \mathtt{makeDS}, \mathtt{fuzzyQ})$ be an EFSE scheme and K a valid key. In a practical scenario, let $\mathbf{C}$ be the set of ciphertexts currently in an encrypted database, encrypted under K. The server runs $\mathtt{makeDS}(\mathbf{C})$ to create a data structure $\mathtt{DS}_{\mathbf{C}}$, and upon a new query $c = \mathcal{E}nc_K(m)$, runs $\mathtt{fuzzyQ}(\mathbf{C}, \mathtt{DS}_{\mathbf{C}}, c)$ and returns the results, $\mathbf{E}$ and $\mathbf{F}$, to the user. By correctness of the scheme, $\mathbf{F}$ consists of all ciphertexts in $\mathbf{C}$ whose messages are close to m, and no ciphertexts whose messages are far from m. Since the scheme is efficient, such a query will take time sub-linear in the size of the database $\mathbf{C}$ (assuming the number of close messages itself is also sub-linear in the size of $\mathbf{C}$.) Also note that the scheme supports efficient exact-match search through $\mathbf{E}$.

As a side note, in a practical implementation, additional functions (e.g. add, remove, edit) would be useful to efficiently update the data structure as the database changes. In our analysis, we are less focused on efficiency of the data structure maintenance, so for simplicity we just let the (possibly inefficient) function `makeDS` construct the data structure from the entire database. And we leave it as an interesting open problem for future work to extend and realize the primitive so that "closeness" be specified during encryption.

Finally, observe that the "difficult" part of building an EFSE scheme is ensuring that `fuzzyQ` is efficient. Thus, the construction of $\mathcal{E}nc$ might as well be designed with the efficiency of `fuzzyQ` in mind. In our constructions, as detailed in Sect. 4, ciphertexts outputted by $\mathcal{E}nc$ will contain "encoded tags" such that ciphertexts of close messages share a common encoded tag. Thus, indexing ciphertexts by encoded tags in an efficiently searchable data structure, like a binary search tree, leads to an efficient construction of `fuzzyQ`.

Optimal Security for EFSE schemes. We construct the following indistinguishability-based security definition, called IND-CLS-CPA[2], for analyzing the security of EFSE schemes. Intuitively, this notion is identical to IND-CPA with the additional condition that left-right queries have the same *closeness pattern* (in the second requirement below.)

Definition 1. *Let* $\mathtt{FSE}$ *be an EFSE scheme on closeness domain* $\Lambda = (\mathcal{D}, Cl)$. *For bit* $b \in \{0,1\}$ *and adversary* A, *let* $\mathbf{Exp}^{ind\text{-}cls\text{-}cpa\text{-}b}_{\mathtt{FSE}}(A)$ *be the standard*

[2] We do not study chosen-ciphertext security here as it can be achieved using the encrypt-then-MAC method [5].

IND-CPA experiment $\mathbf{Exp}^{ind\text{-}cpa\text{-}b}_{\mathtt{FSE}}(A)$ *recalled in Fig. 1, but with the following restriction: if* $(m_0^1, m_1^1), \ldots, (m_0^q, m_1^q)$ *are the queries A makes to its LR encryption oracle* $\mathcal{E}nc(K, \mathcal{LR}(\cdot,\cdot,b))$*, then*

1. $|m_0^i| = |m_1^i|$ *for all* $i \in [q]$;
2. *for all* $i, j \in [q]$, $Cl(m_0^i, m_0^j) = Cl(m_1^i, m_1^j)$.

For an adversary A, define its IND-CLS-CPA advantage *against* FSE *as*

$$\mathbf{Adv}^{ind\text{-}cls\text{-}cpa}_{\mathtt{FSE}}(A) = \Pr\left[\mathbf{Exp}^{ind\text{-}cls\text{-}cpa\text{-}1}_{\mathtt{FSE}}(A)=1\right] - \Pr\left[\mathbf{Exp}^{ind\text{-}cls\text{-}cpa\text{-}0}_{\mathtt{FSE}}(A)=1\right].$$

We say that FSE *is* indistinguishable under same-closeness-pattern chosen-plaintext attacks *(IND-CLS-CPA-secure) if the IND-CLS-CPA advantage of any adversary against* FSE *is small*[3,4].

Experiment $\mathbf{Exp}^{\text{ind-cpa-}b}_{\mathcal{SE}}(A)$
$K \xleftarrow{\$} \mathcal{K}$
$b' \xleftarrow{\$} A^{\mathcal{E}nc(K,\mathcal{LR}(\cdot,\cdot,b))}$
Return b' .

Fig. 1. The IND-CPA experiment.

It should be apparent that IND-CLS-CPA-security is optimal for EFSE schemes on *rigid* closeness domains: revealing equality/closeness patterns of LR-queries is unavoidable as an adversary can run the (public) `fuzzyQ` function on ciphertexts to test for equality and closeness. It may seem that the optimal security definition on *general* closeness domains, where `fuzzyQ` is given flexibility over near message pairs, should not allow distinguishing near messages as it is not needed for search functionality. However, while a stronger security definition than IND-CLS-CPA would be possible, the notion would necessarily depend on the scheme's construction, i.e., the left-right query restriction would rely on how `fuzzyQ` sends near message pairs to close or far ciphertexts. To define a security notion that is independent from the construction of `fuzzyQ`, the IND-CLS-CPA experiment forces left-right query pairs to match near-to-near, as `fuzzyQ` is permitted to distinguish near ciphertexts from close and far ciphertexts.

4 Template Tag-Encoding Construction for EFSE

In this somewhat technical section, we build up to a general construction of an EFSE scheme given a valid "tagging function" on the desired closeness domain. In addition, we show that under certain conditions, the scheme is IND-CLS-CPA-secure. First, though, we define several primitives, along with relevant security

[3] We use the informal term "small" because the main building blocks of symmetric cryptography, blockciphers, have keys of fixed length in practice. Thus, instead of requiring advantages to be negligible in a security parameter, we leave appropriate concrete bounds to be determined on a case-by-case basis depending on the application.

[4] According to our definitions, advantage can be negative; note that "small" refers to an advantage close to zero. For every adversary with negative advantage there is one with positive advantage, who just outputs the complement bit.

notions, that will be components of the construction. The primitives are: efficient searchable encryption (ESE) schemes [2], closeness-preserving tagging functions, and privacy-preserving batch-encoding families. We emphasize that, despite the technical language, these primitives are conceptually simple and can be instantiated in natural ways—the formalism is simply aimed to achieve fuller generality in isolating theoretical requirements from possible instantiations.

Efficient searchable encryption and security. The ESE scheme primitive [2] is recalled in the full version [8]. Intuitively, an ESE is an encryption scheme that "leaks equality," that is, there is a (public) way to tell if two ciphertexts are encryptions of the same message. In particular, deterministic functions F, G are provided such that if c_1 and c_2 are both encryptions of m under key K, $G(c_1) = F(K, m) = G(c_2)$ (and this is unlikely if c_1 and c_2 are encryptions of different messages.) The appropriate security notion for ESE was defined by [4] and is called *indistinguishability under distinct chosen plaintext attacks* (IND-DCPA)—it is also recalled in the full version [8]. The notion is identical to IND-CPA except that LR-queries must have the same "equality pattern" (and so avoiding the obvious attack, as ESE leaks equality.) Note that any PRF implies an IND-DCPA-secure ESE scheme [4] so there are many options for instantiation.

Closeness-preserving tagging functions. Fix a closeness domain $\Lambda = (\mathcal{D}, \mathrm{Cl})$. Let $\mathtt{TagUniv}$ be a (finite or infinite) set and let $\mathtt{Tags} : \mathcal{D} \to 2^{\mathtt{TagUniv}}$ be a function assigning a subset of $\mathtt{TagUniv}$ to every domain element. We call $\mathtt{Tags}$ a *closeness-preserving tagging function* (CPTF) from Λ into $\mathtt{TagUniv}$ if for every $x, y \in \mathcal{D}$ with $\mathrm{Cl}(x, y) = \mathtt{close}$, there exists $t \in \mathtt{TagUniv}$ such that $t \in \mathtt{Tags}(x) \cap \mathtt{Tags}(y)$; and for every $x, y \in \mathcal{D}$ with $\mathrm{Cl}(x, y) = \mathtt{far}$, $\mathtt{Tags}(x) \cap \mathtt{Tags}(y) = \emptyset$.

Further, a CPTF $\mathtt{Tags}$ is *consistent* with respect to closeness domain Λ if for any message sets $\{m_0^1, \ldots, m_0^q\}$ and $\{m_1^1, \ldots, m_1^q\}$ having the same closeness pattern[5], we have $\left|\bigcap_{i\in[q]} \mathtt{Tags}(m_0^i)\right| = \left|\bigcap_{i\in[q]} \mathtt{Tags}(m_1^i)\right|$. Consistency can be understood intuitively as follows: whenever a set of messages has the same closeness pattern as another set of messages, each set should have the same number of common tags.

Examples of CPTFs are integral to our constructions and several are introduced in the remainder of this paper.

Privacy-preserving batch-encoding. We say that $\mathcal{F} = (\mathcal{K}, \mathtt{En})$ is an *encoding family* on domain $\mathcal{D}$ and range $\mathcal{R}$ if $\mathcal{K}$ outputs random keys and $\mathtt{En}$ takes a key K and an element of $\mathcal{D}$ and outputs an element of $\mathcal{R}$ such that $\mathtt{En}(K, \cdot)$ is a (deterministic) function from $\mathcal{D}$ to $\mathcal{R}$. We further say that $\mathcal{F}_{\mathtt{Ben}} = (\mathcal{K}_{\mathtt{Ben}}, \mathtt{En}, \mathtt{Ben})$ is a *batch-encoding family* if $(\mathcal{K}_{\mathtt{Ben}}, \mathtt{En})$ is an encoding family and $\mathtt{Ben}$ takes a key K and a set of elements $M \subseteq \mathcal{D}$ and outputs $\{\mathtt{En}(K, m) \mid m \in M\}$. Given a function family $(\mathcal{K}', \mathtt{En}')$ it is easy to construct a batch-encoding family $(\mathcal{K}_{\mathtt{Ben}}, \mathtt{En}, \mathtt{Ben})$: let $\mathcal{K}_{\mathtt{Ben}} = \mathcal{K}'$ and $\mathtt{En} = \mathtt{En}'$, and define $\mathtt{Ben}(K, \cdot)$ to take a set of messages, run $\mathtt{En}(K, \cdot)$ on each, and return the set of results.

[5] That is, $\mathrm{Cl}(m_0^i, m_0^j) = \mathrm{Cl}(m_1^i, m_1^j)$ for all $i, j \in [q]$.

Experiment $\mathbf{Exp}^{\text{pp-cba-}b}_{\mathcal{F}_{\mathtt{Ben}}}(A)$

$K \xleftarrow{\$} \mathcal{K}_{\mathtt{Ben}}$

$b' \xleftarrow{\$} A^{\mathtt{Ben}(K,\mathcal{LR}(\cdot,\cdot,b))}$

Return b' ,

Fig. 2. The PP-CBA experiment.

We say that a encoding family $(\mathcal{K}_{\mathtt{Ben}}, \mathtt{En})$ or a batch-encoding family $(\mathcal{K}_{\mathtt{Ben}}, \mathtt{En}, \mathtt{Ben})$ is *collision-free* if for any key K, $\mathtt{En}(K, \cdot)$ is one-to-one on $\mathcal{D}$. Now, we define security for batch-encoding families. Called *privacy-preserving under chosen batch attacks*, it is essentially the IND-DCPA generalized to objects of the batch-encoding primitive.

Definition 2. *Let $\mathcal{F}_{\mathtt{Ben}} = (\mathcal{K}_{\mathtt{Ben}}, \mathtt{En}, \mathtt{Ben})$ be a batch-encoding family on domain $\mathcal{D}$ and range $\mathcal{R}$. For an adversary A and $b \in \{0,1\}$ consider the experiment defined in Fig. 2, where it is required that, if $(M_0^1, M_1^1), \ldots, (M_0^q, M_1^q)$ are the queries that A makes to its $\mathcal{LR}$-batch-encoding oracle (note: each M_j^i is a* set *of elements of $\mathcal{D}$), for all $I \subseteq [q]$ we have $\left|\bigcap_{i\in I} M_0^i\right| = \left|\bigcap_{i\in I} M_1^i\right|$.*

For an adversary A, define its PP-CBA advantage *against $\mathcal{F}_{\mathtt{Ben}}$ as*

$$\mathbf{Adv}^{\text{pp-cba}}_{\mathcal{F}_{\mathtt{Ben}}}(A) \;=\; \Pr\left[\mathbf{Exp}^{\text{pp-cba-1}}_{\mathcal{F}_{\mathtt{Ben}}}(A) = 1\right] - \Pr\left[\mathbf{Exp}^{\text{pp-cba-0}}_{\mathcal{F}_{\mathtt{Ben}}}(A) = 1\right].$$

We say that $\mathcal{F}_{\mathtt{Ben}}$ is privacy-preserving under chosen batch attacks *(PP-CBA-secure) if the PP-CBA advantage of any adversary against $\mathcal{F}_{\mathtt{Ben}}$ is small.*

Notice that the requirement rules out an obvious attack: suppose to the contrary that, without loss of generality, the adversary could query $(M_0^1, M_1^1), \ldots, (M_0^q, M_1^q)$ with $\left|\bigcap_{i\in[q]} M_0^i\right| > \left|\bigcap_{i\in[q]} M_1^i\right|$. If $\mathtt{En}(K, \cdot)$ is collision-free, $\left|\bigcap_{i\in[q]} \mathtt{Ben}(K, M_b^i)\right| = \left|\bigcap_{i\in[q]} \{\mathtt{En}(K, m) \mid m \in M_b^i\}\right| = \left|\bigcap_{i\in[q]} M_b^i\right|$, so by computing $\left|\bigcap_{i\in[q]} \mathtt{Ben}(K, M_b^i)\right|$ from the oracle responses the adversary can identify b.

On how to instantiate a privacy-preserving, collision-free batch-encoding scheme. Anticipating that our EFSE constructs will use PP-CBA-secure batch-encoding schemes, how can we construct one? In fact, a PP-CBA-secure batch-encoding scheme can be created straightforwardly out of a pseudorandom function (PRF), as we now demonstrate.

Let $\mathtt{PRF} = (\mathcal{K}_{\mathtt{PRF}}, \mathcal{F}_{\mathtt{PRF}})$ be a function family on domain $\mathcal{D}$ to some range $\mathcal{R}$. Let $\mathcal{F}_{\mathtt{Ben}} = (\mathcal{K}_{\mathtt{Ben}}, \mathtt{En}, \mathtt{Ben})$ where $\mathcal{K}_{\mathtt{Ben}} = \mathcal{K}_{\mathtt{PRF}}$, $\mathtt{En} = \mathcal{F}_{\mathtt{PRF}}$, and $\mathtt{Ben}$ is defined in the standard way using $\mathtt{En}$ as described above. We claim that if $\mathtt{PRF}$ is a PRF, then $\mathcal{F}_{\mathtt{Ben}}$ is PP-CBA-secure. See the following result, which is proved in [8].

Proposition 1. *For $\mathcal{F}_{\mathtt{Ben}}$ constructed as above out of function family $\mathtt{PRF}$, and any adversary A, there exist PRF adversaries F_0 and F_1 such that*

$$\mathbf{Adv}^{\text{pp-cba}}_{\mathcal{F}_{\mathtt{Ben}}}(A) = \mathbf{Adv}^{prf}_{\mathtt{PRF}}(F_0) + \mathbf{Adv}^{prf}_{\mathtt{PRF}}(F_1).$$

Further, if A submits queries of total length γ to its oracle, then F_1 and F_2 each submit queries of total length γ to their oracles as well.

$\mathtt{FSE}_{\mathtt{Etag}}[\mathtt{Tags}, \mathcal{F}_{\mathtt{Ben}}, \mathtt{ESE}] = (\mathcal{K}, \mathcal{E}nc, \mathcal{D}ec, \mathtt{makeDS}, \mathtt{fuzzyQ})$ where

- $\mathcal{K}$ runs $K_{\mathtt{Ben}} \xleftarrow{\$} \mathcal{K}_{\mathtt{Ben}}$ and $K_{\mathtt{ESE}} \xleftarrow{\$} \mathcal{K}_{\mathtt{ESE}}$, and returns $K_{\mathtt{Ben}} \| K_{\mathtt{ESE}}$.
- $\mathcal{E}nc(K_{\mathtt{Ben}} \| K_{\mathtt{ESE}}, m)$ runs $T_m \leftarrow \mathtt{Tags}(m)$; $\mathtt{Etags} \leftarrow \mathtt{Ben}(K_{\mathtt{Ben}}, T_m)$; $c_R \leftarrow \mathcal{E}nc_{\mathtt{ESE}}(K_{\mathtt{ESE}}, m)$, and returns $c \leftarrow \mathtt{Etags} \| c_R$.
- $\mathcal{D}ec(K_{\mathtt{Ben}} \| K_{\mathtt{ESE}}, c)$ parses c as $\mathtt{Etags} \| c_R$ and returns $\mathcal{D}ec_{\mathtt{ESE}}(K_{\mathtt{ESE}}, c_R)$.
- $\mathtt{makeDS}(\mathbf{C})$ initializes an efficient self-balancing search tree $\mathcal{T}$ representing an associative array from elements of $\mathcal{R}_{\mathtt{En}}$ to ciphertexts. For each ciphertext $c \in \mathbf{C}$ parsed as $c = \mathtt{Etags} \| c_R$, and for each $t \in \mathtt{Etags}$, add the node $(t \mapsto c)$ to $\mathcal{T}$. Output $\mathtt{DS}_{\mathbf{C}} \leftarrow \mathcal{T}$.
- $\mathtt{fuzzyQ}_{\mathbf{C},\mathtt{DS}_{\mathbf{C}}}(c)$ parses c as $\mathtt{Etags} \| c_R$ and interprets $\mathtt{DS}_{\mathbf{C}}$ as search tree $\mathcal{T}$. Let $\mathbf{E}, \mathbf{F} = \emptyset$. For each $t \in \mathtt{Etags}$, search $\mathcal{T}$ for nodes indexed by t; for any $(t \mapsto c')$ that exist, parse $c' = \mathtt{Etags}' \| c'_R$. Then, if $G(c_R) = G(c'_R)$, add c' to $\mathbf{E}$; otherwise, add c' to $\mathbf{F}$. Return $\mathbf{E}, \mathbf{F}$.

Fig. 3. General tag-encoding construction of a StructFSE scheme given $\mathtt{Tags}$, $\mathcal{F}_{\mathtt{Ben}}$, $\mathtt{ESE}$.

As will soon become clear, what we actually need is a PP-CBA-secure *collision-free* batch-encoding scheme, a natural extension of a IND-DCPA deterministic encryption scheme. To theoretically achieve collision resistance, a pseudorandom permutation (PRP) would be necessary. But concretely, statistical collision resistance should suffice—i.e. on random inputs, a collision occurs after $\sqrt{|\mathcal{R}|}$ inputs with probability approximately 1/2. We suggest using any blockcipher (permutation) that is a PRF (and thus PP-CBA-secure), though one may have to augment the blockcipher into a variable-input-length blockcipher [7] as described in [20], or into an encryption scheme like those of [2,22].

Template tag-encoding EFSE construction. We now provide a general "template" construction for an EFSE scheme given a closeness-preserving tagging function $\mathtt{Tags}$, batch-encoding family $\mathcal{F}_{\mathtt{Ben}}$, and ESE scheme $\mathtt{ESE}$. We remark that this template is a generalization of the technique used in [18], though we have expanded, formalized, and refined it significantly. All forthcoming EFSE constructions use this general construction as a template.

Let $\Lambda = (\mathcal{D}, \mathrm{Cl})$ be a closeness domain, $\mathtt{Tags}$ a function from $\mathcal{D}$ to subsets of a set $\mathtt{TagUniv}$, $\mathcal{F}_{\mathtt{Ben}} = (\mathcal{K}_{\mathtt{Ben}}, \mathtt{En}, \mathtt{Ben})$ a batch-encoding family on domain $\mathcal{D}_{\mathtt{En}} = \mathtt{TagUniv}$ and range $\mathcal{R}_{\mathtt{En}}$, and $\mathtt{ESE} = (\mathcal{K}_{\mathtt{ESE}}, \mathcal{E}nc_{\mathtt{ESE}}, \mathcal{D}ec_{\mathtt{ESE}}, F, G)$ an ESE scheme on $\mathcal{D}$. Then we define a general *tag-encoding* StructFSE scheme $\mathtt{FSE}_{\mathtt{Etag}}[\mathtt{Tags}, \mathcal{F}_{\mathtt{Ben}}, \mathtt{ESE}]$ in Fig. 3.

Conditions for correctness and efficiency. The following result, proved in [8], establishes conditions under which the template construction is a valid StructFSE scheme and when it is EFSE.

Theorem 1. *If $\mathcal{F}_{\mathtt{Ben}}$ is collision-free and $\mathtt{Tags}$ is closeness-preserving, then the scheme $\mathtt{FSE}_{\mathtt{Etag}}[\mathtt{Tags}, \mathcal{F}_{\mathtt{Ben}}, \mathtt{ESE}]$ is StructFSE. In addition, it is an EFSE scheme if $\mathtt{Tags}$, $\mathcal{F}_{\mathtt{Ben}}$, and $\mathtt{ESE}$ are efficient and $\mu = \max_m |\mathtt{Tags}(m)|$ is small.*

Conditions for optimal security. Now, fix a closeness domain $\Lambda = (\mathcal{D}, \mathrm{Cl})$, and let $\mathtt{Tags}$ be a CPTF from Λ into a set $\mathtt{TagUniv}$, $\mathcal{F}_{\mathtt{Ben}}$ a collision-free batch-encoding family on $\mathtt{TagUniv}$, and $\mathtt{ESE}$ an ESE scheme on $\mathcal{D}$, so that $\mathtt{FSE}_{\mathtt{Etag}}[\mathtt{Tags}, \mathcal{F}_{\mathtt{Ben}}, \mathtt{ESE}]$ is a valid StructFSE scheme by Theorem 1. The next result, proved in [8], gives sufficient conditions for $\mathtt{FSE}_{\mathtt{Etag}}[\mathtt{Tags}, \mathcal{F}_{\mathtt{Ben}}, \mathtt{ESE}]$ to be IND-CLS-CPA-secure.

Theorem 2. *If* $\mathtt{Tags}$ *is consistent with respect to* Λ, $\mu = \max_m |\mathtt{Tags}(m)|$ *is small,* $\mathcal{F}_{\mathtt{Ben}}$ *is PP-CBA-secure, and* $\mathtt{ESE}$ *is IND-DCPA-secure, then* $\mathtt{FSE}_{\mathtt{Etag}}[\mathtt{Tags}, \mathcal{F}_{\mathtt{Ben}}, \mathtt{ESE}]$ *is IND-CLS-CPA-secure.*

Finally, the following result, proved in [8], shows that consistency of $\mathtt{Tags}$ is a necessary condition for the template scheme to be IND-CLS-CPA-secure.

Theorem 3. *If* $\mathtt{Tags}$ *is not consistent, then valid EFSE* $\mathtt{FSE}_{\mathtt{Etag}}[\mathtt{Tags}, \mathcal{F}_{\mathtt{Ben}}, \mathtt{ESE}]$ *is not IND-CLS-CPA-secure.*

Summing up, if CPTF $\mathtt{Tags}$ is consistent, $\mu = \max_m |\mathtt{Tags}(m)|$ is small, batch-encoding oracle $\mathcal{F}_{\mathtt{Ben}}$ is PP-CBA-secure *and* collision-free, and ESE scheme $\mathtt{ESE}$ is IND-DCPA-secure, then $\mathtt{FSE}_{\mathtt{Etag}}[\mathtt{Tags}, \mathcal{F}_{\mathtt{Ben}}, \mathtt{ESE}]$ is a valid, (optimally) IND-CLS-CPA-secure EFSE. If $\mathtt{Tags}$ is not consistent, the scheme is not IND-CLS-CPA-secure.

5 Toward an Optimally-Secure Scheme

We now seek an EFSE construction achieving the optimal level of security, IND-CLS-CPA, as defined in Definition 1. First, we show that the only previously existing candidate is, in general, not IND-CLS-CPA-secure due to Theorem 3. Then, we construct the first IND-CLS-CPA-secure EFSE scheme using the template from Sect. 4. Finally, we show that in a sense, the space-inefficiency of the secure scheme is necessary to accommodate general closeness domains.

Analysis of an EFSE scheme similar to [18]. The only previously existing EFSE-type scheme is presented in [18]. As noted, the basic structure of our template tag-encoding scheme is a generalization of their method, so it is natural to define a tag-encoding scheme in our model that captures the essence of (and perhaps improves) the [18] scheme. Here we show that this scheme has poor space-efficiency (length of ciphertext linear in the degree of a message) and yet fails to achieve IND-CLS-CPA-security. (Moreover, it only works on certain closeness domains.) In contrast, the schemes we develop in later sections either achieve IND-CLS-CPA-security, or have much better space-efficiency.

In [18], the authors construct several variants of a fuzzy-searchable scheme; here we present a variant/generalization[6]. This construction only works on closeness domains $\Lambda = (\mathcal{D}, \mathrm{Cl})$ with the following constraint: for any $m_1, m_2 \in \mathcal{D}$, if

[6] There are minor differences—notably, $\mathtt{FSEtagNbs}$ uses an IND-DCPA-secure ESE rather than a (stronger) IND-CPA-secure scheme, but this is not an issue as [18] leaks equality already through its encoding strategy. Moreover, we could instantiate $\mathtt{FSEtagNbs}$ with an IND-CPA-secure scheme in place of $\mathtt{ESE}$ and the attack described would still work, since the attack exploits the $\mathcal{F}_{\mathtt{Ben}}$-tagged neighbors, not $\mathtt{ESE}$. Other differences in [18] are inconsequential to the analysis.

$\mathrm{Cl}(m_1, m_2) = \texttt{far}$, then there exists no m with $\mathrm{Cl}(m_1, m) = \mathrm{Cl}(m_2, m) = \texttt{close}$. (In particular, this generally rules out rigid closeness domains.) We define the *neighbor set* of an element m to be $\texttt{Nb}_m = \{m' \in \mathcal{D} \mid m' \neq m, \mathrm{Cl}(m, m') = \texttt{close}\}$. Define $\texttt{TagNbs} : \mathcal{D} \to \mathcal{V}_\Lambda$ as $\texttt{TagNbs}(m) = \texttt{Nb}_m \cup \{m\}$, where $\mathcal{V}_\Lambda$ is the power set of $\mathcal{D}$.

Note that if $\mathrm{Cl}(m, m') = \texttt{close}$ then $\texttt{TagNbs}(m) \cap \texttt{TagNbs}(m') \supseteq \{m, m'\} \neq \emptyset$, and if $\mathrm{Cl}(m, m') = \texttt{far}$, $\texttt{TagNbs}(m) \cap \texttt{TagNbs}(m') = \emptyset$ by the condition on Λ, so TagNbs is a CPTF on Λ. Let $\mathcal{F}_{\texttt{Ben}}$ be a collision-free batch-encoding family on $\mathcal{V}_\Lambda$ and ESE an ESE scheme on $\mathcal{D}$, and define FSEtagNbs to be $\texttt{FSE}_{\texttt{Etag}}[\texttt{TagNbs}, \mathcal{F}_{\texttt{Ben}}, \texttt{ESE}]$ as per Fig. 3. If the max degree $\Delta = \max_{m \in \mathcal{D}} |\texttt{Nb}_m|$ of Λ is small, FSEtagNbs is an EFSE. However, the ciphertext size is linear in Δ.

We claim that FSEtagNbs is IND-CLS-CPA-insecure for the closeness domains considered by [18], as well as most other conceivably useful domains. Suppose, for example, that the closeness domain has two pairs of close messages with different numbers of common close neighbors: i.e.,

$$\mathrm{Cl}(m_0, m_2) = \mathrm{Cl}(m_1, m_2) = \texttt{close}\ ;\ |\texttt{Nb}_{m_0} \cap \texttt{Nb}_{m_2}| \neq |\texttt{Nb}_{m_1} \cap \texttt{Nb}_{m_2}|. \tag{1}$$

Then the condition of Theorem 3 is satisfied for $q = 2$, so that FSEtagNbs is IND-CLS-CPA-insecure for any domain having m_0, m_1, m_2 that satisfy (1).

The schemes of [18] are, essentially, instantiations of FSEtagNbs on closeness domains defined in terms of keywords and edit distance (the minimum number of operations—insertions, deletions, substitutions—required to transform one string into the other.) If $\delta > 2$ is the threshold edit distance, take m_2 to be any message of length at least $\delta + 1$. Let m_0 be m_2 but with the first letter changed. Let m_1 be m_2 but with the last δ letters changed. Then m_0 and m_2 share more neighbors than m_1 and m_2 share, so these messages satisfy (1) and FSEtagNbs is IND-CLS-CPA-insecure in this case.

Construction of the first secure EFSE scheme. We now improve on the scheme of [18] and construct an EFSE scheme that is IND-CLS-CPA-secure even on rigid closeness domains. Let $\Lambda = (\mathcal{D}, \mathrm{Cl})$ be a closeness domain with $\mathcal{D}$ finite. Let $\mathcal{G}_\Lambda = (\mathcal{V}_\Lambda, \mathcal{E}_\Lambda)$ be the closeness graph of Λ. For $m \in \mathcal{D}$, let $E_m = \{\{m, m'\} \in \mathcal{E}_\Lambda \mid m' \in \mathcal{V}_\Lambda\}$ be the set of incident edges to m in $\mathcal{G}_\Lambda$, and note that message degree $\Delta_m = |E_m|$ and max degree $\Delta = \max_{m \in \mathcal{D}} \Delta_m$.

So that all messages have the same number of close neighbors, we introduce dummy messages. Construct a new graph $\mathcal{G}_{\texttt{dum}} = (\mathcal{V}_{\texttt{dum}}, \mathcal{E}_{\texttt{dum}})$ where $\mathcal{V}_{\texttt{dum}} = \mathcal{V}_\Lambda \cup \{w_1, \ldots, w_\Delta\}$, and $\mathcal{E}_{\texttt{dum}}$ consists of all edges in $\mathcal{E}_\Lambda$, plus for any $m \in \mathcal{V}_\Lambda$, if $\Delta - \Delta_m > 0$ then let $\mathcal{E}_{\texttt{dum}}$ also contain edges $\{m, w_1\}, \ldots, \{m, w_{\Delta - \Delta_m}\}$. We call these additional edges *dummy edges* and $w_1, \ldots, w_\Delta$ *dummy vertices*. $\mathcal{G}_{\texttt{dum}}$ is thus a graph in which every element of $\mathcal{V}_\Lambda \subset \mathcal{V}_{\texttt{dum}}$ has degree Δ.

Define $\texttt{TagEdges} : \mathcal{D} \to \mathcal{E}_{\texttt{dum}}$ as $\texttt{TagEdges}(m) = \{e \in \mathcal{E}_{\texttt{dum}} \mid m \in e\}$. Note: if $\mathrm{Cl}(m, m') = \texttt{close}$ then $\texttt{TagEdges}(m) \cap \texttt{TagEdges}(m') \supseteq \{\{m, m'\}\} \neq \emptyset$; and if $\mathrm{Cl}(m, m') = \texttt{far}$ then $\texttt{TagEdges}(m) \cap \texttt{TagEdges}(m') = \emptyset$. So TagEdges is a CPTF.

Let $\mathcal{F}_{\texttt{Ben}}$ be a collision-free batch-encoding family on domain $\mathcal{E}_{\texttt{dum}}$ and some range $\mathcal{R}_{\texttt{En}}$, and let ESE be an ESE scheme on $\mathcal{D}$. Define the StructFSE scheme FSEtagEdges as $\texttt{FSE}_{\texttt{Etag}}[\texttt{TagEdges}, \mathcal{F}_{\texttt{Ben}}, \texttt{ESE}]$ according to Fig. 3. Notice that for

all $m \in \mathcal{D}$, $|\texttt{TagEdges}(m)| \leq \Delta$. So, if Λ has small max degree, `FSEtagEdges` is efficient.

Now, Theorem 4 provides the security guarantee of `FSEtagEdges`. The proof is in [8], and simply shows the main condition of Theorem 2 (i.e., consistency of `TagEdges`) is satisfied in this case.

Theorem 4. *If the max degree Δ of the closeness domain is small, and if* `ESE` *is IND-DCPA-secure and $\mathcal{F}_{\texttt{Ben}}$ is PP-CBA-secure, then* `FSEtagEdges` *is IND-CLS-CPA-secure.*

Recall that certain blockcipher-based constructions (discussed earlier) satisfy the necessary efficiency, security, and functionality conditions for `ESE` and $\mathcal{F}_{\texttt{Ben}}$. The final missing piece to achieve an efficient IND-CLS-CPA-secure scheme is that `TagEdges` should be efficient; i.e., for any message $m \in \mathcal{D}$ it should be easy to compute E_m. Thus, `FSEtagEdges` is an IND-CLS-CPA-secure EFSE scheme on Λ if the following two conditions hold:

(1) the max degree of Λ is small;

(2) E_m is predetermined or calculated on-the-fly.

Of course, whether these conditions are satisfied depends on the closeness domain Λ. It is an interesting question to identify when (1) holds, and how to achieve (2) in those situations. However, the possibilities are wide-ranging and so we leave this as a topic of future research.

Now, we have successfully created a IND-CLS-CPA-secure scheme, but at what cost? It is apparent that, even if the max degree Δ is small enough for the scheme to be efficient, its size can lead to huge space-inefficiency, since ciphertexts in `FSEtagEdges` have length linear in Δ. And Δ could certainly be quite large—for instance, on a dense or high-dimensional metric closeness domain, even a small threshold supplies each message with many close neighbors.

Nevertheless, if we desire a general FSE construction to work on arbitrary closeness domains, such long ciphertexts are necessary. We explain in the following section.

Lower bound on ciphertext length of an FSE scheme for general closeness domains. Notice that our `FSEtagEdges` scheme is defined independently of the closeness graph—in particular, the algorithms `makeDS` and `fuzzyQ` did not exploit any special structure of the closeness graph. In the following result, we show that to have such a scheme construction that is valid for "general" closeness domains, it requires ciphertext length linear in the max degree of the closeness domain. Moreover, note that this is an informational theoretic requirement, and relies only on functionality, rather than security, of the schemes. The proof of the theorem is in [8].

Theorem 5. *Let $\mathcal{D}$ be a fixed domain and Δ an integer with $2 \leq \Delta \ll |\mathcal{D}|$. There exists a family of closeness domains $\{\Lambda_i = (\mathcal{D}, Cl_i)\}_{i \in I}$, each with max degree at most Δ, so that if $\{FSE_i\}_{i \in I}$ is a family of FSE schemes on the respective closeness domains that have common* `makeDS` *and* `fuzzyQ` *algorithms and a common ciphertext space, then the ciphertext length is at least $\Delta/2$.*

The bound on ciphertext length asymptotically matches the space-efficiency of scheme `FSEtagEdges` from the previous section, demonstrating that `FSEtagEdges` is "best-possible" for FSE schemes that work on general closeness domains.

6 Space-Efficient Schemes

Theorem 5 indicates that it is costly to construct EFSE schemes on general closeness domains. A natural question is whether we can improve efficiency by focusing on closeness domains that have nice structure. In particular, to avoid the strict conditions leading to Theorem 5 we should consider non-rigid closeness domains, where near message pairs enable "false positives" in a fuzzy query. However, note that if an adversary has any probabilistic edge in distinguishing near message pairs that lead to false positives and those that don't, he can easily break IND-CLS-CPA-security. To avoid such an attack, one must force the probability a near message pair is sent to a close ciphertext pair to be *uniform* over all near message pairs. But this negates the flexibility advantage of near messages—we expect an EFSE scheme satisfying this uniformity condition on near pairs would be as inefficient as the `FSEtagEdges` scheme. Thus, it appears that IND-CLS-CPA-security is too strong for more efficient EFSEs to achieve, even on non-rigid closeness domains. So to evaluate more efficient schemes, we need a new, weaker notion of security.

Intuitively, what information must a EFSE scheme on a non-rigid closeness domain Λ leak, given that some number of ciphertexts are known? Let H be the set of messages corresponding to known ciphertexts. For two messages in the same component of the induced nearness subgraph $\mathcal{G}_{\Lambda}^{\mathtt{N}}(H)$ (we say they are in the same *nearness component*) an EFSE is designed so that anyone might discover this fact by running `fuzzyQ` on their ciphertexts. So, by using EFSE we automatically give up a large amount of information about messages in the same nearness component (namely, their link through a chain of known near pairs.) It is a natural step to consider allowing more information leakage relating messages within the same nearness component, while protecting as much as possible about messages in different components, and hiding the "general location" of a message in the domain. We also might restrict our view to schemes on "regular" closeness domains—that is, domains where message closeness is defined in a similar manner in all parts of the space. Otherwise, irregularities in the domain would inherently reveal message locations.

Toward this end, we focus on real ℓ-dimensional domains where closeness of messages is defined regularly throughout the space. In particular, there is a regular lattice $\mathcal{L}$ such that the closeness function is invariant by $\mathcal{L}$-translations. Our new security notion then requires schemes to hide all information about plaintexts in different nearness components except for their "local structure" with respect to this lattice. The important implication is that nothing major (i.e., only "local structure") is revealed about the relationship between a pair of disconnected messages (i.e., messages that cannot be connected through a chain of near known corresponding ciphertext pairs). Hence, it is a sort of "macrostructure security" across disconnected nearness components.

In this section and related sections deferred to the full version [8], we focus on schemes achieving this security on certain metric closeness domains over $\mathbb{R}^\ell$. Suppose we can select a lattice $\mathcal{L} \subset \mathbb{R}^\ell$ and "anchor radius" $\rho > 0$ so that close messages are each within distance ρ of a common lattice point, and far messages are not. Then an obvious tagging strategy is to send a message to its *anchor points*: the lattice points within distance ρ of the message. We prove that the resulting scheme is secure with respect to $\mathcal{L}$ under the new definition. This new "macrostructure-secure" construction leads to a more detailed discussion that is relegated to the full version [8]. There, we pose an optimization problem related to the general construction, present some simple scheme constructions and a way to stitch simple constructions together to build useful schemes, then describe a practical instantiation of the scheme for fuzzy search on biometric data. Finally, in [8] we propose a direction of further research toward "probabilistic EFSE" schemes built out of locality-sensitive hash functions.

6.1 Macrostructure Security on Lattice-Regular Closeness Domains

Our new notion of security will apply to closeness domains over $\mathbb{R}^\ell$ for which closeness is defined in a "regular" manner over the entire space. We characterize this regularity using a regular lattice on $\mathbb{R}^\ell$. Then, the security notion will hide everything about plaintexts except for how they locally relate to this regular lattice.

Lattice-regular closeness domains. Let $\mathcal{L}$ be a regular lattice in $\mathbb{R}^\ell$, that is, a set of vectors characterized as all integer combinations of a finite set of linearly independent basis vectors. We say a closeness domain $\Lambda = (\mathbb{R}^\ell, \mathrm{Cl})$ is $\mathcal{L}$*-regular* if for any $\mathbf{x}, \mathbf{y} \in \mathbb{R}^\ell$ and any $\mathbf{w} \in \mathcal{L}$, $\mathrm{Cl}(\mathbf{x}, \mathbf{y}) = \mathrm{Cl}(\mathbf{x} + \mathbf{w}, \mathbf{y} + \mathbf{w})$. That is, closeness relations are invariant under translation by any lattice vector. We say $\mathcal{L}$ is a *regularity lattice* of Λ. Also, if $\mathbf{z} = \mathbf{x} + \mathbf{w}$ for some $\mathbf{x}, \mathbf{z} \in \mathbb{R}^\ell$ and $\mathbf{w} \in \mathcal{L}$, we say that $\mathbf{x}$ and $\mathbf{z}$ are in the same $\mathcal{L}$*-class* and that $\mathbf{w}$ is the $\mathcal{L}$*-witness* from $\mathbf{x}$ to $\mathbf{z}$.

Macrostructure security. Let $\mathcal{L}$ be a regular lattice on $\mathbb{R}^\ell$ and let $\Lambda = \big(\mathbb{R}^\ell, \mathrm{Cl}\big)$ be a $\mathcal{L}$-regular closeness domain on $\mathbb{R}^\ell$. The security notion is as follows.

Definition 3. *Let* $\mathtt{FSE} = (\mathcal{K}, \mathcal{E}nc, \mathcal{D}ec, \mathtt{makeDS}, \mathtt{fuzzyQ})$ *be an EFSE scheme on* $\mathcal{L}$*-regular closeness domain* Λ*. For an adversary* A *and* $b \in \{0, 1\}$*, let* $\mathbf{Exp}^{ind\text{-}nr\mathcal{L}\text{-}cpa\text{-}b}_{\mathtt{FSE}}(A)$ *be identical to IND-CPA experiment* $\mathbf{Exp}^{ind\text{-}cpa\text{-}b}_{\mathtt{FSE}}(A)$ *in Fig. 1, but with the restriction: for LR-queries* (m^i_0, m^i_1)*,* $i \in [q]$ *made by the adversary, letting* $H_0 = \{m^1_0, \ldots, m^q_0\}$ *and* $H_1 = \{m^1_1, \ldots, m^q_1\}$*, require*

1. $|m^i_0| = |m^i_1|$ *for all* $i \in [q]$*;*
2. $\forall i \in [q]$*,* m^i_0 *and* m^i_1 *are in the same* $\mathcal{L}$*-class; furthermore, the* $\mathcal{L}$*-witness from* m^i_0 *to* m^i_1 *is also the* $\mathcal{L}$*-witness from* m^j_0 *to* m^j_1 *whenever* m^i_0 *and* m^j_0 *are in the same connected component of* $\mathcal{G}^{\mathtt{N}}_{\Lambda}(H_0)$*.*

For an adversary A, define its IND-NR$\mathcal{L}$-CPA advantage *against* FSE *as*

$$\mathbf{Adv}^{ind\text{-}nr\mathcal{L}\text{-}cpa}_{\mathtt{FSE}}(A) = \Pr\left[\mathbf{Exp}^{ind\text{-}nr\mathcal{L}\text{-}cpa\text{-}1}_{\mathtt{FSE}}(A)=1\right] - \Pr\left[\mathbf{Exp}^{ind\text{-}nr\mathcal{L}\text{-}cpa\text{-}0}_{\mathtt{FSE}}(A)=1\right].$$

We say that FSE *is* indistinguishable under same-nearness-component-$\mathcal{L}$-class chosen-plaintext attacks *(IND-NR$\mathcal{L}$-CPA-secure) or, alternatively,* macro-structure-secure with respect to anchor lattice $\mathcal{L}$ *(MacroStruct-$\mathcal{L}$-secure) if the IND-NR$\mathcal{L}$-CPA advantage of any adversary against* FSE *is small.*

The second LR-query requirement asks that a left-query component of $\mathcal{G}^{\mathtt{N}}_{\Lambda}(H_0)$ is a $\mathcal{L}$-translation (translation by a vector in $\mathcal{L}$) of the corresponding right-query component of $\mathcal{G}^{\mathtt{N}}_{\Lambda}(H_1)$. This implies that left and right queries have the same equality/closeness pattern, which we can see by the following. If $m^i_0 = m^j_0$ then these messages are in the same nearness component (as they are the same vertex) so $\exists l \in \mathcal{L}$ with $m^i_1 = m^i_0 + l = m^j_0 + l = m^j_1$. If $\mathrm{Cl}(m^i_0, m^j_0) \in \{\mathtt{close}, \mathtt{near}\}$ then these messages are in the same nearness component so $\exists l \in \mathcal{L}$ with $m^i_1 = m^i_0 + l$, $m^j_1 = m^j_0 + l$, implying $d(m^i_1, m^j_1) = d(m^i_1 + l, m^j_1 + l) = d(m^i_0, m^j_0)$, so $\mathrm{Cl}(m^i_1, m^j_1) = \mathrm{Cl}(m^i_0, m^j_0)$. Thus, MacroStruct-$\mathcal{L}$-security is clearly weaker than IND-CLS-CPA-security.

Returning to the big picture, an MacroStruct-$\mathcal{L}$-secure scheme may leak how all messages in a nearness component lie with respect to nearby points in the regularity lattice. However, since the lattice itself is regular, no information is leaked about where those nearby lattice points actually are. Thus, for messages in different nearness components, an adversary learns nothing about the distance between them, or their approximate locations in the space, besides some bits with low significance, and that the distance is above $\delta^{\mathtt{F}}$ (which is by design.)

Practitioners should be aware that, depending on the application, Macro-Struct-$\mathcal{L}$-security is not always an appropriate security guarantee. For instance, consider a scenario where IP addresses are encrypted by a MacroStruct-$\mathcal{L}$-secure scheme and the lattice points are IP addresses with the final byte equal to 0. The scheme could possibly leak the last byte of each IP address, perhaps revealing the particular types of conversants in IP traffic data. In general, when the "least significant" bits of data contain sensitive information, MacroStruct-$\mathcal{L}$-security may not be enough.

6.2 General Macrostructure-Secure Construction on Metric Closeness Domains

We aim to construct space-efficient EFSE schemes that meet our new notion of MacroStruct-$\mathcal{L}$-security for some regularity lattice. For practicality, we focus on the metric closeness domain on $\mathbb{R}^\ell$, Euclidean metric d, close threshold $\delta^{\mathtt{C}} > 0$, and far threshold $\delta^{\mathtt{F}} \geq \delta^{\mathtt{C}}$, i.e., $\Lambda = \left(\mathbb{R}^\ell, \mathcal{M}_d^{\delta^{\mathtt{C}}, \delta^{\mathtt{F}}}\right)$. Notice that Λ is $\mathcal{L}$-regular for *any* lattice $\mathcal{L} \subset \mathbb{R}^\ell$. We now define a few useful objects that will play a leading role in the general construction. Then, the construction follows.

Anchor radii and points. Fix a lattice $\mathcal{L}$ in $\mathbb{R}^\ell$. For $\rho > 0$, we say that ρ is an *anchor radius* on closeness domain Λ and lattice $\mathcal{L}$, and $\{\mathbf{v} \in \mathcal{L} \mid d(\mathbf{m}, \mathbf{v}) \leq \rho\}$ is

the set of *anchor points* of message $\mathbf{m}$, if (1) any two close messages $\mathbf{m}, \mathbf{m}' \in \mathcal{D}$ have a common anchor point, and (2) any two far messages $\mathbf{m}, \mathbf{m}' \in \mathcal{D}$ have no common anchor points.

General macrostructure-secure construction and its security. If ρ is an anchor radius on Λ and $\mathcal{L}$, then $\mathtt{TagsAnc}^{\rho}_{\mathcal{L}} : \mathbb{R}^{\ell} \to \mathcal{L}$ defined as $\mathtt{TagsAnc}^{\rho}_{\mathcal{L}}(\mathbf{m}) = \{\mathbf{v} \in \mathcal{L} \mid d(\mathbf{m}, \mathbf{v}) \leq \rho\}$ is a CPTF on Λ, as condition (1) implies that whenever $d(\mathbf{m}, \mathbf{m}') \leq \delta^{\mathtt{C}}$, there exists $\mathbf{v} \in \mathcal{L}$ such that $\mathtt{TagsAnc}^{\rho}_{\mathcal{L}}(\mathbf{m}) \cap \mathtt{TagsAnc}^{\rho}_{\mathcal{L}}(\mathbf{m}') \supseteq \{\mathbf{v}\}$; and condition (2) implies $\mathtt{TagsAnc}^{\rho}_{\mathcal{L}}(\mathbf{m}) \cap \mathtt{TagsAnc}^{\rho}_{\mathcal{L}}(\mathbf{m}') = \emptyset$ whenever $d(\mathbf{m}, \mathbf{m}') > \delta^{\mathtt{F}}$. Thus, if ρ is an anchor radius on Λ and $\mathcal{L}$, $\mathcal{F}_{\mathtt{Ben}} = (\mathcal{K}_{\mathtt{Ben}}, \mathtt{En}, \mathtt{Ben})$ is a collision-free batch-encoding family on domain $\mathcal{D}_{\mathtt{En}} = \mathcal{L}$, and $\mathtt{ESE}$ is an ESE scheme on $\mathcal{D}$, then the scheme $\mathtt{FSEtagAnc}^{\rho}_{\mathcal{L}} = \mathtt{FSE}_{\mathtt{Etag}}[\mathtt{TagsAnc}^{\rho}_{\mathcal{L}}, \mathcal{F}_{\mathtt{Ben}}, \mathtt{ESE}]$ is a StructFSE scheme by Theorem 1. The following result is proved in [8].

Theorem 6. $\mathtt{FSEtagAnc}^{\rho}_{\mathcal{L}}$ *defined ias above is MacroStruct-*$\mathcal{L}$*-secure provided* $\mathtt{ESE}$ *is IND-DCPA-secure,* $\mathcal{F}_{\mathtt{Ben}}$ *is PP-CBA-secure,* $\mu = \max_{\mathbf{m} \in \mathcal{D}} |\{\mathbf{v} \in \mathcal{L} \mid d(\mathbf{m}, \mathbf{v}) \leq \rho\}|$ *is small, and we can efficiently compute anchor points.*

Together, Theorem 1 and Theorem 6 say that if we can find an anchor radius ρ on closeness domain Λ and lattice $\mathcal{L}$ such that the maximum number of anchor points μ is small, and we can efficiently compute anchor points, $\mathtt{FSEtagAnc}^{\rho}_{\mathcal{L}}$ as constructed above is an MacroStruct-$\mathcal{L}$-secure EFSE scheme on Λ.

Note that the problem of finding a given message's anchor points is essentially the *ρ-close vectors problem (ρ-CVP)* on the appropriate parameters. Unfortunately, this problem is harder (assuming fixed maximum number of anchor points μ) then the standard closest vector problem with unlimited preprocessing, which has been shown to be NP-hard in general [19]. Thus, to ensure both efficiency and security in our specific constructions, it is vital to demonstrate how to efficiently compute anchor points.

The general "anchor-point" construction presented above provides a template for defining macrostructure-secure schemes. In the full version [8], we analyze some of the ramifications and possibilities. First, we pose the general open problem of how to choose anchor lattice and anchor radius to optimize space-efficiency and flexibility of a scheme. We next present several specific schemes, and identify how to stitch methods together to create a scheme supporting "conjunctive" closeness. Then, to enhance understanding, we describe and analyze a scheme for a practical application: supporting fuzzy search on biometric (fingerprint) data.

References

1. Adjedj, M., Bringer, J., Chabanne, H., Kindarji, B.: Biometric identification over encrypted data made feasible. In: Prakash, A., Sen Gupta, I. (eds.) ICISS 2009. LNCS, vol. 5905, pp. 86–100. Springer, Heidelberg (2009)
2. Amanatidis, G., Boldyreva, A., O'Neill, A.: Provably-secure schemes for basic query support in outsourced databases. In: Barker, S., Ahn, G.-J. (eds.) Data and Applications Security 2007. LNCS, vol. 4602, pp. 14–30. Springer, Heidelberg (2007)

3. Bellare, M., Boldyreva, A., O'Neill, A.: Deterministic and efficiently searchable encryption. In: Menezes, A. (ed.) CRYPTO 2007. LNCS, vol. 4622, pp. 535–552. Springer, Heidelberg (2007)
4. Bellare, M., Kohno, T., Namprempre, C.: Breaking and provably repairing the SSH authenticated encryption scheme: a case study of the encode-then-encrypt-and-MAC paradigm. ACM Trans. Inf. Syst. Secur. **7**(2), 206–241 (2004)
5. Bellare, M., Namprempre, C.: Authenticated encryption: relations among notions and analysis of the generic composition paradigm. In: Okamoto, T. (ed.) ASIACRYPT 2000. LNCS, vol. 1976, pp. 531–545. Springer, Heidelberg (2000)
6. Bellare, M., Ristenpart, T., Rogaway, P., Stegers, T.: Format-preserving encryption. In: Jacobson Jr, M.J., Rijmen, V., Safavi-Naini, R. (eds.) SAC 2009. LNCS, vol. 5867, pp. 295–312. Springer, Heidelberg (2009)
7. Bellare, M., Rogaway, P.: On the construction of variable-input-length ciphers. In: Knudsen, L.R. (ed.) FSE 1999. LNCS, vol. 1636, pp. 231–244. Springer, Heidelberg (1999)
8. Boldyreva, A., Chenette, N.: Efficient fuzzy search on encrypted data. Full version of this paper. Cryptology ePrint Archive (2013). https://eprint.iacr.org/
9. Boldyreva, A., Chenette, N., Lee, Y., O'Neill, A.: Order-preserving symmetric encryption. In: Joux, A. (ed.) EUROCRYPT 2009. LNCS, vol. 5479, pp. 224–241. Springer, Heidelberg (2009)
10. Bringer, J., Chabanne, H., Kindarji, B.: Error-tolerant searchable encryption. In: Proceedings of the 2009 IEEE International Conference on Communications, ICC 2009, pp. 768–773. IEEE Press, Piscataway, NJ, USA (2009)
11. Bringer, J., Chabanne, H., Kindarji, B.: Identification with encrypted biometric data. Secur. Commun. Netw. **4**(5), 548–562 (2011)
12. Chang, Y.-C., Mitzenmacher, M.: Privacy preserving keyword searches on remote encrypted data. In: Ioannidis, J., Keromytis, A.D., Yung, M. (eds.) ACNS 2005. LNCS, vol. 3531, pp. 442–455. Springer, Heidelberg (2005)
13. Curtmola, R., Garay, J.A., Kamara, S., Ostrovsky, R.: Searchable symmetric encryption: improved definitions and efficient constructions. In: Juels, A., Wright, R.N., di Vimercati, S.D.C. (eds.) ACM Conference on Computer and Communications Security, pp. 79–88. ACM, New York (2006)
14. Gentry, C.: A fully homomorphic encryption scheme. PhD Thesis, Stanford University (2009)
15. Katz, J., Sahai, A., Waters, B.: Predicate encryption supporting disjunctions, polynomial equations, and inner products. In: Smart, N.P. (ed.) EUROCRYPT 2008. LNCS, vol. 4965, pp. 146–162. Springer, Heidelberg (2008)
16. Kuzu, M., Islam, M.S., Kantarcioglu, M.: Efficient similarity search over encrypted data. In: Proceedings of the 2012 IEEE 28th International Conference on Data Engineering, ICDE 2012, pp. 1156–1167. IEEE Computer Society, Washington, DC, USA (2012)
17. Lewko, A., Okamoto, T., Sahai, A., Takashima, K., Waters, B.: Fully secure functional encryption: attribute-based encryption and (hierarchical) inner product encryption. In: Gilbert, H. (ed.) EUROCRYPT 2010. LNCS, vol. 6110, pp. 62–91. Springer, Heidelberg (2010)
18. Li, J., Wang, Q., Wang, C., Cao, N., Ren, K., Lou, W.: Fuzzy keyword search over encrypted data in cloud computing. In: INFOCOM, pp. 441–445. IEEE (2010)
19. Micciancio, D.: The hardness of the closest vector problem with preprocessing. IEEE Trans. Inf. Theory **47**(3), 1212–1215 (2001)
20. Naor, M., Reingold, O.: On the construction of pseudorandom permutations: Luby-Rackoff revisited. J. Cryptol. **12**(1), 29–66 (1999)

21. Pandey, O., Rouselakis, Y.: Property preserving symmetric encryption. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT 2012. LNCS, vol. 7237, pp. 375–391. Springer, Heidelberg (2012)
22. Rogaway, P., Shrimpton, T.: Deterministic authenticated-encryption: a provable-security treatment of the key-wrap problem. IACR Cryptology ePrint Archive (2006)
23. Song, D.X., Wagner, D., Perrig, A.: Practical techniques for searches on encrypted data. In: IEEE Symposium on Security and Privacy, pp. 44–55 (2000)
24. Vaikuntanathan, V.: Computing blindfolded: new developments in fully homomorphic encryption. In: Ostrovsky, R. (ed.) FOCS, pp. 5–16. IEEE, New York (2011)
25. Wang, C., Wang, Q., Ren, K.: Towards secure and effective utilization over encrypted cloud data. In: Proceedings of the 2011 31st International Conference on Distributed Computing Systems Workshops, ICDCSW 2011, pp. 282–286. IEEE Computer Society, Washington, DC, USA (2011)

Author Index