KATHOLIEKE UNIVERSITEIT
LEUVEN

Arenberg Doctoral School of Science, Engineering & Technology
Faculty of Engineering
Department of Electrical Engineering (ESAT)

Automated Techniques for Hash Function and Block Cipher Cryptanalysis

Nicky MOUHA

Dissertation presented in partial
fulfillment of the requirements for
the degree of Doctor of Engineering

June 2012

Automated Techniques for Hash Function and Block Cipher Cryptanalysis

Nicky MOUHA

Jury:
Prof. em. dr. ir. Yves D. Willems, chair
Prof. dr. ir. Bart Preneel, supervisor
Prof. dr. ir. Vincent Rijmen, secretary
Prof. dr. ir. Frank Piessens
Dr. Svetla Petkova-Nikova
Dr. Matt Robshaw
(Applied Cryptography Group,
Orange Labs, France)

Dissertation presented in partial fulfillment of the requirements for the degree of Doctor of Engineering

June 2012

Our greatest glory is not in never falling, but in rising every time we fall.

Confucius

D/2012/7515/71
ISBN 978-94-6018-535-9

If I have seen further it is only by standing on the shoulders of giants.

Isaac Newton

Acknowledgments

Every research paper is a puzzle piece, linking other pieces together to reveal a bigger picture. As such, research can't be done in isolation. In the past few years, I've been extremely fortunate to meet the world's best and brightest in the field of symmetric-key cryptography. Many times, I've found myself to be the stupidest and most ignorant person in the room. To me, such experiences have been exciting and humbling, but most of all a great opportunity to learn.

Therefore, I would like to personally thank everyone who contributed in one way or another to this thesis. Unfortunately, this will not be possible. I have been inspired by far too many brilliant people at conferences, workshops and seminars throughout the world. Thanks to everyone who is not explicitly mentioned in these acknowledgments. Often, your simple insights completely changed my view of the world.

Most of all, I would like to thank my supervisor, prof. dr. ir. Bart Preneel. He has given me the freedom to work independently, yet was always there for me when I needed his advice. I've been assigned a low workload in teaching, supervising and administration. To compensate for this, I've frequently volunteered to organize conferences, guide newcomers and perform computer-related tasks. I've been very fortunate to travel the world many times over. This has been a lot of fun for me, but also a great cultural enrichment. For all of these reasons and more, I consider myself to be one of the luckiest Ph.D. students. I can't thank my supervisor enough for this.

Special thanks go to the the members of the jury: prof. em. dr. ir. Yves D. Willems, prof. dr. ir. Bart Preneel, prof. dr. ir. Vincent Rijmen, prof. dr. ir. Frank Piessens, dr. Svetla Petkova-Nikova and dr. Matt Robshaw. Their detailed comments and suggestions have greatly improved the text of this Ph.D. thesis.

I faced a setback before I even started working on my Ph.D., when my application for an FWO-Vlaanderen scholarship was rejected. However, my supervisor continued to encourage me, and helped me through a successful application for an IWT-Vlaanderen scholarship. I would like to thank the jury composed by IWT, and diligent jury member Joan Daemen in particular, for their positive evaluation. Thanks to IWT for the financial support, and to scholarship coordinators Marc Pollet and Michèle Oleo for the administrative support.

The invaluable input of my co-authors should be acknowledged: Jean-Philippe Aumasson, Tor E. Bjørstad, Andrey Bogdanov, Christophe De Cannière, Dawu Gu,

Sebastiaan Indesteege, Özgül Küçük, Gaëtan Leurent, Willi Meier, Florian Mendel, Thomas Peyrin, Raphael C.-W. Phan, Bart Preneel, Vincent Rijmen, Yu Sasaki, Gautham Sekar, Yue Sun, Petr Susil, Søren S. Thomsen, Elmar Tischhauser, Deniz Toz, Meltem Sönmez Turan, Markus Ullrich, Kerem Varıcı, Vesselin Velichkov, Meiqin Wang and Qingju Wang. I've always tried to involve other people as early as possible in the scientific process of writing a paper. I know that this slows down the research, but I do believe that the resulting papers have a much higher quality than I could achieve on my own.

I would also like to thank all the other members of the symmetric-key cryptography subgroup at COSIC, for providing a very pleasant working environment: Mohamed Ahmed Abdelraheem, Elena Andreeva, Begül Bilgin, Jiazhe Chen, Yoni De Mulder, Orr Dunkelman, Barış Ege, Emilia Käsper, Kota Ideguchi, Bart Mennink, Svetla Nikova, Christian Rechberger, Kyoji Shibutani, Fatih Sulak, Vasin Suttichaya, Gauthier Van Damme and Hirotaka Yoshida. Special thanks go to prof. dr. ir. Vincent Rijmen for leading our subgroup, and advising us along the way.

I'd especially like to thank secretary Péla Noë, European projects coordinator Saartje Verheyen and accountants Elvira Wouters, Elsy Vermoesen and Wim Devroye. Without their help, it would be impossible to get anything done at all. Thanks to Sebastiaan Indesteege, who saved me countless hours by providing me with a template to write this thesis. Unfortunately, COSIC has grown too large to thank all other members individually.

In 2011, I've enjoyed a brief stay at Tsinghua University under the supervision of prof. dr. Xiaoyun Wang. I am extremely grateful for this opportunity. Thanks to secretary Qi Shi for sorting out all the administrative and technical issues. I'd like to say *xièxie* to all my colleagues: Jingguo Bi, Kuan Cheng, Jiazhe Chen, Dan Ding, Lianzhi Fu, Lidong Han, Changhui Hu, Keting Jia, Jianwei Li, Leibo Li, Mingjie Liu, Xichen Mu, Yongchuan Niu, Yue Sun, Chengliang Tian, An Wang, Maoning Wang, Zhongyue Wang, Wei Wei, Hongbo Yu, Yanyan Yu, Jingyuan Zhao, Xuexin Zheng and Guizhen Zhu. Non-Chinese readers may struggle to read the list of names, but they are very familiar names to me. I can talk for hours about the exciting time I've spent with each and every one of them. Thanks for providing me with a home away from home.

During the last two years of my Ph.D., I volunteered as a resident assistant at Studentenwijk Arenberg, the largest group of KU Leuven-owned buildings for student housing. I'd to thank Els Bruyninckx and Steven Timmermans for their continuous support, as well as for selecting me in the first place. Thanks to all the other resident assistants of the Heverlee Area Residences: Alexander, Andreas, Bram, Charlotte, Gladys, Harm, Lies, Liesbeth, Lode, Lotte, Michiel, Olivier, Pascale, Rebecca, Romina, Stefaan, Stefanie, Stein, Vincent and Wim. I've learned a lot from you. Teaming up with you to organize parties, quizzes, barbecues and much more made my job an amazing experience. Thanks as well to all the 110 inhabitants, in particular those who didn't make too much noise, kept the kitchens clean and didn't forget their keys too often.

Last but not least, I would like to thank my parents and grandparents, as well as my brothers Yannick and Dylan. They never knew exactly what I was doing — which is understandable: most of the time, I didn't even know it myself. I know that I've caused many sleepless nights after saying things such as "Hey mom, dad, I'm going to China for a few months." Still, my family always loved and supported me. For that, I'll be forever grateful.

Nicky Mouha
Heverlee, June 2012

The main purpose of science is simplicity, and as we understand more things, everything is becoming simpler.

Edward Teller

Abstract

Cryptography is the study of mathematical techniques that ensure the confidentiality and integrity of information. This relatively new field started out as classified military technology, but has now become commonplace in our daily lives. Cryptography is not only used in banking cards, secure websites and electronic signatures, but also in public transport cards, car keys and garage door openers.

Two building blocks in the domain of cryptography are *block ciphers* and *(cryptographic) hash functions*. Block ciphers use a secret key to transform a plaintext into a ciphertext, in such a way that this secret key is needed to recover the original plaintext. Hash functions transform an arbitrary-length message into a fixed-length hash value. These hash values can serve as "fingerprints" for the original messages: it should be infeasible to find two distinct messages with the same hash value (a collision).

Yet, Wang et al. recently showed that finding collisions is feasible for MD5 and SHA-1, two of the most commonly used hash functions today. Although the SHA-2 family currently remains unbroken, its design is very similar. For this reason, the United States National Institute of Standards and Technology (NIST) launched an international competition for a new hash function standard: SHA-3.

The research performed in this Ph.D. thesis closely follows the evaluation period of the SHA-3 competition. Results were obtained for hash functions ARIRANG, BLAKE, ESSENCE, Hamsi, Khichidi-1, LUX, Sarmal, Skein and TIB3. Outside of the competition, results were also obtained for a simplified version of the hash function HAS-V. In the area of cryptographic theory, observations were made on the resistance of regular hash functions against the birthday attack.

The most commonly used hash functions: MD5, SHA-1 and SHA-2, as well two out of the five SHA-3 finalists (BLAKE and Skein) use operations such as addition modulo 2^n, exclusive OR, bitwise Boolean functions, bit shifts and bit rotations. Dissatisfied with commonly used *ad hoc* techniques to analyze such constructions, we introduced the framework of S-functions to allow for a simple and automated analysis.

Recently, *meet-in-the-middle* attacks became a very popular way to analyze block ciphers and hash functions. We constructed a novel variant of this technique, and applied it in an automated way to the block ciphers XTEA and GOST. Our attacks require very few known plaintext-ciphertext pairs.

Automated “black box” techniques, such as SAT solvers or Gröbner basis computations, have become increasingly sophisticated and powerful. In the domain of algebraic cryptanalysis, they are used to attack cryptosystems. What are the limits of these techniques? We revisited the differential-algebraic attacks of Albrecht and Cid, and showed that their attacks do not perform better than differential cryptanalysis. As a result, it seems that there is currently no efficient symmetric-key cipher that can be broken faster using algebraic techniques than using conventional techniques.

But not all hope in automatic solvers is lost. We showed how MILP (Mixed-Integer Linear Programming) solvers can be used to prove the security of ciphers against linear and differential cryptanalysis. Our technique involves writing out only simple linear inequality constraints, and therefore significantly reduces the workload of cryptanalysts and the probability of making of errors. We applied our technique to the Enocoro-128v2 stream cipher and to the block cipher AES, and illustrated how we can prove the security of these ciphers against linear and differential cryptanalysis in less than five minutes using an off-the-shelf solver.

Het belangrijkste doel van de wetenschap is eenvoud, en naar gelang we meer dingen begrijpen, wordt alles eenvoudiger.
Edward Teller

Samenvatting

Cryptografie is de studie van wiskundige technieken die de geheimhouding en integriteit van informatie waarborgen. Dit relatief nieuw studiegebied begon als geheime militaire technologie, maar is nu een alledaags iets geworden. Cryptografie wordt niet alleen gebruikt in bankkaarten, beveiligde websites en elektronische handtekeningen, maar ook in chipkaarten voor openbaar vervoer, autosleutels en de garagedeuropeners.

Twee bouwstenen in de cryptografie zijn *blokcijfers* en *(cryptografische) hashfuncties*. Blokcijfers gebruiken een geheime sleutel om een klaartekst om te zetten in een cijfertekst, zodanig dat deze geheime sleutel nodig is om de oorspronkelijke klaartekst te achterhalen. Hashfuncties transformeren een bericht van een willekeurige lengte naar een hashwaarde van een vaste lengte. Deze hashwaarden kunnen dienen als "vingerafdrukken" voor de oorspronkelijke berichten: het moet onhaalbaar zijn om twee verschillende berichten te vinden met dezelfde hashwaarde (een botsing).

Toch toonden Wang e.a. recent aan dat het haalbaar is om botsingen te vinden voor MD5 en SHA-1, twee van de meest gebruikte hashfuncties op dit moment. Hoewel de SHA-2 familie momenteel niet gekraakt is, is deze op een zeer gelijkaardige manier ontworpen. Om deze reden organiseerde het National Institute of Standards and Technology (NIST) van de VS een internationale wedstrijd voor een nieuwe hashfunctiestandaard: SHA-3.

Het onderzoek dat voor deze doctoraatsthesis uitgevoerd werd, volgt van dichtbij de evaluatiefase van de SHA-3-wedstrijd. Resultaten werden behaald voor hashfuncties ARIRANG, BLAKE, ESSENCE, Hamsi, Khichidi-1, LUX, Sarmal, Skein en TIB3. Buiten deze wedstrijd werden ook resultaten behaald voor vereenvoudigde versie van de hashfunctie HAS-V. Op het gebied van de theorie van cryptografie werden opmerkingen gemaakt bij de bestandheid van reguliere hashfuncties tegen de verjaardagsaanval.

De meest gebruikte hashfuncties: MD5, SHA-1 en SHA-2, evenals twee van de vijf SHA-3 finalisten (BLAKE en Skein) gebruiken bewerkingen zoals optelling modulo 2^n, exclusieve OR, bitsgewijze Booleaanse functies, bitverschuivingen en bitrotaties. Ontevreden over de veelgebruikte *ad hoc* technieken om dergelijke constructies te analyseren, introduceerden we het raamwerk van S-functies, die een eenvoudige en geautomatiseerde analyse mogelijk maken.

Recent werden *ontmoeting-in-het-midden*-aanvallen een zeer populaire manier

om blokcijfers en hashfuncties te analyseren. We construeerden een nieuwe variant van deze techniek, en pasten deze toe op een geautomatiseerde manier op de blokcijfers XTEA en GOST. Onze aanvallen vereisen slechts een zeer beperkt aantal gekende klaartekst-cijfertekstparen.

Geautomatiseerde "zwarte doos"-technieken, zoals algoritmen voor het oplossen van het vervulbaarheidsprobleem of het berekenen van een Gröbnerbasis, worden steeds gesofistikeerder en krachtiger. In het gebied van de algebraïsche cryptanalyse worden ze gebruikt om cryptosystemen aan te vallen. Wat zijn de grenzen van deze technieken? We herbekeken de differentieel-algebraïsche aanvallen van Albrecht en Cid, en toonden aan dat hun aanvallen het niet beter doen dan differentiële cryptanalyse. Hierdoor lijkt er momenteel geen efficiënt symmetrischesleutel cryptosysteem te bestaan dat sneller gebroken kan worden met behulp van algebraïsche technieken dan met conventionele technieken.

Maar niet alle hoop in automatische solvers is verloren. We hebben aangetoond hoe algoritmen die het MILP (gemengd geheeltallig programmeringsprobleem) oplossen, gebruikt kunnen worden om de veiligheid van cijfers tegen lineaire en differentiële cryptanalyse bewijzen. Onze techniek vereist enkel het uitschrijven van eenvoudige lineaire beperkingen, en vermindert dus de werklast van cryptanalisten en de kans dat een fout gemaakt wordt. We pasten onze techniek toe op het stroomcijfer Enocoro-128v2 en op het blokcijfer AES, en toonden hiermee aan hoe we de veiligheid van deze algoritmen tegen lineaire en differentiële cryptanalyse in minder dan vijf minuten kunnen bewijzen met een standaard softwarepakket.

Contents

List of Figures

List of Tables

List of Symbols

n	Length of a word in bits
N	Length of the hash value in bits, or the block size in bits
B	Block length in bits for iterated hash functions
P	Plaintext
C	Ciphertext
K	(Secret) Key
$\mathcal{P}$	Plaintext space
$\mathcal{C}$	Ciphertext space
$\mathcal{K}$	Key space
$E(P, K)$	Encryption of plaintext P under key K
$D(C, K)$	Decryption of ciphertext C under key K
$\Phi(K)$	Function that computes a related key $\Phi(K)$ from a secret key K
$x \parallel y$	Concatenation of the binary strings x and y
$x \wedge y$	Bitwise AND of x and y
$x \vee y$	Bitwise OR of x and y
$x \oplus y$	Bitwise XOR of x and y
$\neg x$	Bitwise NOT of x
$\lvert x \rvert$	Absolute value of x: $\lvert x \rvert = \sqrt{x^2}$
$\lvert A \rvert$	Number of elements of set A
$x \ll s$	Shift of x to the left by s positions
$x \gg s$	Shift of x to the right by s positions
$x \lll s$	Rotation of x to the left by s positions
$x \ggg s$	Rotation of x to the right by s positions
$x + y$	Addition modulo 2^n (in text), if the context clarifies that x and y are n-bit words
$x \boxplus y$	Addition modulo 2^n (in figures)
$x - y$	Subtraction modulo 2^n (in text), if the context clarifies that x and y are n-bit words
$x \boxminus y$	Subtraction modulo 2^n (in figures)
$x[i]$	Selection: bit (or element) at position i of word x, where $i = 0$ is the least significant bit (element)
$x[j \dots i]$	Select bits k where $j \geq k \geq i$, $k = 0$ is the LSB
0^k	Concatenation of k times the string '0'

List of Abbreviations

AES	Advanced Encryption Standard
API	Application Programming Interface
ARX	Addition modulo 2^n, Bit Rotation and XOR
CA	Certificate Authority
COPACOBANA	Cost-Optimized Parallel Code Breaker
COSIC	Computer Security and Industrial Cryptography
CP	Chosen Plaintext
CPU	Central Processing Unit
DES	Data Encryption Standard
ESAT	Electronics, Systems, Automation and Technology
FPGA	Field-Programmable Gate Array
FSM	Finite State Machine
FWO	Fonds Wetenschappelijk Onderzoek – Vlaanderen
GB	Gigabyte
GHz	Gigahertz
HAS	Hash Function Algorithm Standard
HTTPS	Hypertext Transfer Protocol Secure
IBM	International Business Machines Corporation
IFF	Identify Friend or Foe
ILP	Integer Linear Programming
IMAP	Internet Message Access Protocol
ISO	International Organization for Standardization
IV	Initial Value
IWT	Agentschap voor Innovatie door Wetenschap en Technologie
KCDSA	Korean Certificate-Based Digital Signature Algorithm
KP	Known Plaintext
L-characteristic	Linear Characteristic
LP	Linear Programming
LSB	Least Significant Bit
ls-regular	Linear Subset Regular
MD4	Message Digest Algorithm 4
MD5	Message Digest Algorithm 5

MDS	Maximum Distance Separable
MILP	Mixed-Integer Linear Programming
MitM	Meet-in-the-Middle Attack
MSB	Most Significant Bit
NIST	National Institute of Standards and Technology
NL-characteristic	Non-Linear Characteristic
PC	Personal Computer
Ph.D.	Doctor of Philosophy
PIN	Personal Identification Number
PolyBoRi	Polynomials over Boolean Rings
POP3	Post Office Protocol 3
RACE	Research and Development in Advanced Communications Technologies in Europe
RAM	Random access memory
RFID	Radio Frequency Identification
RIPEMD	RACE Integrity Primitives Evaluation Message Digest
RK	In a Related-Key Setting
RSA	Algorithm by Rivest, Shamir and Adleman
SAC	Strict Avalanche Criterion
SAT	Boolean Satisfiability Problem
S-box	Substitution Box
S-function	State Function
SHA	Secure Hash Algorithm
SMS	Short Message Service
s-regular	Subset Regular
SSL	Secure Sockets Layer
TEA	Tiny Encryption Algorithm
TLS	Transport Layer Security
UNAF	Unsigned Non-Adjacent Form
US	United States
XOR	Exclusive OR

Part I

Automated Techniques for Hash Function and Block Cipher Cryptanalysis

Chapter 1

Introduction

1.1 Motivation

When is a system secure? Although the question may seem easy, the answer turns out to be very difficult.

Take a mobile phone for example. It's quite straightforward to test if the device works correctly. You could try to make and receive phone calls, and do this under a wide range of conditions: in flat regions and in mountainous areas, standing still and in high-speed trains, in urban and in rural areas. If a problem occurs, it can typically be isolated and resolved: maybe the antenna is malfunctioning, the display is defective, a button is broken,...

But is the phone secure? For example, is it possible for attackers to listen in on your conversations? Can an attacker make phone calls or send SMS messages and charge them to your account? When your phone is stolen, can it be used without knowing the PIN? Such questions are much more difficult to answer, if they can be answered at all. Of course, experts can perform a security analysis to find and resolve several vulnerabilities. But in many cases, problems are only revealed after a successful attack. Or even much later, if it is not immediately apparent that the security has been breached.

And attacks do happen. Here are just a few examples from 2011:

A SecurID token is a small device that displays numbers that change every minute. For a user to get access into a system – for example a website for on-line banking – he or she must enter both a password or PIN and the number on the SecurID token's screen. On March 17, 2011, RSA publicly disclosed that attackers broke into their systems and obtained secret information about the company's SecurID product [43]. This information was subsequently used to break into systems of US government defense contractors Lockheed Martin and L-3 Communications. EMC, the parent company of RSA, revealed that they had spent 50 million euros in the second quarter to deal with the cyber attack [163].

On April 19, 2011, Sony discovered that attackers broke into Sony's PlayStation Network, and obtained for all 77 million registered accounts (amongst other information) the names, addresses, date of births, logins and passwords, and possibly even credit card numbers and expiration dates of their users [72]. Following the security breach, Sony shut down the PlayStation Network on April 20, resulting in a worldwide outage until May 15 [157]. Sony estimated the cost associated with the unauthorized network access at about 138 million euros [156].

On August 28, 2011, an Iranian citizen reported that when he tried to access his Gmail account using the Google Chrome browser, an error message indicated that the certificate was invalid [5]. According to the browser, the certificate was signed by DigiNotar. But starting from Chrome 13, Google had implemented *public key pinning* [97]: certificates from all but a small number of certificate authorities are rejected for Google websites. For this reason, Chrome rejected DigiNotar's certificate. Google subsequently removed DigiNotar from Chrome's list of trusted root certificates, other browser manufacturers quickly followed suit. These events forced DigiNotar into bankruptcy, and caused its parent company VASCO estimated losses between 2.5 million to 3.7 million euros [167].

These are just a few examples of recent high-profile attacks, and mention nothing about the widespread occurrence of botnets, phishing, viruses, denial-of-service attacks,... Attacks are becoming increasingly sophisticated, with a goal of either monetary gain or the collecting of strategic information. Securing the world's information is therefore one of the biggest challenges of the 21st century.

1.2 Challenges

Information security is a very broad domain, in which every aspect of protecting information and information systems is studied. The field of cryptography is concerned with mathematical techniques to secure information. An extended overview of these techniques can be found in the "Handbook of Applied Cryptography" [113].

Interestingly, the high-profile attacks described in the previous section are not the result of weaknesses in cryptographic building blocks. This does not mean that weak cryptography never leads to practical attacks. Counterexamples include:

- the attack on the KeeLoq block cipher used to secure the car keys of several leading automakers [79],
- the attack on the Crypto-1 stream cipher used in many public transportation systems [63], and
- the attack on the MD5 hash function which lead to the creation of a rogue CA certificate that could impersonate any website secured by HTTPS [158].

Recent attacks indicate that although good cryptographic algorithms are necessary to secure our digital information, they are not sufficient. We now provide

a brief and incomplete overview of other considerations that should be taken into account.

Even when cryptographic algorithms and protocols are secure, they may be implemented in an insecure way. If this is the case, the secret key may leak through a side channel attack. Possible side channels include timing information [92], power consumption [93] and electromagnetic radiation [62].

For example, a cache-timing attack allows a key recovery attack for a common implementation of the AES block cipher [16]. Another implementation problem can be found in the domain of RFID tags, which can be used as "electronic bar codes" to trace products. A variety of protocols exist to provide privacy for RFID tags, see [71] for a recent overview. However, there is no privacy anymore if every RFID tag has a unique physical-layer fingerprint [186].

Users are known to choose weak passwords, and use the same few passwords again for all their accounts [152]. Many people use unpatched software, leaving the door open for attackers [143]. Users may even be too confused about security software to make use of it at all [183]. Or what if the attacker can trick the user in some way, in order to gain access to their accounts? It seems that the user is often the weakest link in the security chain. This should not be overlooked when designing a secure system.

But even if cryptography is not always the problem, it can be part of the solution. The problem of weak user passwords can be mitigated by introducing two-factor authentication: requiring not only a password, but also a security code from a cryptographic device (such as the SecurID mentioned earlier). Cryptographers should also design algorithms and protocols with the possibility of a secure implementation in mind. Lastly, it is important that cryptographic components are easy to understand and analyze.

And that's exactly the research focus of this thesis: how to analyze cryptographic components in an easy way. Our approach is as follows:

- A number of cryptographic algorithms are analyzed. Mostly hash functions submitted to NIST's SHA-3 competition [126], but several other ciphers as well.
- Breaking cryptosystems gives an insight into which design principles are secure, and which are not. From this, general principles for design and cryptanalysis can be distilled.
- These insights are then used to build general frameworks that allow hash functions, block ciphers and stream ciphers to be analyzed.
- A special focus is put on automated techniques. Not only because they are easier to use and understand, but also because they have a much broader range of applicability.
- To maximize the impact of the research, the tools that we have developed are publicly available.

1.3 Thesis Outline

This Ph.D. thesis is based on publications, and consists of two parts. The first part consists of five chapters. In this chapter, we motivated the need for security and cryptography, and situated the research performed in this Ph.D. thesis. Cryptographic hash functions are defined in Chapter 2. We describe which security properties they should satisfy, and discuss several ways to construct a hash function. Our analysis of ten different hash functions is presented in Sect. 2.6.

Block ciphers are described in Chapter 3. After defining the concept of a block cipher and considering relevant attack models, we present meet-in-the-middle attacks on the block ciphers XTEA (Sect. 3.3.1) and GOST (Sect. 3.3.2). Our attacks work in the single-key setting, and require very few known plaintexts.

Chapter 4 investigates automated techniques for the cryptanalysis of block ciphers and hash functions. The focus is not on particular ciphers, but on methods that can be applied in general.

We introduce the framework of S-functions in Sect. 4.3, and show how to efficiently calculate their properties with respect to differential cryptanalysis.

Another way to analyze ciphers in an automated way, is by representing ciphers using systems of equations, and solving these using algebraic techniques. The differential-algebraic attacks of Albrecht and Cid are investigated in Sect. 4.4. We explain why their techniques do not perform better than standard differential cryptanalysis, using both theoretic insights and numerous computer experiments.

Linear and differential cryptanalysis are two of the most powerful techniques in symmetric-key cryptanalysis. In Sect. 4.5, we show how to easily prove security against these attacks, both in single-key and related-key models.

We conclude in Chapter 5, where we also provide directions for future research.

A selection of our publications can be found in the second part of this thesis, where eight publications are reproduced in their entirety as separate chapters. A full list of publications can be found on p. 63.

Chapter 2

Hash Functions

2.1 Introduction

Hash functions turn an arbitrary-length message into a short, fixed-length "fingerprint" of this message. Originally, the main objective of hash functions was to provide data integrity: making sure that data was not modified by unauthorized or unknown means. Nowadays, hash functions have become the "Swiss army knife of cryptography": they are used not only for data integrity but also for commitment schemes, key derivation, pseudorandom number generation,...

This chapter defines hash functions, and explains the security properties that they should satisfy. For a more detailed treatment of hash functions, we refer to Preneel [136]. We have obtained results for no fewer than ten hash functions.

Outline. Section 2.2 explains what hash functions are, and lists the standard security requirements that they have to fulfill. An overview of some additional security requirements is given in Sect. 2.3. We explore the theoretic foundation of cryptographic hash functions in Sect. 2.4, where we also include our own observations on regular hash functions. After introducing the concept of iterated hash functions in Sect. 2.5, we explain the Merkle-Damgård construction. Section 2.6 presents an overview of our cryptanalysis results. We discuss our results for the SHA-3 candidates ARIRANG, BLAKE, ESSENCE, Hamsi, Khichidi-1, LUX, Sarmal, Skein and TIB3. Outside of the SHA-3 competition, we analyzed the hash function HAS-V.

2.2 Definition

Hash functions are functions h that transform an arbitrary-length message m in a deterministic way into a fixed-length hash value $h(m)$. The algorithm of h should be efficiently computable. We use N to denote the length of $h(m)$ in bits. Throughout this thesis, "hash function" will always refer to a "cryptographic hash

function". The reader should not confuse cryptographic and non-cryptographic hash functions. The latter are used for example in hash tables [90].

Cryptographic hash functions should satisfy several security requirements, otherwise we refer to them as "broken". We will investigate these security requirements in detail. Every security requirement can be violated by a generic attack, i.e. an attack that applies to every hash function. The complexities of these generic attacks depend only on N, the length of the hash value. This explains the following subtlety: we do not say that it should be *impossible* to break the security requirements, but N should be designed to be large enough so that it becomes (computationally) *infeasible*.

Traditionally [113, 136], the security requirements are preimage resistance, second preimage resistance and collision resistance. We now define these notions in an informal way. In Sect. 2.4, we will discuss some criticisms [6, 141] on the informal definitions of the following sections.

2.2.1 Preimage Resistance

Hash functions should be one-way functions. This means that given a hash value Y, it should be infeasible to find a preimage, i.e. a message m for which $Y = h(m)$.

A generic attack to find a preimage goes as follows. Let us assume that the hash function can be modeled as a random function. Assume that a hash value Y is chosen uniformly at random from the set of all N-bit strings. As the length of the hash value is N, after 2^N hash function evaluations on a set of distinct messages R, the probability that a message $r \in R$ is found for which $h(r) = Y$ is

$$1 - \left(1 - \frac{1}{2^N}\right)^{2^N} . \tag{2.1}$$

As 2^N is typically very large, we can obtain a good approximation by using $\lim_{x\to\infty} \left(1 - \frac{1}{x}\right)^x = e^{-1}$. Therefore, the success probability of this generic preimage attack for hash functions is about $1 - e^{-1} \approx 63\%$.

2.2.2 Second Preimage Resistance

Second preimage resistance can be defined as follows. Given a message m and a hash value $h(m)$, it should be infeasible to find a second message m' ($m \neq m'$) for which $h(m) = h(m')$.

The notions of preimage resistance and second preimage resistance are closely related. A one-way function is both preimage and second preimage resistant, and the same generic attack applies to both security notions.

Yet, it is possible to construct hash functions that are second preimage resistant, but not preimage resistant [113, Note 9.20]. Therefore, preimage resistance and second preimage resistance are considered to be separate security notions.

2.2.3 Collision Resistance

A hash function is collision resistant if it is infeasible to find two distinct messages m, m' ($m \neq m'$) such that $h(m) = h(m')$. Collisions exist for every hash function because of the pigeonhole principle: a set of $2^N + 1$ messages will always contain a collision, because an N-bit hash function cannot output more than 2^N distinct hash values. However, a much more efficient attack exists. Again, we model the hash function as a random function.

As the length of the hash value is N, after $2^{(N+1)/2}$ hash function evaluations on a set of messages R, two messages $r, r' \in R$ where $r \neq r'$ are found for which $h(r) = h(r')$ with a success probability of about $1 - e^{-1} \approx 63\%$.[1] This generic collision attack against hash functions is referred to as the birthday attack. Its name refers to the birthday paradox, which states that in a room with 23 people, there is a probability of more than 50% that two people share the same birthday.

It is not necessary to store all messages and hash values. Quisquater and Delescaille [138] showed that collisions for meaningful messages can also be found with negligible memory requirements. An efficient parallel implementation of their algorithm was proposed by Van Oorschot and Wiener [164].

Collision resistance implies second-preimage resistance, but does not imply preimage resistance. The same counterexample given in [113, Note 9.20] illustrates this.

2.3 Other Security Requirements

The security requirements of a hash function strongly depend on the application. In some applications, collision resistance is not required. Feldhofer and Rechberger surveyed nine hash-function based RFID protocols, and found that only one protocol required collision resistance [59].

For applications where collision resistance is required, N is by design chosen to be large enough to thwart collision attacks: performing about $2^{(N+1)/2}$ hash function evaluations should be computationally infeasible. In this case, it does not make sense to require a hash function to have a preimage resistance with a complexity of more than $2^{(N+1)/2}$.

This observation lead to an efficiency improvement of the Lesamnta-LW hash function [73]. For the SHA-3 competition, NIST made 2^N preimage resistance a design requirement [126]. If this were not the case, the SHA-3 finalist hash function Keccak [21] would also have been designed more efficiently [44].

In other applications, hash functions require more than preimage, second preimage and collision resistance. An application may truncate an N-bit hash value to

[1] We choose a set of $2^{(N+1)/2}$ messages, so that the success probability of the collision attack is the same as that of the preimage and second preimage attacks described in this section. For a set of $2^{N/2}$ messages, the success probability would be about $1 - e^{-1/2} \approx 39\%$. These success probabilities are calculated using Eq. (B.4) of [136, § B.3].

M bits, where $M < N$. In this case, there should be no attacks better than the generic attacks for M-bit hash functions.

Another additional requirement is resistance against length extension attacks (also known as message expansion attacks). In a length extension attack, the attacker knows $h(m)$ and the length of m, but not m itself. The attacker chooses a value x in a specific way, and can efficiently calculate the hash value $h(m \parallel x)$. This length extension attack is applicable to all commonly used hash functions: MD5, SHA-1, SHA-2.

Not all implementers are aware of this attack, as evidenced by a recent security vulnerability in an API released by Flickr [57], one of the most popular photo sharing websites. Resistance against length extension attacks is a design requirement for SHA-3 candidates [126], and according to Kelsey of NIST [84], this may be the most important reason to recommend implementers to switch to SHA-3.

2.4 Theory of Hash Functions

In the field of provable security, the security of cryptographic protocols and applications is proven based on the security of the underlying cryptographic primitives. For this, a more rigorous mathematical foundation of hash functions is required.

From a theoretic point of view, the ideal hash function against all attacks is a *random oracle* [15]. For every message, a random oracle chooses the corresponding hash value independently and uniformly at random. This is done in such a way, that consistent hash values are given if an earlier message is encountered.

In [14, 37], Bellare and Kohno argue that "regular functions fare better than random functions [against the birthday attack]." A hash function is regular if every hash value has the same number of preimages in the domain. In [122] (see p. 181), we counter their arguments by showing that the success probability of the birthday attack against a regular hash function can be made arbitrarily close to that of a random hash function (for the same number of trials).

One issue with Bellare and Kohno's result is the following. Consider an N-bit hash function that simply outputs the first N bits of the message as the hash value. This hash function is a regular hash function by construction. Assume that the domain D consists of all messages of a certain fixed length. Clearly, performing the birthday attack as defined by Bellare and Kohno (Fig. 1 of [14]) will have a complexity of $2^{(N+1)/2}$ for a success probability of about 63%. The reason is that messages in their birthday attack are chosen uniformly at random from the domain D.

However, Bellare and Kohno also note in the same paper that "there are several variants of [the birthday attack] which differ in the way the [messages] $x_1, \ldots, x_q$ are chosen." Our example hash function will clearly be very weak against a variant of the birthday attack, where messages are chosen such that the first N bits are the same. Therefore, we also point out in [122] (see p. 181) that contrary to the birthday problem, the distribution of the hash values in the birthday attack do

not only depend on the hash function, but also on how the attacker chooses the domain points.

A random oracle is a very useful tool to construct theoretic security proofs, but cannot be implemented as an efficient function [38]. When Rogaway and Shrimpton attempted to formalize the concept of real-world hash functions (i.e. functions that can be efficiently evaluated), two main problems arose [141].[2]

Firstly, for certain hash functions it may be easy to find preimages for some hash values, but difficult for other hash values. To address this issue, Rogaway and Shrimpton consider both preimage security against random hash values, as well as against fixed hash values. If finding (second) preimages is difficult for every hash value, the hash function is said to be "everywhere" secure.

Secondly, we want to state more formally that it should be infeasible to find a collision. But we cannot do this by saying that no efficient algorithm exists. These exist for every hash function: for example, programs exist that simply contain a hard coded collision, although it may be very difficult to find such programs. Rogaway made an attempt to sidestep this issue by claiming that "human ignorance" prevents finding these programs [140]. However, a security proof based on human limitations is obviously less rigid than a proof based on information theory or complexity theory.

A related issue is that it is very easy to distinguish an efficient hash function (for example, SHA-1) from a random oracle. An attacker could take a random message r and query the hash value. If the hash value is SHA-1(r), the attacker concludes that the hash function is not a random oracle. The algorithm to test for non-random behavior is referred to as a *distinguisher*. For the distinguisher that we just described, the error probability is 2^{-160}.

Rogaway and Shrimpton used keyed hash functions to address some of the problems mentioned earlier. Confusingly, the term "key" is not used in this context for something secret and unknown to the attacker, but as an index of one particular hash function out of a family of hash functions. This key can either be fixed by the attacker or chosen at random. If a keyed hash function is (second) preimage secure for all keys (i.e. every hash function in the hash function family is (second) preimage secure), then the keyed hash function is referred to as "always" secure.

Yet, the most commonly used hash functions (MD5, SHA-1 and SHA-2) are not designed with the concept of keys in mind, and most protocols using hash functions do not make use of keys. This is where this theoretical framework falls short.

2.5 Iterated Hash Functions

Hash functions transform a variable-length message into a fixed-length hash value. This is commonly done by padding the message and chopping it into fixed-length

[2]Note that the claim by Rogaway and Shrimpton in [141] that "ePre $\rightarrow$ Pre" was later shown to be incorrect by Andreeva and Stam [6].

message blocks. Most hash functions are iterated hash functions. They use a chaining value, and update the chaining value by iteratively processing every message block. This has the advantage that the message does not have to be stored into memory.

A disadvantage is that iterated hash functions can have *state collisions*, i.e. collisions in the chaining value. This has several interesting consequences.

- If padded messages m and m' ($m \neq m'$) result in a state collision for a given hash function h, then this results not only in $h(m) = h(m')$, but also $h(m \parallel x) = h(m' \parallel x)$ for any suffix x. This observation allows a hash function to be distinguished from a random oracle.
- Given two hash functions h and g, a message m may be transformed in to the concatenated hash value $h(m) \parallel g(m)$. If f and g are iterated hash functions, the concatenated hash value is not much more secure than either f or g used separately. This statement applies to preimage, second preimage and collision resistance [82].
- For long messages, iterated hash functions allow second preimages to be found in much less than 2^N hash function evaluations [85].

In the next section, we will focus on the construction of iterated hash functions using the Merkle-Damgård construction. Note that alternatives exist, such as for example the sponge construction [20].

2.5.1 Merkle-Damgård construction

The Merkle-Damgård construction uses a fixed-length compression function $f : \{0,1\}^{B+N} \rightarrow \{0,1\}^N$ to construct an arbitrary-length hash function $h : \{0,1\}^* \rightarrow \{0,1\}^N$.

First, the message m is padded to a multiple of the block length B, by adding the bit '1' followed by zero or more times the bit '0':

$$m_1 \parallel \ldots \parallel m_L \leftarrow m \parallel 10\ldots0 \ , \tag{2.2}$$

where $\forall 1 \leq i \leq L : |m_i| = B$.

The initial value IV is fixed by design and independent of the message m. The hash value $h(m)$ is then obtained as follows (see also Fig. 2.1):

$$H_0 \leftarrow \text{IV}, \tag{2.3}$$

$$H_i \leftarrow f(m_i \parallel H_{i-1}), \qquad i = 1, 2, \ldots, L, \tag{2.4}$$

$$h(m) \leftarrow f(|m| \parallel H_L). \tag{2.5}$$

The name of this construction refers to the work of Merkle [114] and Damgård [49]. Lai and Massey introduced the term "Merkle-Damgård strengthening" [96] for

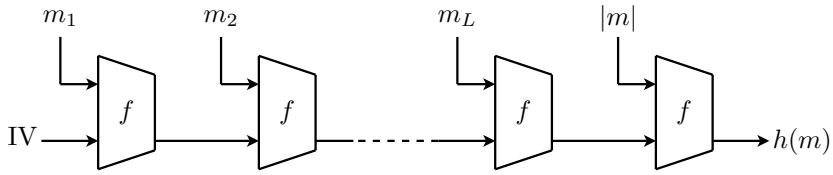


Figure 2.1 – Merkle-Damgård construction

the process of adding the message length in this construction. By adding the message length, it is possible to prove for the Merkle-Damgård construction that if a collision is found for the hash function h, a collision for the compression function f can be found as well. It is therefore possible to prove the collision resistance of h, based on the collision resistance of f [49, 114].

Most commonly used hash functions use the Merkle-Damgård construction, including the hash functions MD4, MD5, SHA-1 and SHA-2. From the description of the construction, it now becomes evident why the length extension attack described in Sect. 2.3 applies to all of these hash functions. Length extension attacks can be prevented in several ways, for example by processing the last message block in a special way [22].

2.6 Analysis of Hash Functions

2.6.1 Introduction

In the early 2000s, there was a good understanding in the field of cryptography of the design and cryptanalysis of block ciphers. The US National Institute of Standards and Technology (NIST) had selected Rijndael as the winner of the Advanced Encryption Standard (AES) competition [47]. This block cipher has a simple mathematical structure, and its wide-trail design results in provable security against both linear and differential cryptanalysis.

Yet not much was known about the secure construction of hash functions. In 2005, attacks by Wang et al. on the hash functions MD4 [175], MD5 [177], RIPEMD [175] and SHA-1 [176] gave a new impulse to the research of hash functions. For the widely used MD5, only a few milliseconds on a PC are sufficient to find collisions [160]. Finding collisions is feasible for SHA-1 as well, although a lot more computing power is required [51]. At this time of writing, SHA-2 remains unbroken, but it is based on the same design principles.

NIST acknowledged the attacks by Wang et al. and decided to take action. In 2007, they launched a competition for a new hash function standard: the SHA-3 competition [126]. Cryptographers from all over the world were asked to submit their hash function proposals. A total of 64 submissions were made, of which 56 were made publicly available. On December 10, 2008, NIST announced the

51 submissions that advanced to the first round. This was the starting shot of a four-year public evaluation period. Fourteen submissions advanced to the second round, and five candidates reached the final round. The winner will be selected in 2012.

The research in this Ph.D. thesis was performed in parallel to the SHA-3 competition. The next sections give a list of hash functions for which contributions were made. Some of these hash functions were broken as a result of our efforts, and therefore did not advance further into the competition. For the other hash functions, a better understanding of the inner workings can be gained from our analysis.

2.6.2 ESSENCE

ESSENCE [107], a family of cryptographic hash functions designed by Martin, was accepted into the first round of the SHA-3 competition. ESSENCE includes a security proof against linear and differential cryptanalysis. Until our analysis [121] (see p. 95), this security proof remained unchallenged. Our results presented the first known attacks on ESSENCE.

In [96], Lai and Massey introduced the concept of a semi-free-start collision for a hash function. For a semi-free-start collision, the IV is under control of the attacker, but the same IV must be used for messages m and m'.

For 31 out of 32 rounds of ESSENCE-512, we present a semi-free-start collision attack. This attack invalidates the design claim that at least 24 rounds of ESSENCE are secure against differential cryptanalysis. To satisfy the first nine rounds of the differential characteristic, we developed a novel technique.

We also constructed several distinguishers on a 14-round ESSENCE block cipher and the corresponding compression function, by exploiting non-randomness in the outputs of the feedback function F. Each distinguisher requires only 2^{17} output bits. This observation was extended to key-recovery attacks on the corresponding ESSENCE block cipher.

Furthermore, we explained that the omission of round constants allows slid pairs and fixed points to be found, independent of the number of rounds. As a result of our attack, ESSENCE did not advance to the second round of the SHA-3 competition [108]. Independent results on ESSENCE were later obtained by Naya-Plasencia et al. [128]

2.6.3 Khichidi-1

Also included in the first round of the NIST SHA-3 competition, was the hash function Khichidi-1 [170], designed by Tata Consultancy Services. This company is the largest IT services firm in Asia, with a revenue of 6.35 billion euros for the 2011 fiscal year [161]. Only 3 hours and 44 minutes after the algorithm was made public, a collision for the Khichidi-1 hash function was sent to NIST as an official comment [117]. This collision was the result of a second preimage attack. For

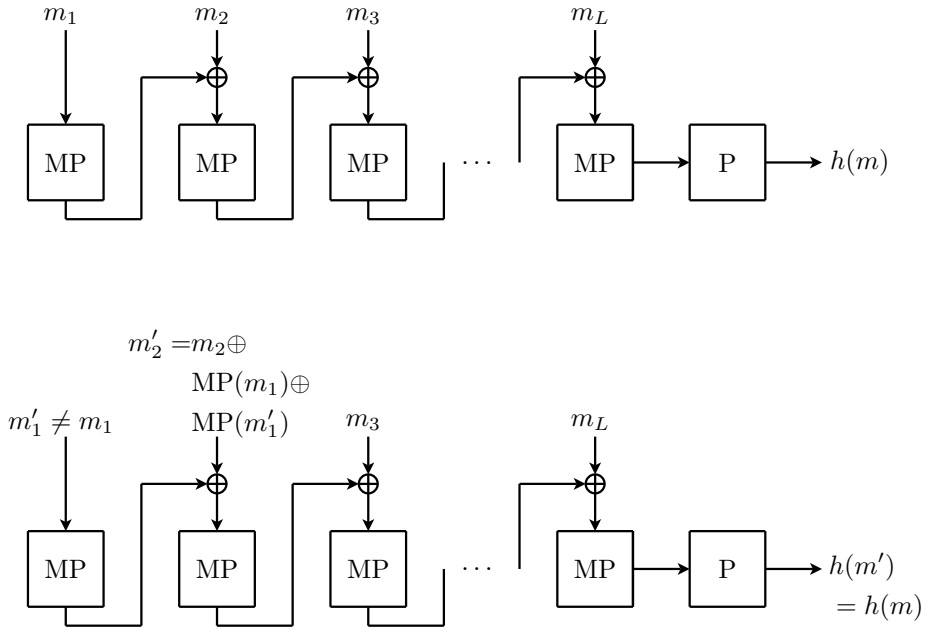


Figure 2.2 – Constructing a second preimage m' for Khichidi-1 (m: an arbitrary message, MP: "Message Preprocessing" function, P: post-processing function)

Khichidi-1, a *correcting block attack* applies [136, § 2.5.2.4]. To construct message m', all blocks of message m can be substituted without any restrictions, except for some block m_j. To construct m'_j, the XOR operation is inverted to ensure that $h(m) = h(m')$. This process is illustrated in Fig. 2.2. As a result of this attack, the designers withdrew their submission from the SHA-3 competition.

2.6.4 LUX

Another first-round SHA-3 candidate was LUX, a design by Nikolić et al. [130]. LUX has a byte-oriented design. It is inspired by the RadioGatún hash function [19], but reuses components of AES. Let LUX-N denote the LUX hash function with an N-bit output, where $N \in \{224, 256, 384, 512\}$.

Wu et al. discovered that all LUX-256 hash values contain redundancy [185]. An algorithm can therefore be constructed that distinguishes a LUX hash value from random with a high success probability, regardless of the message. Wu et al.'s results turned out to be incorrect, presumably because of calculation mistakes. We corrected the results, extended them to 224-, 384- and 512-bit hash value lengths and publicly released a software program to verify the results [118]. The

Table 2.1 – Preimage, second preimage and collision attacks against LUX; every attack has a success probability of about 63%

Attack Complexity	LUX-224	LUX-256	LUX-384	LUX-512
Preimage	2^{176}	2^{200}	2^{344}	2^{456}
Second Preimage	2^{176}	2^{200}	2^{344}	2^{456}
Collision	$2^{88.5}$	$2^{100.5}$	$2^{172.5}$	$2^{228.5}$

observation can be described as follows. We split the N-bit hash value into $K = N/8$ bytes:

$$h(m) = h_1 \parallel h_2 \parallel \ldots \parallel h_K \ . \tag{2.6}$$

For LUX-224 and LUX-256, the following equation holds for $0 \leq i \leq (N/32) - 2$:

$$\texttt{4f} \cdot S(h_{4i+4}) = h_{4i+5} + \texttt{26} \cdot h_{4i+6} + \texttt{9f} \cdot h_{4i+7} + \texttt{f7} \cdot h_{4i+8} \ . \tag{2.7}$$

As for AES, the multiplication and addition operations are performed in the finite field GF(2^8). The elements of GF(2^8) are considered to be polynomials with coefficients in GF(2), multiplication is done using the irreducible polynomial $m(x) = x^8 + x^4 + x^3 + x + 1$. In the formulas, the hexadecimal representation of the elements is used. For more details on the mathematical operations, we refer to [47, § 2.1].

Similarly, for LUX-384 and LUX-512, the following equation holds for $0 \leq i \leq (N/64) - 2$:

$$\begin{aligned} \texttt{53} \cdot S(h_{8i+8}) = h_{8i+9} + \texttt{c1} \cdot h_{8i+10} + \texttt{db} \cdot h_{8i+11} + \texttt{60} \cdot h_{8i+12} \\ + \texttt{17} \cdot h_{8i+13} + \texttt{af} \cdot h_{8i+14} + \texttt{5d} \cdot h_{8i+15} + \texttt{78} \cdot h_{8i+16} \ . \end{aligned} \tag{2.8}$$

For LUX-256, (2.7) provides seven equations, each involving eight bits. Therefore, only one out of $2^{7 \cdot 8} = 2^{56}$ hash values are possible for LUX-256. This allows us to distinguish the output of LUX based only on the hash value, with a very high success probability of $1 - 2^{-56}$. Using (2.7), it is straightforward to obtain a similar result for LUX-224. Results for LUX-384 and LUX-512 can be obtained using (2.8).

As observed by Watanabe [48] and Ferguson [60], this effectively reduces the strength of LUX-256 to $256 - 56 = 200$ bits. An overview of the complexity of preimage, second-preimage and collision attacks for all versions of LUX can be found in Table 2.1. The success probability of each of the attacks is about $1 - e^{-1} \approx 63\%$. The mechanisms of the generic attacks of Table 2.1 were explained in Sect. 2.2.1–2.2.3.

LUX was not selected for the second round of the SHA-3 competition.

2.6.5 Sarmal

Sarmal is a hash function submitted to the SHA-3 competition by Varıcı et al. [165]. The design and structure of Sarmal is quite similar to that of SHA-3 candidate ARIRANG [39]. Both are inspired by the hash function FORK-256 [75]. We analyzed the impact of recent attacks by Guo et al. on ARIRANG [70] with respect to Sarmal [119].

It appears that Sarmal is less vulnerable against these attacks. More specifically, we could not obtain pseudo-collisions for Sarmal faster than using a generic attack. However, we found that compression function of Sarmal can be distinguished from a pseudorandom function by using only two compression function calls. The result holds only for the compression function, and it seems not possible to extended it to the full hash function.

2.6.6 Skein and BLAKE

Five hash functions were selected as finalists for the SHA-3 competition: the hash functions BLAKE [10], Grøstl [64], JH [184], Keccak [21] and Skein [61].

Two of the finalists (BLAKE and Skein) have a core that consists of only three operations: addition modulo 2^n, bit rotation and exclusive-or (ARX). Although they are very fast in software, the security of ARX-based constructions is not well understood.

For the original Skein, the most successful attacks are based on rotational cryptanalysis. In [87], the concept of rotational cryptanalysis is formally explained, and applied to reduced rounds of the Threefish block cipher, the core of the Skein hash function. These results were further improved in [88], where cryptanalytical results were obtained for an estimated 53 rounds of Skein-256, and 57 rounds of Skein-512. In response to these rotational attacks, the designers changed the value of the key schedule constant.

To analyze the patched versions of BLAKE and Skein, we introduced a new type of cryptanalysis called tuple cryptanalysis [11]. Our technique is inspired by square [45], saturation [106], internal collision [65] and multiset cryptanalysis [28]. Multiset cryptanalysis, proposed by Biryukov and Shamir on the SASAS construction, is similar to tuple cryptanalysis: a tuple is an ordered list of (possibly repeating) elements, whereas a multiset is unordered.

Using tuple cryptanalysis, we obtained preliminary results for up to 9 rounds of Threefish-256, and up to 12 rounds of Threefish-512. Threefish is the block cipher core of the Skein hash function. We also obtained results for 4 rounds of the BLAKE hash function.

2.6.7 Other SHA-3 Results

Many other SHA-3 candidate algorithms were analyzed as well. For the hash functions TIB3 [115] and ARIRANG [39], our overlapping results were scooped

by Mendel et al. [112] and Guo et al. [70] respectively. This illustrates the strong interest and fierce international competition among cryptanalysts during the competition. In this section, we provide a brief overview of other notable results.

The 512-bit version of the hash function Sarmal [165] is vulnerable to the standard length extension attack described in Sect. 2.3, in violation of NIST's security requirements. The designers were notified of this problem, but did not update their implementation. Sarmal did not reach the second round of the SHA-3 competition.

Both the 384-bit and 512-bit versions of the second-round SHA-3 candidate Hamsi [94] had a very serious implementation problem in all submitted implementations, which was fixed after we notified the designer. Because of this bug, only half of the message bits were processed. The seriousness of this bug is illustrated by the following colliding message pair:

$$m = \text{“I will pay \$1.00 to Nicky Mouha.”}$$
$$m' = \text{“I will pay \$1000 to Nicky Mouha.'}$$

Some help was provided during the design phase of the hash functions LANE [78] and Hamsi [94]. A reference implementation was provided for LANE, and several corrections were made to the Hamsi design document.

2.6.8 HAS-V

The hash function HAS-V [133] was designed by Park et al. to meet the requirements of the KCDSA [100], a digital signature algorithm standardized by ISO.

The hash function HAS-V is not part of the SHA-3 competition. However, its structure is similar to that of recently broken hash functions MD4 [175], MD5 [177], RIPEMD [175] and SHA-1 [176]. The goal of the analysis of HAS-V was to get a better understanding of the underlying cryptanalytical techniques. Other published results on HAS-V [111, 150, 151] are all based on the observation that the HAS-V step function is non-injective.

In [51], De Cannière and Rechberger introduced a technique to cryptanalyze the SHA-1 hash function. We explained their method in further detail, generalized it, and applied it to a simplified version of HAS-V [120] (see p. 69). The reason for the simplification was because HAS-V processes two parallel streams: in order for the techniques to apply, one stream was left out.

We extended the technique of De Cannière and Rechberger by introducing *double conditions*, which are conditions involving not one, but two pairs of bits. These not only make it possible to calculate probabilities more accurately, but also help to find inconsistencies in the characteristics. Another extension is the inclusion of a fourth stage, not present in the analysis by De Cannière and Rechberger. Using our improvements, we constructed a collision attack on 45 out of 60 steps with a complexity of 2^{46} compression function evaluations.

2.7 Conclusion

Hash functions are often referred to as the "Swiss army knives" of cryptography because of their wide range of applications. It is not so straightforward to define the security requirements of a hash function: they strongly depend on the application and hash functions are sometimes used in unexpected ways.

For the purpose of data integrity, we defined the concepts of preimage, second preimage and collision resistance in Sect. 2.2. If we model the N-bit hash function as a random function, the complexities of these attacks are 2^N, 2^N and $2^{(N+1)/2}$ hash function evaluations respectively, where each attack has a success probability of about 63%.

Often, a hash function must meet additional security requirements. We gave a a brief overview of these in Sect. 2.3.

In Sect. 2.4, we described the concept of a random oracle as a theoretically ideal hash function. We also provide more insight into regular and random hash functions with respect to the birthday attack.

As explained in Sect. 2.5, most practical hash functions are iterated hash functions. This allows messages to be hashed "on the fly" instead of storing the entire message into memory.

Several hash functions were analyzed. Section 2.6 gives an overview. We obtained results for the SHA-3 candidates ARIRANG, BLAKE, ESSENCE, Hamsi, Khichidi-1, LUX, Sarmal, Skein and TIB3. As a result of our analysis, several hash functions did not advance in the SHA-3 competition. Outside of the SHA-3 competition, we also obtained cryptanalysis results for the hash function HAS-V.

Chapter 3

Block Ciphers

3.1 Introduction

Another building block in the domain of cryptography are block ciphers. Using a secret key, a block cipher transforms a plaintext into a ciphertext. This process is referred to as *encryption*. To recover the original plaintext (*decryption*), the secret key is needed. As the same secret key is used for both encryption and decryption, block ciphers fall into the field of symmetric-key cryptography. This is different from public-key encryption, where one key is used for encryption and another for decryption.

Block ciphers process blocks of a fixed length. To process plaintexts of an arbitrary length, a mode of operation is required. Most modes of operation cannot detect if the ciphertext has been modified by unauthorized means. If such functionality is required, an authenticated encryption mode can be used. We will not discuss such modes in this thesis. For a detailed overview of modes that are standardized by NIST, we refer to [127].

The first block cipher that became widely used, was the Data Encryption Standard (DES) [125]. DES was standardized by the US National Bureau of Standards, now known as NIST. It has a key length of 56 bits, and processes 64-bit blocks.

Today, DES should be considered completely insecure for most purposes because of its short key length. Kumar et al. showed in 2006 that for a cost of less than $10,000, an exhaustive key search for DES can be performed in less than nine days on average [95]. Their platform was named COPACOBANA, and consisted of an array of FPGAs. If a similar device were to be built today, the time and/or cost would be significantly lower because improved hardware is now available. Reproducing their results is easy: COPACOBANA is for sale, and renting computing time on the device is possible as well.

To overcome the shortcomings of DES, NIST launched a worldwide competi-

tion in search for a block cipher, resulting in the AES standard [47]. AES processes blocks of 128 bits, and supports key lengths of 128, 192 and 256 bits. In legacy applications where DES is difficult to replace, it is advisable to instead use 3DES [125].

Outline. Section 3.2 formally defines the concept of a block cipher. Our explanation is based on [113, Chap. 7]; we refer to this work for more details. We provide a classification of attacks, based on how the attacker can obtain plaintexts and corresponding ciphertexts. The concept of related-key attacks is defined as well, which will be relevant for Sect. 4.5 where we show how to prove security against related-key linear and differential attacks in an automated way.

Recently, *meet-in-the-middle* attacks became a very popular way to analyze block ciphers and hash functions. We constructed a novel variant of this technique, and applied it in an automated way to the block ciphers XTEA and GOST in Sect. 3.3. Our attacks require very few known plaintext-ciphertext pairs.

3.2 Definition

We use the symbols $\mathcal{P}$, $\mathcal{C}$ and $\mathcal{K}$ to denote the plaintext space, the ciphertext space and the key space respectively. Let $|\mathcal{P}| = |\mathcal{C}| = 2^N$ and $|\mathcal{K}| = 2^k$. We refer to $P \in \mathcal{P}$ as the plaintext, $K \in \mathcal{K}$ as the key and $C \in \mathcal{C}$ as the ciphertext of the block cipher.

A block cipher is a function $E : \mathcal{P} \times \mathcal{K} \to \mathcal{C}$, such that for every key $K \in \mathcal{K}$, $C = E(P, K)$ is an invertible mapping: the encryption function. The inverse mapping $P = D(C, K)$ is referred to as the decryption function. Both E and D should be efficiently computable.

There are $2^N!$ permutations from N bits to N bits. As the key consists of k bits, there are $(2^N!)^{2^k}$ possible block ciphers. An *ideal cipher* is a block cipher that is chosen uniformly at random from the space of all possible block ciphers. Therefore, an ideal cipher is the block cipher analogy of a random oracle for hash functions. In fact, Coron et al. proved that the random oracle model and the ideal cipher model are equivalent [41]. Holenstein et al. recently found an error in the proof of [41], but provided a new proof in [74].

A *pseudorandom permutation* is one $N \times N$-bit permutation chosen according to the uniform distribution out of a family of 2^k permutations (addressable by a key K), in such a way that it is computationally infeasible to distinguish it from an $N \times N$-bit permutation that is chosen uniformly at random [105]. The distinguisher can make close to 2^N queries, both encryption and decryption queries are allowed.[1]

In the pseudorandom permutation model, only one key K is used for all encryptions and decryptions. This is different from the ideal cipher model, which allows encryptions and decryptions under different keys. Both the ideal cipher model

[1] Luby and Rackoff [105] use the term *super pseudorandom* instead of *pseudorandom*, to indicate that decryptions are also allowed. For simplicity, we do not make this distinction.

and the pseudorandom permutation model are useful to analyze the complexity of generic block cipher attacks, as well as to prove the security of protocols and modes of operations based on block ciphers.

Depending on the setting, a block cipher is considered to be theoretically broken if it can be distinguished from either a pseudorandom permutation or from an ideal cipher. Whether a block cipher attack has practical consequences, depends on the power of the attacker in a particular setting. A large variety of settings exist. The next section provides a brief overview.

3.2.1 Attack Models

We now describe several attack models, sorted by increasing power to the adversary. Each of the attacks can be used to distinguish a block cipher from a pseudorandom permutation.

In a *ciphertext-only attack*, the attacker has access to the ciphertexts, and to some statistical information on the plaintexts. For example, the attacker may know that the plaintext is a text in English.

A *known plaintext attack* assumes that the attacker can obtain plaintexts and their corresponding ciphertexts, all encrypted under the same (secret) key. The attacker may, for example, know the header of a file that has been encrypted. A known plaintext attack is a passive attack, because the attacker cannot influence the content of the plaintexts.

In a *chosen plaintext attack*, the attacker can choose a set of plaintexts, and obtain the corresponding ciphertext values. This may seem unrealistic, but it is exactly what happens in the "Identify Friend or Foe" (IFF) protocol, used in many electronic car keys as part of the KeeLoq [79] access control system. The car key acts as an encryption device, and will output the encryption of any plaintext chosen by the attacker.

In an *adaptive chosen plaintext attack*, the attacker chooses plaintexts adaptively based on information learned from previously obtained ciphertexts.

A *chosen plaintext / chosen ciphertext attack* considers that the attacker can not only obtain encryptions for a chosen set of plaintexts, but can also choose ciphertexts and obtain the corresponding plaintexts. Similarly, an *adaptive chosen plaintext / chosen ciphertext attack* assumes that the attacker can choose both plaintexts and ciphertexts adaptively, based on the results of earlier queries. This may seem even more unrealistic, but there exists a practical attack for websites secured by SSL v3.0 under this model. In the attack by Bleichenbacher [29], the attacker recovers the key by choosing ciphertexts in an adaptive way and finding out if the corresponding plaintexts correspond to correctly padded messages.[2]

The number of queries that the attacker makes (encryptions or decryptions), is referred to as the data complexity of the attack. The maximum number of

[2]Bleichenbacher's attack does not involve a block cipher, but the public-key cipher RSA. However, the same attack model is applicable.

queries that can be made under a single key, may be limited by the protocol: the protocol may frequently renegotiate a new, uniformly random key. The block cipher attacks obtained in Sect. 3.3 are known plaintext attacks with a low data complexity. Therefore, our attacks are applicable even under very pessimistic assumptions for the attacker.

3.2.2 Related-Key Attacks

A single, fixed key K is chosen uniformly at random from the entire key space in the *standard (single-key) setting*. The adversary can then query the block cipher E for the encryption of different plaintexts P_i, all under the same key K.

In a *related-key setting*, the adversary can query the block cipher E for encryptions under different plaintexts P_i and different keys K_i. These keys satisfy the relationship $K_i = \Phi_i(K)$, where the attacker chooses the functions Φ_i. As noted in [13], restrictions are necessary on functions Φ_i, in order to ensure that not every block cipher E is vulnerable to a trivial related-key attack.

If related-key attacks exist, a block cipher is not secure under the ideal cipher model. This may result in attacks for some constructions, for example for hash functions based on block ciphers [137]. However, the same block cipher may still be secure in the pseudorandom permutation model. Using the technique introduced in Sect. 4.5 (see p. 229 and [124]), it becomes easy to prove the security of ciphers against linear and differential cryptanalysis in a related-key setting.

3.3 Meet-in-the-Middle Attacks

Meet-in-the-middle attacks are a relatively old idea in the field of symmetric-key cryptanalysis, dating back to the attack on the Double DES block cipher by Diffie and Hellman in 1977 [53]. For hash functions, the first meet-in-the-middle attack was given by Merkle [137] on Rabin's block cipher-based hash function [139]. The idea behind a meet-in-the-middle attack is to separate the mathematical equations that describe a symmetric-key cryptographic primitive into two or more groups. This is done in such a way, that some variables do not appear in at least one of the groups of equations. Ciphers typically consist of several rounds, and the separation of the cipher equations into groups often occurs near the middle rounds of the cipher, hence term "meet-in-the-middle" attacks.

In recent years, there has been a vast increase in the number of results using the meet-in-the-middle technique. They were used to attack a reduced-round version of DES [56]. The full version of several block ciphers were broken using meet-in-the-middle techniques: KeeLoq [79], KTANTAN [32, 180, 181], GOST [80] and even AES [30]. Meet-in-the middle attacks have been very successful in finding preimages for hash functions, see Table 3.1 for an overview of attacks on hash functions outside of the SHA-3 competition. Most of the underlying techniques

are not only applicable to hash functions, but have inspired the field of block cipher cryptanalysis as well.

Table 3.1 – Overview of meet-in-the-middle preimage attacks on hash functions

Hash function	Rounds Attacked	Reference
AES-MMO	7/10	[145]
HAS-160	68/80	[76, 147]
HAS-V	100/100	[150, 151]
3-pass HAVAL	96/96	[12, 148]
4-pass HAVAL	128/128	[148]
5-pass HAVAL	158/160	[142, 148]
Truncated 3-pass HAVAL	96/96	[144]
MD4	48/48	[8, 69]
MD5	64/64	[8, 12, 149]
PKC98-Hash	96/96	[150, 151]
RIPEMD-128	33/80	[131, 172]
RIPEMD-160	32/80	[131]
RIPEMD	47/48	[172]
SHA-0	52/80	[9, 52]
SHA-1	48/80	[9, 52]
SHA-256	43/64	[7, 69, 81]
SHA-512	46/80	[7, 69, 81]
Tiger	24/24	[69, 81, 171]
Whirlpool	5/10	[145]

In Sect. 3.3.1 and Sect. 3.3.2, we will present our own meet-in-the-middle attacks on XTEA [154] (see p. 147) and GOST [153] (see p. 169) respectively. All our attacks require very few known plaintexts and a negligible amount of memory.

3.3.1 XTEA

TEA (Tiny Encryption Algorithm) is a block cipher designed by Wheeler and Needham [182]. It has a key size of 128 bits, and uses 64 rounds to encrypt 64-bit blocks. Most notably, TEA uses only a few, simple operations and has an extremely small implementation in software. QQ Instant Messenger, a very popular instant messaging program with 712 million active users [162], uses a variant of TEA where the number of rounds is halved [166]. TEA is also used in Microsoft's Xbox gaming console [159].

After its publication, the cipher was attacked in several ways. Most notably, Kelsey et al. showed that there are equivalent keys for TEA [86], which reduces the

effective key size to 126 bits. In response to these attacks, XTEA was released [129] as a redesign of TEA. Like TEA, XTEA also has a block size of 64 bits and a key size of 128 bits. Both TEA and XTEA are implemented in the Linux kernel [68].

In [154] (see p. 147), we present meet-in-the-middle attacks on twelve variants of the XTEA block cipher. Each variant consists of 23 rounds. Two out of these attacks require only 18 known plaintexts and a computational effort equivalent to testing about 2^{117} keys. The success probability of our attacks is $1 - 2^{-1025}$. Currently, there are no published attacks on 23 or more rounds of XTEA, with a lower time and data complexity than our attacks.

An overview of attacks on XTEA can be found in Table 3.2. After our paper was published, further results on XTEA were obtained by Chen et al. [40], and by Bogdanov et al. [33]. Also, Sasaki recently announced that our meet-in-the-middle attacks could be improved [146]. He constructed meet-in-the-middle attacks on 29, 30 and 31 rounds of XTEA. We could not include his attacks in the overview table, because the details are not yet published.

3.3.2 GOST

The block cipher GOST (GOST 28147-89) is a Russian standard for encryption and message authentication [134]. For simplicity, we simply refer to this cipher as "GOST". The cipher is used in many applications, including OpenSSL 1.0.0 [132]. In 2010, GOST was submitted to ISO for standardization.

GOST has a block size of 64 bits, a key size of 256 bits and uses 32 rounds. Following the declassification of GOST in 1989, several cryptanalysis results were published on GOST. Table 3.3 provides an overview. Attacks on weak key classes, as well as related-key attacks are omitted. Results that were obtained after our observations are listed as well, including an attack by Isobe that breaks the full GOST [80].

In [153] (see p. 169), we present meet-in-the middle attacks on up to 22 rounds of GOST. Our 22-round attack has a computational complexity equivalent to testing about 2^{223} keys and a success probability of $1-2^{-65}$. The attack requires only 5 known plaintexts. At the time of writing, our attack is the best (going by the number of rounds) low data complexity key-recovery attack on GOST.

3.4 Conclusion

Block ciphers are an important primitive in cryptography. A block cipher uses a secret key to transform a plaintext into a ciphertext. Without knowledge of the secret key, it should be computationally infeasible to recover the original plaintext. Block ciphers are well studied and standardized, and can be used to build other cryptographic primitives, including hash functions, message authentication codes and stream ciphers.

Table 3.2 – Key recovery attacks on XTEA where the time complexities are averages, if explicitly stated in the original paper, average success probabilities are given as well (KP: known plaintext, CP: chosen plaintext, RK: in a related-key setting)

Attack	Ref.	# Rounds	Time	Data	Pr[Success]
• Attacks in the standard (single-key) setting					
Meet-in-the-middle	**p. 147**	**7**	$\mathbf{2^{95.00}}$	**2 KPs**	$\mathbf{1-2^{-33}}$
Impossible differential	[116]	14	2^{85}	$2^{62.5}$ CPs	Not given
Differential	[77]	15	2^{120}	2^{59} CPs	Not given
Meet-in-the-middle	**p. 147**	**15**	$\mathbf{2^{95.00}}$	**3 KPs**	$\mathbf{1-2^{-65}}$
Truncated differential	[77]	23	$2^{120.65}$	$2^{20.55}$ CPs	0.969
Meet-in-the-middle	**p. 147**	**23**	$\mathbf{2^{117.00}}$	**18 KPs**	$\mathbf{1-2^{-1025}}$
Impossible Differential	[40]	23	$2^{116.9}$	2^{62} CPs	Not given
Impossible Differential	[40]	23	$2^{105.6}$	2^{63} CPs	Not given
Zero-Correlation Linear Hull	[33]	25	$2^{124.53}$	$2^{62.62}$ KPs	0.846
Zero-Correlation Linear Hull	[33]	27	$2^{120.71}$	2^{64}	1
• Attacks in a related-key setting					
Related-key truncated differential	[91]	27	$2^{115.15}$	$2^{20.5}$ RK-CPs	0.969
Related-key rectangle (for $2^{108.21}$ weak keys)	[98]	34	$2^{31.94}$	2^{62} RK-CPs	Not given
Related-key rectangle	[104]	36	$2^{126.44}$	$2^{64.98}$ RK-CPs	0.63
Related-key rectangle (for $2^{110.67}$ weak keys)	[104]	36	$2^{104.33}$	$2^{63.83}$ RK-CPs	0.80
Related-key	[34]	37	2^{125}	2^{63} RK-CPs	Not given
Related-key (for $2^{107.5}$ weak keys)	[34]	51	2^{123}	2^{63} RK-CPs	Not given

We defined the concept of a block cipher in Sect. 3.2. An overview of attack models was given in Sect. 3.2.1. The protocol or application determines the power of the attacker, and therefore also the practical consequences of an attack on the block cipher. We looked into related-key attacks in Sect. 3.2.2. In a related-key attack, the attacker can request encryptions or decryptions under two or more distinct unknown keys, where the relation between the keys is under control of the attacker.

We described meet-in-the-middle attacks in Sect. 3.3. In Sect. 3.3.1, we pre-

Table 3.3 – Full-key recovery attacks on GOST; if explicitly stated in the original paper, success probabilities are given as well (KP: known plaintext, CP: chosen plaintext); attacks on weak key classes or using related keys are not included

Attack	Ref.	# Rounds	Time	Data	Pr[Success]
Meet-in-the-middle	**p. 169**	**8**	$\mathbf{2^{127.00}}$	**3 KPs**	$\mathbf{1-2^{-65}}$
Meet-in-the-middle	**p. 169**	**9, 10**	$\mathbf{2^{159.00}}$	**3 KPs**	$\mathbf{1-2^{-33}}$
Meet-in-the-middle	**p. 169**	**11, 12**	$\mathbf{2^{191.00}}$	**4 KPs**	$\mathbf{1-2^{-65}}$
Differential	[155]	13	Not given	2^{51} CPs	Not given
Meet-in-the-middle	**p. 169**	**13, 14**	$\mathbf{2^{223.00}}$	**4 KPs**	$\mathbf{1-2^{-33}}$
Meet-in-the-middle	**p. 169**	**16**	$\mathbf{2^{223.00}}$	**5 KPs**	$\mathbf{1-2^{-65}}$
Meet-in-the-middle	**p. 169**	**22**	$\mathbf{2^{223.00}}$	**5 KPs**	$\mathbf{1-2^{-65}}$
Slide	[23]	24	2^{64}	$\approx 2^{64}$ KPs	Not given
Slide	[23]	30	$2^{253.7}$	$\approx 2^{64}$ KPs	Not given
Reflection	[83]	30	2^{224}	2^{32} KPs	Not given
Reflection-meet-in-the-middle	[54, 80]	32	2^{225}	2^{32} KPs	Not given
Differential	[42]	32	$2^{226.3}$	2^{64}	0.5

sented several meet-in-the-middle attacks on up to 23 rounds of XTEA. Most notable about our attacks, is that they require no more than 18 known plaintexts. Our 23-round attacks have a time complexity of $2^{117.00}$ equivalent encryptions, and a success probability of $1-2^{-1025}$. At this time of writing, there are no published attacks with a lower time and data complexity than our attacks.

Again using meet-in-the-middle techniques, we attacked up to 22 rounds of GOST in Sect. 3.3.2. The 22-round attack requires a computational complexity of 2^{223} keys and has a success probability of $1-2^{-65}$. Only 5 known plaintexts are required. Going by the number of rounds, our attack is the best low data complexity key-recovery attack on GOST.

Chapter 4

Automated Techniques

4.1 Introduction

Besides a theoretical treatment of cryptographic primitives, the previous chapters focused mostly on the cryptanalysis of specific block ciphers and hash functions. In Chapter 2, we obtained results for nine SHA-3 candidates, as well as for the hash function HAS-V outside of the SHA-3 competition. We constructed low-data complexity meet-in-the-middle attacks on the block ciphers XTEA and GOST in Chapter 3.

In this chapter, our focus is not on a particular block cipher or hash function. Instead, we consider automated tools, with which many cryptographic primitives can be analyzed. So to clarify the title of this chapter: although "automated techniques" were also used in previous chapters, we now consider them not just as a means to an end, but worthy of a study on their own.

The Importance of Tools

For both the design and cryptanalysis of cryptographic primitives, computer programs are becoming increasingly important.

Unfortunately, many software tools are custom-made to analyze a particular cipher. The programmers also often lack the incentive to clean up their source code and make it publicly available. In fact, closely guarding such tools even provides a competitive advantage to obtain additional research results.

The lack of publicly available tools not only represents an entrance barrier for new researchers, who waste their time reinventing the wheel. If research results are based on a computer program that is not publicly available, how can we verify that this tool exists and works correctly?

We feel that the reproducibility, honesty and quality of research results are hurt, if the tools used to obtain them are not released. In this spirit, we launched the

"Tools for Cryptography" initiative[1] as part of the ECRYPT II project. A large number of tools are collected to assist cryptographers and cryptanalysts. Tools that were developed to obtain the results of this Ph.D. thesis can be found there as well.

Outline. We explain the concepts of differential and linear cryptanalysis in Sect. 4.2. These are two of the most powerful techniques in the cryptanalysis of symmetric-key primitives, and will be used in all results of this chapter.

In Sect. 4.3, the concept of S-functions is introduced. They can be used as a general framework to analyze ciphers based on components such as addition modulo 2^n, bitwise rotation, XOR and bitwise Boolean functions. In the context of differential cryptanalysis, we develop fast techniques to calculate the probability of differentials, to count the number of output differences and to obtain a list of output differences, sorted in descending order of probability.

We revisit the differential-algebraic attacks proposed by Albrecht and Cid [1,3, 4] in Sect. 4.4. In an differential-algebraic attack, differential cryptanalysis is used to generate equations, which are then solved using an automated technique, for example by computing a Gröbner basis or using a SAT solver. We point out mistakes in the results by Albrecht and Cid, and explain why their differential-algebraic attacks do not provide an advantage over differential cryptanalysis. Instead, we propose our own differential-algebraic attacks for reduced-round versions of the ISO lightweight encryption standard PRESENT [31].

Sect. 4.5 proposes a novel technique that uses Mixed-Integer Linear Programming (MILP) to easily prove the resistance of ciphers against linear and differential cryptanalysis. Our technique involves writing out simple linear inequality constraints, which can then be automatically solved using a software package for MILP programs.

4.2 Differential and Linear Cryptanalysis

We already considered meet-in-the middle attacks for block ciphers and hash functions in Sect. 3.3. In this section, we describe two other techniques that are essential for the design and cryptanalysis of symmetric-key cryptographic primitives: differential cryptanalysis and linear cryptanalysis. For a more detailed explanation of these techniques, we refer to [50].

Differential cryptanalysis was introduced by Biham and Shamir [24]. The technique turned out to be very successful, and resulted in the first attack on the full DES [25], as well as the first related-key attack on the full AES-192 and AES-256 [26, 27].

A differential attack can be used to distinguish a cipher from a random cipher, or even to recover the secret key. A differential distinguisher can be constructed as follows. Assume that we are given two plaintext-ciphertext pairs $C = E(P, K)$ and $C' = E(P', K)$, both encrypted under the same key K. The attack is a

[1] `http://www.ecrypt.eu.org/tools`

chosen plaintext attack (see Sect. 3.2.1), where the attacker chooses two plaintexts with a *difference* $\Delta P = P \bullet P'$. In this formula, $\bullet$ is a general operation to calculate a difference, common examples are XOR and subtraction modulo 2^n. The attacker does not know the key K, but may distinguish the cipher from a random permutation, if a particular ciphertext difference $\Delta C = C \bullet C'$ occurs with a significantly higher (or lower) probability than for a random cipher. To build a distinguisher with a high success probability, the attacker will have to encrypt a sufficiently large number of chosen plaintext pairs.

The probability that a plaintext difference ΔP leads to a particular ciphertext difference ΔC (averaged over all keys K and all plaintexts P), is referred to as the *differential probability* corresponding to the *differential* $\Delta P \to \Delta C$. Because this probability may be difficult to calculate, the attacker often makes assumptions on intermediate differences $\Delta X, \Delta Y, \ldots$, which appear after every round of the cipher. The sequence of differences $\Delta P, \Delta X, \Delta Y, \ldots, \Delta C$ is referred to as a *differential characteristic*. These concepts are illustrated in Fig. 4.1.

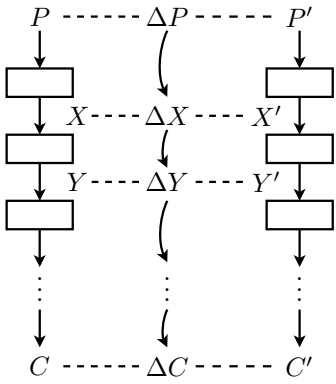


Figure 4.1 – A differential distinguisher

It is often possible to convert a differential distinguisher into a key recovery attack. How this works, is explained in detail in [173] (see p. 203).

Linear cryptanalysis, introduced by Matsui [109], studies an equation of the form

$$\Gamma_P \cdot P \oplus \Gamma_C \cdot C = \Gamma_K \cdot K \ , \tag{4.1}$$

which holds with a probability $p \neq 1/2$. The notation $x \cdot y$ is used to denote the parity of the bitwise product of x and y. Whereas differential cryptanalysis involves differences Δ, linear cryptanalysis involves *linear masks* Γ. Linear cryptanalysis is a known plaintext attack (see Sect. 3.2.1), and can either be used to build a distinguishing attack or a key recovery attack.

4.3 S-functions

4.3.1 Introduction

More and more, cryptographic primitives use components such as addition and subtraction modulo 2^n, as well as bitwise Boolean functions as a source of non-linearity in GF(2). This is the case for the most commonly used hash functions: MD5, SHA-1 and the SHA-2 family. Other examples include the block cipher XTEA [129] and the Salsa20 stream cipher family [17].

In the NIST SHA-3 competition [126], these basic building blocks are used in 6 out of 14 second-round candidates,[2] as well as in two out of the five SHA-3 finalists: BLAKE [10] and Skein [61].

There are many advantages to such constructions. They have a very fast performance on general-purpose CPUs, and can be described using only elementary mathematical operations. There is also no need to store look-up tables, which not only avoids cache timing attacks [92], but also results in a more compact implementation.

However, the security properties of these constructions are currently not well understood. In particular, none of these ciphers are currently proven to be secure against two of the most common attacks against symmetric-key primitives: differential and linear cryptanalysis (see Sect. 4.2). Ciphers built using S-boxes are typically resistant against such attacks: the block cipher AES, for example, is provably secure against both linear and differential cryptanalysis [47].

4.3.2 Background

S-boxes used in cryptographic primitives typically have a size of up to 8×8 bits. This leads to a relatively compact *difference distribution table*, which lists all possible input and output differences and their corresponding probability. The difference distribution table of an S-box of 8×8 bits contains $2^{16} = 65,536$ elements. It is therefore easy to calculate differential probabilities, to sort differentials by their probability and to count the number of possible output differences corresponding to a particular input difference.

For cryptosystems built using addition and subtraction modulo 2^n, we typically have $n = 32$ or $n = 64$. This means the difference distribution table would contain 2^{64} or 2^{128} elements. Because it is impractical to construct such a large table, we need to find other methods to calculate its properties.

Assume that we want to calculate the XOR differential probability of addition (xdp^+): that is, we consider the operation addition modulo 2^n, where input and output differences are expressed using XOR. More formally, let $\Delta x, \Delta y, \Delta z$ be fixed XOR differences such that

$$x_2 = x_1 \oplus \Delta x, \;\; y_2 = y_1 \oplus \Delta y, \;\; z_2 = z_1 \oplus \Delta z \;\; . \tag{4.2}$$

[2] In particular: the hash functions BLAKE [10], Blue Midnight Wish [66], CubeHash [18], Shabal [35], SIMD [99] and Skein [61].

Then, $\mathrm{xdp}^+(\Delta x, \Delta y \to \Delta z)$ is equal to the fraction of pairs (x_1, y_1) for which the following holds:

$$((x_1 \oplus \Delta x) + (y_1 \oplus \Delta y)) \oplus (x_1 + y_1) = \Delta z \ . \tag{4.3}$$

Lipmaa et al. showed in [103] that xdp^+ can be calculated by a multiplication of matrices. Consider the following example where $n = 6$:

$$\begin{aligned} &\mathrm{xdp}^+(\mathtt{11100}, \mathtt{00110} \to \mathtt{10110}) \\ &= LA_{101}A_{100}A_{111}A_{011}A_{000}C = \frac{1}{4} \ , \end{aligned} \tag{4.4}$$

where

$$A_{000} = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}, \ A_{001} = A_{010} = A_{100} = \frac{1}{2}\begin{bmatrix} 0 & 1 \\ 0 & 1 \end{bmatrix} \ , \tag{4.5}$$

$$A_{011} = A_{101} = A_{110} = \frac{1}{2}\begin{bmatrix} 1 & 0 \\ 1 & 0 \end{bmatrix}, \ A_{111} = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} \ , \tag{4.6}$$

$$L = [\ 1 \quad 1\ ], \ C = [\ 1 \quad 0\ ]^T \ . \tag{4.7}$$

For every bit position i, matrix $A_{w[i]}$ is selected, where $w[i] = x_[i] \parallel y[i] \parallel z[i]$. Lipmaa et al. [103] first present the matrices $A_{w[i]}$, and then prove mathematically that they are correct.

4.3.3 Our Results

Inspired by Lipmaa et al.'s work, we raise the following research questions:

- How can we obtain the same matrices $A_{w[i]}$ that were obtained by Lipmaa et al., in such a way that they can be mathematically proven to be correct by construction?
- Lipmaa et al. analyze the XOR differential probability of addition (xdp^+) [102], the Pseudo-Hadamard Transform [101] and the additive differential probability of XOR ($\mathrm{adp}^\oplus$) [103]. How can we analyze related constructions?
- How can we construct an efficient algorithm to find the output difference with the highest probability?
- In some cases, we are not interested in the differential probability, but in the number of possible output differences. Is there an efficient algorithm for this?

To answer all these questions, we introduced the framework of S-functions in [123] (see p. 119). An S-function computes the output at bit position i, based on the inputs at bit position i and a state $S[i]$. Only a finite number of states $S[i]$ are allowed. The paper is accompanied by a publicly available software toolkit.[3]

Subsequently, we also showed how to calculate the additive differential probability of an ARX (Addition-Rotation-XOR) operation [168], which is at the core of the SHA-3 finalists BLAKE and Skein. In [169], we introduced a new type of difference, referred to as UNAF, that can be used to calculate the probabilities of differential characteristics of ARX-based ciphers more accurately.

4.4 Differential-Algebraic Attacks

The idea behind algebraic cryptanalysis for block ciphers is to express the cipher as a multivariate polynomial system of equations. The solution to this system of equations is the secret encryption key K.

Several techniques exist to solve these systems of equations, including computing a Gröbner basis or using a SAT solver. These can be seen as automated "black box" techniques: they can be applied to any non-linear system of equations, but hopefully detect some inherent structure in the equations, so that a solution can be obtained faster than using exhaustive key search.

So far, algebraic techniques have had limited success at breaking real-world cryptographic ciphers. It seems that they are typically not faster than conventional techniques, such as linear and differential cryptanalysis (Sect. 4.2) or meet-in-the-middle attacks (Sect. 3.3).

4.4.1 Our Results

Albrecht and Cid [1, 3, 4] proposed to combine algebraic cryptanalysis and differential cryptanalysis, and claimed that the resulting differential-algebraic attack performs better than standard differential cryptanalysis. More specifically, they introduced three new attacks, which are referred to as Attack A, Attack B and Attack C, and used them to attack reduced rounds of the PRESENT block cipher [31].

The time complexity of Attack A is difficult to determine. For Attack B and Attack C, Albrecht and Cid construct an algebraic system of equations to filter out wrong pairs, and recover the key. In [173] (see p. 203), we show that Attack C cannot be used to filter out wrong pairs with the correct ciphertext difference, and therefore does not provide an advantage over differential cryptanalysis. We also explain why Attack B does not have an advantage over differential cryptanalysis for the block cipher PRESENT.

We have verified our results experimentally, using both PolyBoRi [36] and MiniSat [58]. Later, Albrecht also confirmed the validity of our results [2].

[3] `http://www.ecrypt.eu.org/tools/s-function-toolkit`

Based on our observations, we also presented two new differential-algebraic attacks on reduced-round PRESENT [173] (see p. 203). They have a higher time complexity than the corresponding differential attacks, but their data complexity is lower. Whether these attacks perform better than meet-in-the-middle attacks, is an interesting topic to investigate.

4.5 Mixed Integer-Linear Programming

In Sect. 4.4, we were pessimistic about algebraic cryptanalysis. We showed how the differential-algebraic attacks proposed by Albrecht and Cid do not provide an advantage over differential cryptanalysis. Currently, it seems that no efficient symmetric-key cipher can be broken faster using an algebraic attack than using conventional techniques.

We now take a different approach: we argue that algebraic techniques can be very useful to construct cryptographic attacks, and to prove the security of ciphers against these attacks.

Assume that a cryptographer wants to use an automated technique to prove the security of a cipher against linear and differential cryptanalysis. We then make the following observations:

- This program will only be executed once to prove the security bounds, whereas the cipher will be executed many times to perform encryptions and decryptions.
- The programmer will typically spend much more time on programming, debugging, optimizing, writing a parallel implementation,... than the time it takes to execute the resulting program.
- It is very difficult to verify the correctness of a complicated computer program. For this reason, a simple program is preferred to an efficient one.
- A programmer's time is much more expensive than CPU time.

Therefore, we argue that when we write a program to prove the security of a cipher, the execution time of this program is not so important. Much more important is to have a solution that is easy to program, easy to verify and easy to to parallelize.

4.5.1 Our Results

Linear Programming (LP) studies the optimization (minimization or maximization) problem of a linear objective function $f(x_1, x_2, \ldots, x_n)$, subject to linear constraints involving decision variables x_i, $1 \leq i \leq n$. In an MILP problem, certain decision variables x_i are restricted to integer values.

In [124] (see p. 229), we showed how to prove the security of many ciphers against linear and differential cryptanalysis using MILP. Our analysis applies to both single-key and related-key attacks. The only requirement is that the cipher is composed of a combination of S-box operations, linear permutation layers and/or XOR operations. Our technique is based on counting the minimum number of active S-boxes. The objective function of our MILP program is the number of active S-boxes, which are subject to simple linear constraints that can easily be generated using the description of the cipher. The MILP program can then be solved using an off-the-shelf optimization package, for example IBM's CPLEX.

We considered both the stream cipher Enocoro-128v2 [178, 179] and the block cipher AES [46, 47]. For Enocoro-128v2, the execution time to prove the security against differential cryptanalysis is less than one minute on a 24-core Intel Xeon X5670 processor, and less than four minutes to prove security against linear cryptanalysis. For AES, we calculated the minimum number of active S-boxes for up to 14 rounds. None of the corresponding MILP programs took longer than 0.4 seconds to execute. To the best of our knowledge, we are the first to solve this problem by generating only one MILP program.

Wang and Bogdanov showed in a follow-up paper how MILP can be used to demonstrate the tightness of bounds for Feistel ciphers that use the Diffusion Switching Mechanism (DSM) [174].

4.6 Conclusion

To analyze cryptographic ciphers against cryptanalytic attacks, automated tools are becoming increasingly important. In this chapter, we focused both on evaluating the performance of existing tools, as well as on the development of new tools.

We first explained linear and differential cryptanalysis in Sect. 4.2. These two powerful techniques to analyze symmetric-key ciphers are essential to understand the cryptanalysis tools presented in this chapter.

Increasingly, cryptographic ciphers use operations such as XOR, addition and subtraction modulo 2^n, bitwise Boolean functions, bit shifts and bit rotations. To analyze such constructions in a general way, we introduced the concept of S-functions in Sect. 4.3. The S-functions framework allows us to calculate the probability of differentials, to count the number of output differences, as well as to efficiently find the output difference or differences with the highest probability.

Algebraic cryptanalysis involves writing out a system of multivariate polynomial equations, and solving them using an automated "black box" techniques, for example by using a SAT solver or computing a Gröbner basis. If the equations have a special structure, they may be solved faster than a generic system of non-linear equations.

Our research investigated the interesting combination of algebraic cryptanalysis and differential cryptanalysis, as proposed by Albrecht and Cid at FSE 2009.

Contrary to to their original claims, our theoretic observations and extensive computer experiments showed that their differential-algebraic attacks do not outperform standard differential cryptanalysis. After the publication of our findings, the validity of our results were confirmed by Albrecht.

Linear programming (LP) involves optimizing a linear objective function, subject to linear inequality constraints. In Mixed-Integer Linear Programming (MILP), some of the variables in the optimization program are restricted to integer values. We introduced a novel technique to prove the security of ciphers against both linear and differential cryptanalysis. Our method involves only writing out simple linear inequality constraints, corresponding to the operations of the cipher. The resulting MILP program can be used to prove lower bounds against the number of active S-boxes for both linear and differential cryptanalysis. For the ciphers that we analyzed (AES and Enocoro-182v2), each of the resulting MILP programs took less than four minutes to solve.

Chapter 5

Conclusion

Two cryptographic components that are essential to build secure applications are block ciphers and hash functions. Traditionally, block ciphers provide confidentiality by encrypting information using a secret key, whereas hash functions ensure the integrity of information. However, nowadays block ciphers and hash functions have a myriad of uses, far beyond their original intentions.

Already in 1996, Dobbertin found a collision for the compression function of MD5 [55]. More recently, Wang et al. discovered collision attacks for the commonly used hash functions MD5 and SHA-1. For MD5, it is possible to find collisions for the hash function in just a few milliseconds on a standard PC. Finding collisions for SHA-1 is also feasible, however a lot more computing power is required. Although the SHA-2 hash functions are currently unbroken, they are based on similar design principles. For these reasons, NIST launched the SHA-3 competition in search for a new hash functions standard.

This Ph.D. thesis closely follows the SHA-3 competition. Research results were obtained for the SHA-3 candidates ARIRANG, BLAKE, ESSENCE, Hamsi, Khichidi-1, LUX, Sarmal, Skein and TIB3. As a result of our analysis, the hash function Khichidi-1 was withdrawn, and ESSENCE did not advance to the second round of the competition. We also analyzed the hash function HAS-V, and made some observations on the concept of a regular hash function.

Our research results do not only involve hash functions, we obtained new attacks for block ciphers as well. For the block ciphers XTEA and GOST, we found a novel application of the meet-in-the-middle attack. An advantage of our attacks is that they require only very few known plaintext-ciphertext pairs.

Whereas the attacks of Wang et al. are based on tedious hand calculations, we strongly favor the use of simple automated techniques. In this context, we have constructed a general framework to analyze ciphers based on components such as addition modulo 2^n, XOR, bit rotation and bitwise Boolean functions. Our general S-functions framework allows us to easily calculate the probability of differentials, find the output difference(s) with the highest probability, as well as count the number of output differences. The corresponding S-functions toolkit is publicly

available online.

We also evaluated the differential-algebraic attacks proposed by Albrecht and Cid, which employ automated tools such as MiniSat and PolyBoRi to solve multivariate systems of non-linear equations. Contrary to the claims by Albrecht and Cid, we find that their differential-algebraic attacks do not perform better than standard differential cryptanalysis. Our results are backed up by a large number of computer experiments, and were later confirmed by Albrecht.

Mixed-Integer Linear Programming (MILP) is a technique used in business and economics to solve optimization problems. We show how to construct a single MILP program to calculate the number of active S-boxes for a given cipher. This allows us to prove the security of a cipher against both linear and differential cryptanalysis.

5.1 Directions for Future Research

Symmetric-key cryptography is a relatively new field, given that two of the most powerful techniques (linear and differential cryptanalysis) have only been publicly known for the past twenty years. Although the study of block ciphers and hash functions has matured significantly in recent years, many open problems still remain. We now provide an overview of interesting topics for future research.

- Conditions in a differential characteristic that involve two pairs of bits are referred to as "double conditions" in [120] (see p. 69). As double conditions involve four bits, there are 2^{16} possible double conditions. For the cryptanalysis of MD4 [175], MD5 [177], RIPEMD [175] and SHA-1 [176], a subset of these double conditions was already used: conditions where a bit of one pair is equal to a bit of another pair, and conditions where they are opposite. These conditions also appeared in [110], where they were obtained by automated techniques. It seems interesting to explore the application of all 2^{16} possible double conditions to these hash functions.

- The S-functions toolkit provides an algorithm to find the output difference with the highest probability, using a variant of the A* search algorithm. The algorithm is described in [169]. Further investigation of the A* heuristic is required, in order to get a better understanding of the time and memory complexity of our algorithm.

- After the introduction of our S-function framework [123] (see p. 119) to analyze differential probabilities, we analyzed the differential probability of ARX [168] and reduced-round versions of the stream cipher Salsa20 in [169]. We often found that due to a clustering effect of differential characteristics, the actual differential probability can be much higher than estimated using only one characteristic. Although we focused on developing advanced techniques to obtain more accurate estimates, this effect is still not well understood.

- Alternatively, the actual differential probability of a characteristic may also be much lower than estimated (or even zero). We revisited the SHA-1 results of De Cannière and Rechberger [51]. By searching all messages that satisfy their 64-step characteristic, we found that the characteristic actually has a probability of zero: no colliding message pair exists. Careful inspection shows that the 64-step collision provided in [51] follows the 64-step characteristic only for the first 50 steps. An important open problem is therefore to efficiently determine whether a characteristic has a non-zero probability.

- A powerful method for the differential cryptanalysis of ARX-based algorithms, is to replace every addition modulo 2^n by an XOR operation. The resulting cipher then becomes linear in GF(2). A low-weight codeword for this linear cipher may correspond to a useful characteristic for the original cipher [135]. However, a problem occurs for the first-round SHA-3 candidate EDON-$\mathcal{R}$ [67]. Consider the following equations:

$$\begin{array}{lcl} T_0 & = & (Y_1 + Y_7 + \ldots) \ggg 0 \ , \\ T_1 & = & (Y_1 + Y_4 + \ldots) \ggg 5 \ , \\ T_3 & = & (Y_4 + Y_7 + \ldots) \ggg 11 \ . \end{array}$$

  We found a differential characteristic where the i-th bit positions of Y_1, Y_4 and Y_7 contain an XOR difference of 1 (and i is not the most significant bit). In the characteristic, all differences should cancel out, so that the T-words contain no difference. However, this cancellation will never happen: every XOR difference of 1 corresponds either to the bit pair $(0, 1)$ or $(1, 0)$, and no choice of bit pairs for $(Y_1[i], Y_1'[i])$, $(Y_4[i], Y_4'[i])$ and $(Y_7[i], Y_7'[i])$ will lead to a zero difference in the T-words. An open problem is how to automatically and efficiently search for EDON-$\mathcal{R}$ characteristics that do not have this problem.

- In [173] (see p. 203), two new differential-algebraic attacks were proposed. Although the time complexity of these attacks is much higher than for a differential attack on the same cipher, the data complexity is lower. As a large number of bits are fixed in our new differential-algebraic attacks, it is interesting to investigate whether meet-in-the-middle attacks could outperform our differential-algebraic attacks.

- Our attacks on reduced-round XTEA [154] (see p. 147) and GOST [153] (see p. 169) are based a simple application of the meet-in-the-middle technique. It is interesting to investigate the application of more advanced techniques, for example the biclique cryptanalysis used to attack the SHA-3 candidate Skein-512 [89], the SHA-2 [89] family and the full AES [30].

- We proposed a technique to prove the security of ciphers against linear and differential cryptanalysis using mixed-integer linear programming [124] (see p. 229). A direction for future work is to apply our technique to other attacks, for example to impossible differential cryptanalysis.

Bibliography

[1] M. Albrecht. *Algorithmic Algebraic Techniques and their Application to Block Cipher Cryptanalysis*. PhD thesis, Royal Holloway, University of London, 2010.

[2] M. Albrecht. Personal Communication, May 2011.

[3] M. Albrecht and C. Cid. Algebraic Techniques in Differential Cryptanalysis. In O. Dunkelman, editor, *FSE*, volume 5665 of *Lecture Notes in Computer Science*, pages 193–208. Springer, 2009.

[4] M. Albrecht, C. Cid, T. Dullien, J.-C. Faugère, and L. Perret. Algebraic Precomputations in Differential and Integral Cryptanalysis. In X. Lai, M. Yung, and D. Lin, editors, *Inscrypt*, volume 6584 of *Lecture Notes in Computer Science*, pages 387–403. Springer, 2010.

[5] Alibo. Is This MITM Attack to Gmail's SSL?, August 2011. `https://www.google.com/support/forum/p/gmail/thread?tid=2da6158b094b225a`.

[6] E. Andreeva and M. Stam. The Symbiosis between Collision and Preimage Resistance. In L. Chen, editor, *IMA Int. Conf.*, volume 7089 of *Lecture Notes in Computer Science*, pages 152–171. Springer, 2011.

[7] K. Aoki, J. Guo, K. Matusiewicz, Y. Sasaki, and L. Wang. Preimages for Step-Reduced SHA-2. In M. Matsui, editor, *ASIACRYPT*, volume 5912 of *Lecture Notes in Computer Science*, pages 578–597. Springer, 2009.

[8] K. Aoki and Y. Sasaki. Preimage Attacks on One-Block MD4, 63-Step MD5 and More. In R. M. Avanzi, L. Keliher, and F. Sica, editors, *Selected Areas in Cryptography*, volume 5381 of *Lecture Notes in Computer Science*, pages 103–119. Springer, 2008.

[9] K. Aoki and Y. Sasaki. Meet-in-the-Middle Preimage Attacks Against Reduced SHA-0 and SHA-1. In S. Halevi, editor, *CRYPTO*, volume 5677 of *Lecture Notes in Computer Science*, pages 70–89. Springer, 2009.

[10] J.-P. Aumasson, L. Henzen, W. Meier, and R. C.-W. Phan. SHA-3 proposal BLAKE. Submission to the NIST SHA-3 Competition (Round 3), 2010. http://131002.net/blake/blake.pdf.

[11] J.-P. Aumasson, G. Leurent, W. Meier, F. Mendel, N. Mouha, R. C.-W. Phan, Y. Sasaki, and P. Susil. Tuple cryptanalysis of ARX with application to BLAKE and Skein. ECRYPT II Hash Workshop, 2011. http://www.ecrypt.eu.org/hash2011/.

[12] J.-P. Aumasson, W. Meier, and F. Mendel. Preimage Attacks on 3-Pass HAVAL and Step-Reduced MD5. In R. M. Avanzi, L. Keliher, and F. Sica, editors, *Selected Areas in Cryptography*, volume 5381 of *Lecture Notes in Computer Science*, pages 120–135. Springer, 2008.

[13] M. Bellare and T. Kohno. A Theoretical Treatment of Related-Key Attacks: RKA-PRPs, RKA-PRFs, and Applications. In E. Biham, editor, *EUROCRYPT*, volume 2656 of *Lecture Notes in Computer Science*, pages 491–506. Springer, 2003.

[14] M. Bellare and T. Kohno. Hash Function Balance and Its Impact on Birthday Attacks, 2004. http://cseweb.ucsd.edu/~mihir/papers/balance.html.

[15] M. Bellare and P. Rogaway. Random Oracles are Practical: A Paradigm for Designing Efficient Protocols. In *ACM Conference on Computer and Communications Security*, pages 62–73, 1993.

[16] D. J. Bernstein. Cache-timing attacks on AES, 2005. http://cr.yp.to/antiforgery/cachetiming-20050414.pdf.

[17] D. J. Bernstein. The Salsa20 Family of Stream Ciphers. In M. J. B. Robshaw and O. Billet, editors, *The eSTREAM Finalists*, volume 4986 of *Lecture Notes in Computer Science*, pages 84–97. Springer, 2008.

[18] D. J. Bernstein. CubeHash specification (2.B.1). Submission to the NIST SHA-3 Competition (Round 2), 2009. http://cubehash.cr.yp.to/submission2/spec.pdf.

[19] G. Bertoni, J. Daemen, M. Peeters, and G. V. Assche. RadioGatún, a belt-and-mill hash function. *IACR Cryptology ePrint Archive*, 2006:369, 2006.

[20] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche. Sponge functions. ECRYPT II Hash Workshop, 2007. http://sponge.noekeon.org/.

[21] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche. The Keccak SHA-3 submission. Submission to the NIST SHA-3 Competition (Round 3), 2011. http://keccak.noekeon.org/Keccak-submission-3.pdf.

[22] E. Biham and O. Dunkelman. A Framework for Iterative Hash Functions - HAIFA. Cryptology ePrint Archive, Report 2007/278, 2007. `http://eprint.iacr.org/`.

[23] E. Biham, O. Dunkelman, and N. Keller. Improved Slide Attacks. In A. Biryukov, editor, *FSE*, volume 4593 of *Lecture Notes in Computer Science*, pages 153–166. Springer, 2007.

[24] E. Biham and A. Shamir. Differential Cryptanalysis of DES-like Cryptosystems. *J. Cryptology*, 4(1):3–72, 1991.

[25] E. Biham and A. Shamir. Differential Cryptanalysis of the Full 16-Round DES. In E. F. Brickell, editor, *CRYPTO*, volume 740 of *Lecture Notes in Computer Science*, pages 487–496. Springer, 1992.

[26] A. Biryukov and D. Khovratovich. Related-Key Cryptanalysis of the Full AES-192 and AES-256. In M. Matsui, editor, *ASIACRYPT*, volume 5912 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2009.

[27] A. Biryukov, D. Khovratovich, and I. Nikolic. Distinguisher and Related-Key Attack on the Full AES-256. In S. Halevi, editor, *CRYPTO*, volume 5677 of *Lecture Notes in Computer Science*, pages 231–249. Springer, 2009.

[28] A. Biryukov and A. Shamir. Structural Cryptanalysis of SASAS. *J. Cryptology*, 23(4):505–518, 2010.

[29] D. Bleichenbacher. Chosen Ciphertext Attacks Against Protocols Based on the RSA Encryption Standard PKCS #1. In H. Krawczyk, editor, *CRYPTO*, volume 1462 of *Lecture Notes in Computer Science*, pages 1–12. Springer, 1998.

[30] A. Bogdanov, D. Khovratovich, and C. Rechberger. Biclique Cryptanalysis of the Full AES. In D. H. Lee and X. Wang, editors, *ASIACRYPT*, volume 7073 of *Lecture Notes in Computer Science*, pages 344–371. Springer, 2011.

[31] A. Bogdanov, L. R. Knudsen, G. Leander, C. Paar, A. Poschmann, M. J. B. Robshaw, Y. Seurin, and C. Vikkelsoe. PRESENT: An Ultra-Lightweight Block Cipher. In P. Paillier and I. Verbauwhede, editors, *CHES*, volume 4727 of *Lecture Notes in Computer Science*, pages 450–466. Springer, 2007.

[32] A. Bogdanov and C. Rechberger. A 3-Subset Meet-in-the-Middle Attack: Cryptanalysis of the Lightweight Block Cipher KTANTAN. In A. Biryukov, G. Gong, and D. R. Stinson, editors, *Selected Areas in Cryptography*, volume 6544 of *Lecture Notes in Computer Science*, pages 229–240. Springer, 2010.

[33] A. Bogdanov and M. Wang. Zero Correlation Linear Cryptanalysis with Reduced Data Complexity. In A. Canteaut, editor, *FSE*, Lecture Notes in Computer Science. Springer, 2012.

[34] C. Bouillaguet, O. Dunkelman, G. Leurent, and P.-A. Fouque. Another Look at Complementation Properties. In S. Hong and T. Iwata, editors, *FSE*, volume 6147 of *Lecture Notes in Computer Science*, pages 347–364. Springer, 2010.

[35] E. Bresson, A. Canteaut, B. Chevallier-Mames, C. Clavier, T. Fuhr, A. Gouget, T. Icart, J.-F. Misarsky, M. Naya-Plasencia, P. Paillier, T. Pornin, J.-R. Reinhard, C. Thuillet, and M. Videau. Shabal, a Submission to NIST's Cryptographic Hash Algorithm Competition. Submission to the NIST SHA-3 Competition (Round 2), 2008. `http://ehash.iaik.tugraz.at/uploads/6/6c/Shabal.pdf`.

[36] M. Brickenstein and A. Dreyer. PolyBoRi: A framework for Gröbner-basis computations with Boolean polynomials. *J. Symb. Comput.*, 44(9):1326–1345, 2009.

[37] C. Cachin and J. Camenisch, editors. *Advances in Cryptology - EUROCRYPT 2004, International Conference on the Theory and Applications of Cryptographic Techniques, Interlaken, Switzerland, May 2-6, 2004, Proceedings*, volume 3027 of *Lecture Notes in Computer Science*. Springer, 2004.

[38] R. Canetti, O. Goldreich, and S. Halevi. The random oracle methodology, revisited. *J. ACM*, 51(4):557–594, 2004.

[39] D. Chang, S. Hong, C. Kang, J. Kang, J. Kim, C. Lee, J. Lee, J. Lee, S. Lee, Y. Lee, J. Lim, and J. Sung. ARIRANG. Submission to the NIST SHA-3 Competition (Round 1), 2008. `http://ehash.iaik.tugraz.at/uploads/2/2c/Arirang.pdf`.

[40] J. Chen, M. Wang, and B. Preneel. Impossible Differential Cryptanalysis of the Lightweight Block Ciphers TEA, XTEA and HIGHT. Cryptology ePrint Archive, Report 2011/616, 2011. `http://eprint.iacr.org/`.

[41] J.-S. Coron, J. Patarin, and Y. Seurin. The Random Oracle Model and the Ideal Cipher Model Are Equivalent. In D. Wagner, editor, *CRYPTO*, volume 5157 of *Lecture Notes in Computer Science*, pages 1–20. Springer, 2008.

[42] N. Courtois and M. Misztal. First Differential Attack on Full 32-Round GOST. In S. Qing, W. Susilo, G. Wang, and D. Liu, editors, *ICICS*, volume 7043 of *Lecture Notes in Computer Science*, pages 216–227. Springer, 2011.

[43] A. Coviello. Open Letter to RSA SecurID Customers, 2011. `http://www.rsa.com/node.aspx?id=3891`.

[44] J. Daemen. Personal communication, January 2010.

[45] J. Daemen, L. R. Knudsen, and V. Rijmen. The Block Cipher Square. In E. Biham, editor, *FSE*, volume 1267 of *Lecture Notes in Computer Science*, pages 149–165. Springer, 1997.

[46] J. Daemen and V. Rijmen. The Wide Trail Design Strategy. In B. Honary, editor, *IMA Int. Conf.*, volume 2260 of *Lecture Notes in Computer Science*, pages 222–238. Springer, 2001.

[47] J. Daemen and V. Rijmen. *The Design of Rijndael: AES - The Advanced Encryption Standard*. Springer, 2002.

[48] W. Dai. OFFICIAL COMMENT: LUX. NIST SHA-3 mailing list, 2009. `http://ehash.iaik.tugraz.at/uploads/e/ec/Lux_dai.txt`.

[49] I. Damgård. A Design Principle for Hash Functions. In G. Brassard, editor, *CRYPTO*, volume 435 of *Lecture Notes in Computer Science*, pages 416–427. Springer, 1989.

[50] C. De Cannière, A. Biryukov, and B. Preneel. An introduction to block cipher cryptanalysis. *Proceedings of the IEEE*, 94(2):346–356, February 2006.

[51] C. De Cannière and C. Rechberger. Finding SHA-1 Characteristics: General Results and Applications. In X. Lai and K. Chen, editors, *ASIACRYPT*, volume 4284 of *Lecture Notes in Computer Science*, pages 1–20. Springer, 2006.

[52] C. De Cannière and C. Rechberger. Preimages for Reduced SHA-0 and SHA-1. In D. Wagner, editor, *CRYPTO*, volume 5157 of *Lecture Notes in Computer Science*, pages 179–202. Springer, 2008.

[53] W. Diffie and M. E. Hellman. Exhaustive Cryptanalysis of the NBS Data Encryption Standard. *Computer*, 10(6):74–84, 1977.

[54] I. Dinur, O. Dunkelman, and A. Shamir. Improved Attacks on Full GOST. Cryptology ePrint Archive, Report 2011/558, 2011. `http://eprint.iacr.org/`.

[55] H. Dobbertin. Cryptanalysis of MD5. Presented at a rump session of EuroCrypt '96, 1996.

[56] O. Dunkelman, G. Sekar, and B. Preneel. Improved Meet-in-the-Middle Attacks on Reduced-Round DES. In K. Srinathan, C. P. Rangan, and M. Yung, editors, *INDOCRYPT*, volume 4859 of *Lecture Notes in Computer Science*, pages 86–100. Springer, 2007.

[57] T. Duong and J. Rizzo. Flickr's API Signature Forgery Vulnerability. Netifera Research, September 2009. `Flickr'sAPISignatureForgeryVulnerability`.

[58] N. Eén and N. Sörensson. An Extensible SAT-solver. In E. Giunchiglia and A. Tacchella, editors, *SAT*, volume 2919 of *Lecture Notes in Computer Science*, pages 502–518. Springer, 2003.

[59] M. Feldhofer and C. Rechberger. A Case Against Currently Used Hash Functions in RFID Protocols. In R. Meersman, Z. Tari, and P. Herrero, editors, *OTM Workshops (1)*, volume 4277 of *Lecture Notes in Computer Science*, pages 372–381. Springer, 2006.

[60] N. Ferguson. Re: Official comment: Lux. NIST SHA-3 mailing list, 2009. `http://ehash.iaik.tugraz.at/uploads/2/21/Lux_niels.txt`.

[61] N. Ferguson, S. Lucks, B. Schneier, D. Whiting, M. Bellare, T. Kohno, J. Callas, and J. Walker. The Skein Hash Function Family. Submission to the NIST SHA-3 Competition (Round 3), 2010. `http://www.skein-hash.info/sites/default/files/skein1.3.pdf`.

[62] K. Gandolfi, C. Mourtel, and F. Olivier. Electromagnetic Analysis: Concrete Results. In Çetin Kaya Koç, D. Naccache, and C. Paar, editors, *CHES*, volume 2162 of *Lecture Notes in Computer Science*, pages 251–261. Springer, 2001.

[63] F. D. Garcia, G. de Koning Gans, R. Muijrers, P. van Rossum, R. Verdult, R. W. Schreur, and B. Jacobs. Dismantling MIFARE Classic. In S. Jajodia and J. López, editors, *ESORICS*, volume 5283 of *Lecture Notes in Computer Science*, pages 97–114. Springer, 2008.

[64] P. Gauravaram, L. R. Knudsen, K. Matusiewicz, F. Mendel, C. Rechberger, M. Schläffer, and S. S. Thomsen. Grøstl – a SHA-3 candidate. Submission to the NIST SHA-3 Competition (Round 3), 2011. `http://www.groestl.info/Groestl.pdf`.

[65] H. Gilbert and M. Minier. A Collision Attack on 7 Rounds of Rijndael. In *AES Candidate Conference*, pages 230–241, 2000.

[66] D. Gligoroski, V. Klima, S. J. Knapskog, M. El-Hadedy, J. Amundsen, and S. F. Mjølsnes. Cryptographic Hash Function BLUE MIDNIGHT WISH. Submission to the NIST SHA-3 Competition (Round 2), 2009. `http://people.item.ntnu.no/~danilog/Hash/BMW-SecondRound/Supporting_Documentation/BlueMidnightWishDocumentation.pdf`.

[67] D. Gligoroski, R. S. Ødegård, M. Mihova, S. J. Knapskog, L. Kocarev, A. Drápal, and V. Klima. Cryptographic Hash Function EDON-$\mathcal{R}$. Submission to the NIST SHA-3 Competition (Round 1), 2008. `http://people.item.ntnu.no/~danilog/Hash/Edon-R/Supporting_Documentation/EdonRDocumentation.pdf`.

[68] A. Grothe. Kernel v2.6.14 tea.c. Linux Headquarters, 2004. `http://www.linuxhq.com/kernel/v2.6/14/crypto/tea.c`.

[69] J. Guo, S. Ling, C. Rechberger, and H. Wang. Advanced Meet-in-the-Middle Preimage Attacks: First Results on Full Tiger, and Improved Results on MD4 and SHA-2. In M. Abe, editor, *ASIACRYPT*, volume 6477 of *Lecture Notes in Computer Science*, pages 56–75. Springer, 2010.

[70] J. Guo, K. Matusiewicz, L. R. Knudsen, S. Ling, and H. Wang. Practical Pseudo-collisions for Hash Functions ARIRANG-224/384. In M. J. Jacobson Jr., V. Rijmen, and R. Safavi-Naini, editors, *Selected Areas in Cryptography*, volume 5867 of *Lecture Notes in Computer Science*, pages 141–156. Springer, 2009.

[71] J. Hermans, A. Pashalidis, F. Vercauteren, and B. Preneel. A New RFID Privacy Model. In V. Atluri and C. Díaz, editors, *ESORICS*, volume 6879 of *Lecture Notes in Computer Science*, pages 568–587. Springer, 2011.

[72] K. Hirai. Letter to Honorable Mary Bono Mack and Honorable G. K. Butterfield. PlayStation.Blog, 2011. `http://blog.us.playstation.com/2011/05/04/sonys-response-to-the-u-s-house-of-representatives/`.

[73] S. Hirose, K. Ideguchi, H. Kuwakado, T. Owada, B. Preneel, and H. Yoshida. A Lightweight 256-Bit Hash Function for Hardware and Low-End Devices: Lesamnta-LW. In K. H. Rhee and D. Nyang, editors, *ICISC*, volume 6829 of *Lecture Notes in Computer Science*, pages 151–168. Springer, 2010.

[74] T. Holenstein, R. Künzler, and S. Tessaro. The equivalence of the random oracle model and the ideal cipher model, revisited. In L. Fortnow and S. P. Vadhan, editors, *STOC*, pages 89–98. ACM, 2011.

[75] D. Hong, D. Chang, J. Sung, S. Lee, S. Hong, J. Lee, D. Moon, and S. Chee. A New Dedicated 256-Bit Hash Function: FORK-256. In M. J. B. Robshaw, editor, *FSE*, volume 4047 of *Lecture Notes in Computer Science*, pages 195–209. Springer, 2006.

[76] D. Hong, B. Koo, and Y. Sasaki. Improved Preimage Attack for 68-Step HAS-160. In D. Lee and S. Hong, editors, *ICISC*, volume 5984 of *Lecture Notes in Computer Science*, pages 332–348. Springer, 2009.

[77] S. Hong, D. Hong, Y. Ko, D. Chang, W. Lee, and S. Lee. Differential Cryptanalysis of TEA and XTEA. In J. I. Lim and D. H. Lee, editors, *ICISC*, volume 2971 of *Lecture Notes in Computer Science*, pages 402–417. Springer, 2003.

[78] S. Indesteege. The LANE hash function. Submission to the NIST SHA-3 Competition (Round 1), 2008. `http://www.cosic.esat.kuleuven.be/publications/article-1181.pdf`.

[79] S. Indesteege, N. Keller, O. Dunkelman, E. Biham, and B. Preneel. A Practical Attack on KeeLoq. In N. P. Smart, editor, *EUROCRYPT*, volume 4965 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2008.

[80] T. Isobe. A Single-Key Attack on the Full GOST Block Cipher. In A. Joux, editor, *FSE*, volume 6733 of *Lecture Notes in Computer Science*, pages 290–305. Springer, 2011.

[81] T. Isobe and K. Shibutani. Preimage Attacks on Reduced Tiger and SHA-2. In O. Dunkelman, editor, *FSE*, volume 5665 of *Lecture Notes in Computer Science*, pages 139–155. Springer, 2009.

[82] A. Joux. Multicollisions in Iterated Hash Functions. Application to Cascaded Constructions. In M. K. Franklin, editor, *CRYPTO*, volume 3152 of *Lecture Notes in Computer Science*, pages 306–316. Springer, 2004.

[83] O. Kara. Reflection Cryptanalysis of Some Ciphers. In D. R. Chowdhury, V. Rijmen, and A. Das, editors, *INDOCRYPT*, volume 5365 of *Lecture Notes in Computer Science*, pages 294–307. Springer, 2008.

[84] J. Kelsey. Personal communication, May 2011.

[85] J. Kelsey and B. Schneier. Second Preimages on n-Bit Hash Functions for Much Less than 2^n Work. In R. Cramer, editor, *EUROCRYPT*, volume 3494 of *Lecture Notes in Computer Science*, pages 474–490. Springer, 2005.

[86] J. Kelsey, B. Schneier, and D. Wagner. Key-Schedule Cryptoanalysis of IDEA, G-DES, GOST, SAFER, and Triple-DES. In N. Koblitz, editor, *CRYPTO*, volume 1109 of *Lecture Notes in Computer Science*, pages 237–251. Springer, 1996.

[87] D. Khovratovich and I. Nikolic. Rotational Cryptanalysis of ARX. In S. Hong and T. Iwata, editors, *FSE*, volume 6147 of *Lecture Notes in Computer Science*, pages 333–346. Springer, 2010.

[88] D. Khovratovich, I. Nikolic, and C. Rechberger. Rotational Rebound Attacks on Reduced Skein. In M. Abe, editor, *ASIACRYPT*, volume 6477 of *Lecture Notes in Computer Science*, pages 1–19. Springer, 2010.

[89] D. Khovratovich, C. Rechberger, and A. Savelieva. Bicliques for Preimages: Attacks on Skein-512 and the SHA-2 family. *IACR Cryptology ePrint Archive*, 2011:286, 2011.

[90] D. E. Knuth. *Sorting and Searching*, volume 3 of *The Art of Computer Programming*. Addison-Wesley Professional, second edition, May 1998.

[91] Y. Ko, S. Hong, W. Lee, S. Lee, and J.-S. Kang. Related Key Differential Attacks on 27 Rounds of XTEA and Full-Round GOST. In B. K. Roy and W. Meier, editors, *FSE*, volume 3017 of *Lecture Notes in Computer Science*, pages 299–316. Springer, 2004.

[92] P. C. Kocher. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In N. Koblitz, editor, *CRYPTO*, volume 1109 of *Lecture Notes in Computer Science*, pages 104–113. Springer, 1996.

[93] P. C. Kocher, J. Jaffe, and B. Jun. Differential Power Analysis. In M. J. Wiener, editor, *CRYPTO*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397. Springer, 1999.

[94] Ö. Küçük. The Hash Function Hamsi. Submission to the NIST SHA-3 Competition (Round 2), 2009. `http://www.cosic.esat.kuleuven.be/publications/article-1203.pdf`.

[95] S. S. Kumar, C. Paar, J. Pelzl, G. Pfeiffer, and M. Schimmler. Breaking Ciphers with COPACOBANA - A Cost-Optimized Parallel Code Breaker. In L. Goubin and M. Matsui, editors, *CHES*, volume 4249 of *Lecture Notes in Computer Science*, pages 101–118. Springer, 2006.

[96] X. Lai and J. L. Massey. Hash Function Based on Block Ciphers. In R. A. Rueppel, editor, *EUROCRYPT*, volume 658 of *Lecture Notes in Computer Science*, pages 55–70. Springer, 1992.

[97] A. Langley. Public key pinning, 2011. `http://www.imperialviolet.org/2011/05/04/pinning.html`.

[98] E. Lee, D. Hong, D. Chang, S. Hong, and J. Lim. A Weak Key Class of XTEA for a Related-Key Rectangle Attack. In P. Q. Nguyen, editor, *VIETCRYPT*, volume 4341 of *Lecture Notes in Computer Science*, pages 286–297. Springer, 2006.

[99] G. Leurent, C. Bouillaguet, and P.-A. Fouque. SIMD Is a Message Digest. Submission to the NIST SHA-3 Competition (Round 2), 2009. `http://www.di.ens.fr/~leurent/files/SIMD.pdf`.

[100] C. H. Lim and P. J. Lee. A Study on the Proposed Korean Digital Signature Algorithm. In K. Ohta and D. Pei, editors, *ASIACRYPT*, volume 1514 of *Lecture Notes in Computer Science*, pages 175–186. Springer, 1998.

[101] H. Lipmaa. On Differential Properties of Pseudo-Hadamard Transform and Related Mappings. In A. Menezes and P. Sarkar, editors, *INDOCRYPT*, volume 2551 of *Lecture Notes in Computer Science*, pages 48–61. Springer, 2002.

[102] H. Lipmaa and S. Moriai. Efficient Algorithms for Computing Differential Properties of Addition. In M. Matsui, editor, *FSE*, volume 2355 of *Lecture Notes in Computer Science*, pages 336–350. Springer, 2001.

[103] H. Lipmaa, J. Wallén, and P. Dumas. On the Additive Differential Probability of Exclusive-Or. In B. K. Roy and W. Meier, editors, *FSE*, volume 3017 of *Lecture Notes in Computer Science*, pages 317–331. Springer, 2004.

[104] J. Lu. Related-key rectangle attack on 36 rounds of the XTEA block cipher. *Int. J. Inf. Sec.*, 8(1):1–11, 2009.

[105] M. Luby and C. Rackoff. How to Construct Pseudorandom Permutations from Pseudorandom Functions. *SIAM J. Comput.*, 17(2):373–386, 1988.

[106] S. Lucks. The Saturation Attack - A Bait for Twofish. In M. Matsui, editor, *FSE*, volume 2355 of *Lecture Notes in Computer Science*, pages 1–15. Springer, 2001.

[107] J. W. Martin. ESSENCE: A Family of Cryptographic Hashing Algorithms. `http://www.math.jmu.edu/~martin/essence/Supporting_Documentation/essence_compression.pdf`, 2008.

[108] J. W. Martin. Personal communication, July 2009.

[109] M. Matsui. Linear Cryptoanalysis Method for DES Cipher. In *EUROCRYPT*, pages 386–397, 1993.

[110] F. Mendel, T. Nad, and M. Schläffer. Finding SHA-2 Characteristics: Searching through a Minefield of Contradictions. In D. H. Lee and X. Wang, editors, *ASIACRYPT*, volume 7073 of *Lecture Notes in Computer Science*, pages 288–307. Springer, 2011.

[111] F. Mendel and V. Rijmen. Weaknesses in the HAS-V Compression Function. In K.-H. Nam and G. Rhee, editors, *ICISC*, volume 4817 of *Lecture Notes in Computer Science*, pages 335–345. Springer, 2007.

[112] F. Mendel and M. Schläffer. On Free-Start Collisions and Collisions for TIB3. In P. Samarati, M. Yung, F. Martinelli, and C. A. Ardagna, editors, *ISC*, volume 5735 of *Lecture Notes in Computer Science*, pages 95–106. Springer, 2009.

[113] A. Menezes, P. C. van Oorschot, and S. A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1997.

[114] R. C. Merkle. One Way Hash Functions and DES. In G. Brassard, editor, *CRYPTO*, volume 435 of *Lecture Notes in Computer Science*, pages 428–446. Springer, 1989.

[115] M. Montes and D. Penazzi. The TIB3 Hash. Submission to the NIST SHA-3 Competition (Round 1), 2008. http://www.famaf.unc.edu.ar/~penazzi/tib3/submitted/Supporting_Documentation/TIB3_Algorithm_Specification.pdf.

[116] D. Moon, K. Hwang, W. Lee, S. Lee, and J. Lim. Impossible Differential Cryptanalysis of Reduced Round XTEA and TEA. In J. Daemen and V. Rijmen, editors, *FSE*, volume 2365 of *Lecture Notes in Computer Science*, pages 49–60. Springer, 2002.

[117] N. Mouha. Collision for Khichidi-1. NIST SHA-3 mailing list, 2008. http://ehash.iaik.tugraz.at/uploads/8/89/Khichidi-1.txt.

[118] N. Mouha. RE: OFFICIAL COMMENT: LUX. NIST SHA-3 mailing list, 2009. http://ehash.iaik.tugraz.at/uploads/7/78/Lux_nicky.txt.

[119] N. Mouha, T. E. Bjørstad, and B. Preneel. Non-randomness in the Sarmal compression function, 2009. http://ehash.iaik.tugraz.at/uploads/8/8c/Sarmal.new.pdf.

[120] N. Mouha, C. De Cannière, S. Indesteege, and B. Preneel. Finding Collisions for a 45-Step Simplified HAS-V. In H. Y. Youm and M. Yung, editors, *WISA*, volume 5932 of *Lecture Notes in Computer Science*, pages 206–225. Springer, 2009.

[121] N. Mouha, G. Sekar, J.-P. Aumasson, T. Peyrin, S. S. Thomsen, M. S. Turan, and B. Preneel. Cryptanalysis of the ESSENCE Family of Hash Functions. In F. Bao, M. Yung, D. Lin, and J. Jing, editors, *Inscrypt*, volume 6151 of *Lecture Notes in Computer Science*, pages 15–34. Springer, 2009.

[122] N. Mouha, G. Sekar, and B. Preneel. Challenging the Increased Resistance of Regular Hash Functions Against Birthday Attacks. ECRYPT II Hash Workshop, May 2011. http://www.ecrypt.eu.org/hash2011/.

[123] N. Mouha, V. Velichkov, C. De Cannière, and B. Preneel. The Differential Analysis of S-Functions. In A. Biryukov, G. Gong, and D. R. Stinson, editors, *Selected Areas in Cryptography*, volume 6544 of *Lecture Notes in Computer Science*, pages 36–56. Springer, 2010.

[124] N. Mouha, Q. Wang, D. Gu, and B. Preneel. Differential and Linear Cryptanalysis using Mixed-Integer Linear Programming. In M. Yung and C.-K. Wu, editors, *Inscrypt*, Lecture Notes in Computer Science. Springer, 2011.

[125] National Institute of Standards and Technology. FIPS PUB 46-3: Data Encryption Standard (DES), October 1999. http://www.itl.nist.gov/fipspubs/fip186-2.pdf.

[126] National Institute of Standards and Technology. Announcing Request for Candidate Algorithm Nominations for a New Cryptographic Hash Algorithm (SHA-3) Family. *Federal Register*, 27(212):62212–62220, November 2007. `http://csrc.nist.gov/groups/ST/hash/documents/FR_Notice_Nov07.pdf`.

[127] National Institute of Standards and Technology. Block Cipher Modes – Current Modes, November 2011. `http://csrc.nist.gov/groups/ST/toolkit/BCM/current_modes.html`.

[128] M. Naya-Plasencia, A. Röck, J.-P. Aumasson, Y. Laigle-Chapuy, G. Leurent, W. Meier, and T. Peyrin. Cryptanalysis of ESSENCE. In S. Hong and T. Iwata, editors, *FSE*, volume 6147 of *Lecture Notes in Computer Science*, pages 134–152. Springer, 2010.

[129] R. M. Needham and D. J. Wheeler. TEA extensions. Computer Laboratory, Cambridge University, England, 1997. `http://www.movable-type.co.uk/scripts/xtea.pdf`.

[130] I. Nikolić, A. Biryukov, and D. Khovratovich. Hash family LUX - Algorithm Specifications and Supporting Documentation. Submission to the NIST SHA-3 Competition (Round 1), 2008. `http://ehash.iaik.tugraz.at/uploads/f/f3/LUX.pdf`.

[131] C. Ohtahara, Y. Sasaki, and T. Shimoyama. Preimage Attacks on Step-Reduced RIPEMD-128 and RIPEMD-160. In X. Lai, M. Yung, and D. Lin, editors, *Inscrypt*, volume 6584 of *Lecture Notes in Computer Science*, pages 169–186. Springer, 2010.

[132] OpenSSL version 1.0.0, March 2010. `http://www.openssl.org/`.

[133] N. K. Park, J. H. Hwang, and P. J. Lee. HAS-V: A New Hash Function with Variable Output Length. In D. R. Stinson and S. E. Tavares, editors, *Selected Areas in Cryptography*, volume 2012 of *Lecture Notes in Computer Science*, pages 202–216. Springer, 2000.

[134] J. Pieprzyk and L. Tombak. Soviet Encryption Algorithm, June 1994. `http://freeworld.thc.org/root/phun/stego-challenge/gost-spec.pdf`.

[135] N. Pramstaller, C. Rechberger, and V. Rijmen. Exploiting Coding Theory for Collision Attacks on SHA-1. In N. P. Smart, editor, *IMA Int. Conf.*, volume 3796 of *Lecture Notes in Computer Science*, pages 78–95. Springer, 2005.

[136] B. Preneel. *Analysis and design of cryptographic hash functions*. PhD thesis, Katholieke Universiteit Leuven, 1993.

[137] B. Preneel, R. Govaerts, and J. Vandewalle. Hash Functions Based on Block Ciphers: A Synthetic Approach. In D. R. Stinson, editor, *CRYPTO*, volume 773 of *Lecture Notes in Computer Science*, pages 368–378. Springer, 1993.

[138] J.-J. Quisquater and J.-P. Delescaille. How Easy is Collision Search? Application to DES (Extended Summary). In *EUROCRYPT*, pages 429–434, 1989.

[139] M. Rabin. Digitalized Signatures. In R. A. DeMillo, D. P. Dobkin, A. K. Jones, and R. J. Lipton, editors, *Foundations of Secure Computation*, pages 155–168. New York Academic Press, 1978.

[140] P. Rogaway. Formalizing Human Ignorance. In P. Q. Nguyen, editor, *VIETCRYPT*, volume 4341 of *Lecture Notes in Computer Science*, pages 211–228. Springer, 2006.

[141] P. Rogaway and T. Shrimpton. Cryptographic Hash-Function Basics: Definitions, Implications, and Separations for Preimage Resistance, Second-Preimage Resistance, and Collision Resistance. In B. K. Roy and W. Meier, editors, *FSE*, volume 3017 of *Lecture Notes in Computer Science*, pages 371–388. Springer, 2004.

[142] Y. Sakai, Y. Sasaki, L. Wang, K. Ota, and K. Sakiyama. Preimage Attacks on 5-Pass HAVAL Reduced to 158-Steps and One-Block 3-Pass HAVAL. 9th International Conference on Applied Cryptography and Network Security (ACNS), 2011.

[143] SANS Institute. The Top Cyber Security Risks, 2009. `http://www.sans.org/top-cyber-security-risks/`.

[144] Y. Sasaki. Meet-in-the-Middle Attacks Using Output Truncation in 3-Pass HAVAL. In P. Samarati, M. Yung, F. Martinelli, and C. A. Ardagna, editors, *ISC*, volume 5735 of *Lecture Notes in Computer Science*, pages 79–94. Springer, 2009.

[145] Y. Sasaki. Meet-in-the-Middle Preimage Attacks on AES Hashing Modes and an Application to Whirlpool. In A. Joux, editor, *FSE*, volume 6733 of *Lecture Notes in Computer Science*, pages 378–396. Springer, 2011.

[146] Y. Sasaki. Recent Advances in MITM Preimage Attacks. ECRYPT II Hash Workshop, 2011. `http://www.ecrypt.eu.org/hash2011/`.

[147] Y. Sasaki and K. Aoki. A Preimage Attack for 52-Step HAS-160. In P. J. Lee and J. H. Cheon, editors, *ICISC*, volume 5461 of *Lecture Notes in Computer Science*, pages 302–317. Springer, 2008.

[148] Y. Sasaki and K. Aoki. Preimage Attacks on 3, 4, and 5-Pass HAVAL. In J. Pieprzyk, editor, *ASIACRYPT*, volume 5350 of *Lecture Notes in Computer Science*, pages 253–271. Springer, 2008.

[149] Y. Sasaki and K. Aoki. Finding Preimages in Full MD5 Faster Than Exhaustive Search. In A. Joux, editor, *EUROCRYPT*, volume 5479 of *Lecture Notes in Computer Science*, pages 134–152. Springer, 2009.

[150] Y. Sasaki, F. Mendel, and K. Aoki. Preimage Attacks against PKC98-Hash and HAS-V. In K. H. Rhee and D. Nyang, editors, *ICISC*, volume 6829 of *Lecture Notes in Computer Science*, pages 68–91. Springer, 2010.

[151] Y. Sasaki, F. Mendel, and K. Aoki. Preimage Attacks against PKC98-Hash and HAS-V. *IEICE Transactions*, 95-A(1):111–124, 2012.

[152] K. Scarfone and M. Souppaya. Guide to Enterprise Password Management. NIST special publication 800-118 (draft), National Institute of Standards and Technology (NIST), April 2009. `http://csrc.nist.gov/publications/drafts/800-118/draft-sp800-118.pdf`.

[153] G. Sekar, N. Mouha, and B. Preneel. Meet-in-the-Middle Attacks on Reduced-Round GOST. ISO/IEC JTC1/SC27 N8875, April 2010.

[154] G. Sekar, N. Mouha, V. Velichkov, and B. Preneel. Meet-in-the-Middle Attacks on Reduced-Round XTEA. In A. Kiayias, editor, *CT-RSA*, volume 6558 of *Lecture Notes in Computer Science*, pages 250–267. Springer, 2011.

[155] H. Seki and T. Kaneko. Differential Cryptanalysis of Reduced Rounds of GOST. In D. R. Stinson and S. E. Tavares, editors, *Selected Areas in Cryptography*, volume 2012 of *Lecture Notes in Computer Science*, pages 315–323. Springer, 2000.

[156] Sony Corporation. FY2010 Consolidated Results Forecast Revision, 2011. `http://www.sony.net/SonyInfo/IR/financial/fr/viewer/10revision/slide/03_slide.html`.

[157] Sony Corporation. Restoration Of PlayStation® Network And Qriocity Services Begins, May 2011. `http://www.sony.net/SonyInfo/News/Press/201105/11-0515E/index.html`.

[158] A. Sotirov, M. Stevens, J. Appelbaum, A. Lenstra, D. A. Molnar, D. A. Osvik, and B. de Weger. MD5 considered harmful today: Creating a rogue CA certificate, December 2008. 25th Chaos Communications Congress, Berlin, Germany.

[159] M. Steil. 17 Mistakes Microsoft Made in the Xbox Security System. 22nd Chaos Communication Congress, December 2005. `http://events.ccc.de/congress/2005/fahrplan/events/559.en.html`.

[160] M. Stevens. On Collisions for MD5. Master's thesis, Eindhoven University of Technology, 2007.

[161] Tata Consultancy Services. IT Services, Business Solutions, Outsourcing, 2011. http://www.tcs.com/.

[162] Tencent. About Tencent. http://www.tencent.com/en-us/at/abouttencent.shtml, 2012.

[163] H. Tsukayama. Cyber attack on RSA cost EMC $66 million, 2011. http://www.washingtonpost.com/blogs/post-tech/post/cyber-attack-on-rsa-cost-emc-66-million/2011/07/26/gIQA1ceKbI_blog.html.

[164] P. C. van Oorschot and M. J. Wiener. Parallel Collision Search with Cryptanalytic Applications. *J. Cryptology*, 12(1):1–28, 1999.

[165] K. Varıcı, O. Özen, and Ç. Kocair. Sarmal: SHA-3 Proposal. Submission to the NIST SHA-3 Competition (Round 1), 2008. http://homes.esat.kuleuven.be/~kvarici/Supporting_Documentation/SarmaL.pdf.

[166] Various Authors. libqq-pidgin. http://code.google.com/p/libqq-pidgin/, 2012.

[167] VASCO. VASCO reports range of estimated losses related to DigiNotar bankruptcy and related events, 2011. bit.ly/qwANHz.

[168] V. Velichkov, N. Mouha, C. De Cannière, and B. Preneel. The Additive Differential Probability of ARX. In A. Joux, editor, *FSE*, volume 6733 of *Lecture Notes in Computer Science*, pages 342–358. Springer, 2011.

[169] V. Velichkov, N. Mouha, C. De Cannière, and B. Preneel. UNAF: A Special Set of Additive Differences with Application to the Differential Analysis of ARX. In A. Canteaut, editor, *FSE*, Lecture Notes in Computer Science. Springer, 2012.

[170] N. Vijayarangan. A NEW HASH ALGORITHM: Khichidi-1. Submission to the NIST SHA-3 Competition (Round 1), 2008. http://ehash.iaik.tugraz.at/uploads/d/d4/Khichidi-1.pdf.

[171] L. Wang and Y. Sasaki. Finding Preimages of Tiger Up to 23 Steps. In S. Hong and T. Iwata, editors, *FSE*, volume 6147 of *Lecture Notes in Computer Science*, pages 116–133. Springer, 2010.

[172] L. Wang, Y. Sasaki, W. Komatsubara, K. Ohta, and K. Sakiyama. (Second) Preimage Attacks on Step-Reduced RIPEMD/RIPEMD-128 with a New Local-Collision Approach. In A. Kiayias, editor, *CT-RSA*, volume 6558 of *Lecture Notes in Computer Science*, pages 197–212. Springer, 2011.

[173] M. Wang, Y. Sun, N. Mouha, and B. Preneel. Algebraic Techniques in Differential Cryptanalysis Revisited. In U. Parampalli and P. Hawkes, editors, *ACISP*, volume 6812 of *Lecture Notes in Computer Science*, pages 120–141. Springer, 2011.

[174] Q. Wang and A. Bogdanov. The Provable Constructive Effect of the Diffusion Switching Mechanism for CLEFIA-type Block Ciphers. *Information Processing Letters (To Appear)*, 2012.

[175] X. Wang, X. Lai, D. Feng, H. Chen, and X. Yu. Cryptanalysis of the Hash Functions MD4 and RIPEMD. In R. Cramer, editor, *EUROCRYPT*, volume 3494 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2005.

[176] X. Wang, Y. L. Yin, and H. Yu. Finding Collisions in the Full SHA-1. In V. Shoup, editor, *CRYPTO*, volume 3621 of *Lecture Notes in Computer Science*, pages 17–36. Springer, 2005.

[177] X. Wang and H. Yu. How to Break MD5 and Other Hash Functions. In R. Cramer, editor, *EUROCRYPT*, volume 3494 of *Lecture Notes in Computer Science*, pages 19–35. Springer, 2005.

[178] D. Watanabe, K. Okamoto, and T. Kaneko. A Hardware-Oriented Light Weight Pseudo-Random Number Generator Enocoro-128v2. In *The Symposium on Cryptography and Information Security*, pages 3D1–3, 2010. (in Japanese).

[179] D. Watanabe, T. Owada, K. Okamoto, Y. Igarashi, and T. Kaneko. Update on Enocoro Stream Cipher. In *ISITA*, pages 778–783. IEEE, 2010.

[180] L. Wei, C. Rechberger, J. Guo, H. Wu, H. Wang, and S. Ling. Improved Meet-in-the-Middle Cryptanalysis of KTANTAN. *IACR Cryptology ePrint Archive*, 2011:201, 2011.

[181] L. Wei, C. Rechberger, J. Guo, H. Wu, H. Wang, and S. Ling. Improved Meet-in-the-Middle Cryptanalysis of KTANTAN (Poster). In U. Parampalli and P. Hawkes, editors, *ACISP*, volume 6812 of *Lecture Notes in Computer Science*, pages 433–438. Springer, 2011.

[182] D. J. Wheeler and R. M. Needham. TEA, a Tiny Encryption Algorithm. In B. Preneel, editor, *FSE*, volume 1008 of *Lecture Notes in Computer Science*, pages 363–366. Springer, 1994.

[183] A. Whitten and J. D. Tygar. Why Johnny Can't Encrypt. In *In Proceedings of the 8th USENIX Security Symposium*, 1999.

[184] H. Wu. The Hash Function JH. Submission to the NIST SHA-3 Competition (Round 3), 2011. `http://www3.ntu.edu.sg/home/wuhj/research/jh/jh_round3.pdf`.

[185] S. Wu, D. Feng, and W. Wu. Cryptanalysis of the Hash Function LUX-256. http://ehash.iaik.tugraz.at/uploads/3/36/Analysis_LUX_1.pdf, 2008.

[186] D. Zanetti, P. Sachs, and S. Capkun. On the Practicality of UHF RFID Fingerprinting: How Real is the RFID Tracking Problem? In S. Fischer-Hübner and N. Hopper, editors, *PETS*, volume 6794 of *Lecture Notes in Computer Science*, pages 97–116. Springer, 2011.

Part II

Publications

List of Publications

Lecture Notes in Computer Science

1. Nicky Mouha, Christophe De Cannière, Sebastiaan Indesteege, and Bart Preneel. Finding Collisions for a 45-Step Simplified HAS-V. In Heung Youl Youm and Moti Yung, editors, *WISA*, volume 5932 of *Lecture Notes in Computer Science*, pages 206–225. Springer, 2009.

   – See p. 69.

2. Nicky Mouha, Gautham Sekar, Jean-Philippe Aumasson, Thomas Peyrin, Søren S. Thomsen, Meltem Sönmez Turan, and Bart Preneel. Cryptanalysis of the ESSENCE Family of Hash Functions. In Feng Bao, Moti Yung, Dongdai Lin, and Jiwu Jing, editors, *Inscrypt*, volume 6151 of *Lecture Notes in Computer Science*, pages 15–34. Springer, 2009.

   – See p. 95.

3. Nicky Mouha, Vesselin Velichkov, Christophe De Cannière, and Bart Preneel. The Differential Analysis of S-Functions. In Alex Biryukov, Guang Gong, and Douglas R. Stinson, editors, *Selected Areas in Cryptography*, volume 6544 of *Lecture Notes in Computer Science*, pages 36–56. Springer, 2010.

   – See p. 119.

4. Vesselin Velichkov, Nicky Mouha, Christophe De Cannière, and Bart Preneel. The Additive Differential Probability of ARX. In Antoine Joux, editor, *FSE*, volume 6733 of *Lecture Notes in Computer Science*, pages 342–358. Springer, 2011.

5. Gautham Sekar, Nicky Mouha, Vesselin Velichkov, and Bart Preneel. Meet-in-the-Middle Attacks on Reduced-Round XTEA. In Aggelos Kiayias, editor, *CT-RSA*, volume 6558 of *Lecture Notes in Computer Science*, pages 250–267. Springer, 2011.

   – See p. 147.

6. Meiqin Wang, Yue Sun, Nicky Mouha, and Bart Preneel. Algebraic Techniques in Differential Cryptanalysis Revisited. In Udaya Parampalli and Philip Hawkes, editors, *ACISP*, volume 6812 of *Lecture Notes in Computer Science*, pages 120–141. Springer, 2011.

   – See p. 203.

7. Nicky Mouha, Qingju Wang, Dawu Gu, and Bart Preneel. Differential and Linear Cryptanalysis using Mixed-Integer Linear Programming. In Moti Yung and Chuan-Kun Wu, editors, *Inscrypt*, Lecture Notes in Computer Science. Springer, 2011.

   – See p. 229.

8. Vesselin Velichkov, Nicky Mouha, Christophe De Cannière, and Bart Preneel. UNAF: A Special Set of Additive Differences with Application to the Differential Analysis of ARX. In Anne Canteaut, editor, *FSE*, Lecture Notes in Computer Science. Springer, 2012.

Under Submission to International Journals

1. Gautham Sekar, Nicky Mouha, and Bart Preneel. Meet-in-the-Middle Attacks on Reduced-Round GOST. ISO/IEC JTC1/SC27 N8875, April 2010.

   – Under review at an international journal. See p. 169.

2. Nicky Mouha, Gautham Sekar, and Bart Preneel. Challenging the Increased Resistance of Regular Hash Functions Against Birthday Attacks. ECRYPT II Hash Workshop, May 2011. `http://www.ecrypt.eu.org/hash2011/`.

   – Under review at an international journal. See p. 181.

Invited Lectures

1. Nicky Mouha. ARX-based Cryptography. ECRYPT II Design and Security of Cryptographic Algorithms and Devices, Invited Lecture, June 2011.

2. Nicky Mouha. Cryptanalysis of Symmetric-Key Primitives: Automated Techniques, Invited Lecture. ECRYPT II Summer School on Tools, May 2012.

Book Chapters

1. Nicky Mouha. MD4-MD5. In Henk C. A. van Tilborg and Sushil Jajodia, editors, *Encyclopedia of Cryptography and Security (2nd Ed.)*, pages 768–771. Springer, 2011.

Workshops

1. Nicky Mouha, Christophe De Cannière, Sebastiaan Indesteege, and Bart Preneel. Finding Collisions for a 45-Step Simplified HAS-V. 3rd Benelux Workshop on Information and System Security (WISSec), November 2008.

   – See p. 69.

2. Nicky Mouha, Søren S. Thomsen, Meltem Sönmez Turan, and Bart Preneel. Observations of non-randomness in the ESSENCE compression function. The First SHA-3 Candidate Conference, Rump Session Talk, February 2009.

3. Nicky Mouha, Vesselin Velichkov, Christophe De Cannière, and Bart Preneel. Toolkit for the Differential Cryptanalysis of ARX-based Cryptographic Constructions. ECRYPT II Workshop on Tools for Cryptanalysis, June 2010.

4. Nicky Mouha, Vesselin Velichkov, Christophe De Cannière, and Bart Preneel. The Differential Analysis of S-functions. BCRYPT PhD day, May 2010.

   – See p. 119.

5. Markus Ullrich, Christophe De Cannière, Sebastiaan Indesteege, Özgül Küçük, Nicky Mouha, and Bart Preneel. Finding Optimal Bitsliced Implementations of 4 x 4-bit S-boxes. Symmetric Key Encryption Workshop (SKEW), February 2011.

6. Nicky Mouha, Gautham Sekar, and Bart Preneel. Challenging the Increased Resistance of Regular Hash Functions Against Birthday Attacks. ECRYPT II Hash Workshop, May 2011. `http://www.ecrypt.eu.org/hash2011/`.

   – See p. 181.

7. Jean-Philippe Aumasson, Gaëtan Leurent, Willi Meier, Florian Mendel, Nicky Mouha, Raphael C.-W. Phan, Yu Sasaki, and Petr Susil. Tuple cryptanalysis of ARX with application to BLAKE and Skein. ECRYPT II Hash Workshop, 2011. `http://www.ecrypt.eu.org/hash2011/`.

8. Meiqin Wang, Yue Sun, Nicky Mouha, and Bart Preneel. Algebraic Techniques in Differential Cryptanalysis Revisited. BCRYPT PhD day, May 2011.

   – See p. 203.

Other Publications

1. Nicky Mouha, Tor E. Bjørstad, and Bart Preneel. Non-randomness in the Sarmal compression function, 2009. `http://ehash.iaik.tugraz.at/uploads/8/8c/Sarmal.new.pdf`.

2. Nicky Mouha, Gautham Sekar, Jean-Philippe Aumasson, Thomas Peyrin, Søren S. Thomsen, Meltem Sönmez Turan, and Bart Preneel. Cryptanalysis of the ESSENCE Family of Hash Functions. COSIC seminar, December 2009.

   – See p. 95.

3. Gautham Sekar, Nicky Mouha, and Bart Preneel. Meet-in-the-Middle Attacks on Reduced-Round GOST. ISO/IEC JTC1/SC27 N8875, April 2010.

   – See p. 169.

4. Jonathan Etrog, Dmitry Khovratovich, Willi Meier, Nicky Mouha, Jorge Nakahara Jr., Andrea Röck, Vincent Rijmen Christian Rechberger Martin Schläffer, and Vesselin Velichkov. D.SYM.6: New developments in symmetric key cryptanalysis. ECRYPT II Symlab Report, June 2010.

5. Tor E. Bjørstad, Andrey Bogdanov, Henri Gilbert, Kota Ideguchi, Sebastiaan Indesteege, Özgül Küçük, Gregor Leander, Nicky Mouha, Jorge Nakahara Jr., Axel Poschmann, Christian Rechberger, Vincent Rijmen, Gautham Sekar, Kyoji Shibutani, Martin Schläffer, François-Xavier Standaert, Elmar Tischhauser, Vesselin Velichkov, and Ivan Visconti. D.SYM.5: WG2 Lightweight Cryptographic Algorithms. ECRYPT II Symlab Report, July 2010.

6. Andrey Bodganov, Nicky Mouha, Gautham Sekar, Elmar Tischhauser, Deniz Toz, Kerem Varıcı, Vesselin Velichkov, and Meiqin Wang. Security Evaluation of the K2 Stream Cipher. CRYPTREC Technical Report, March 2010.

7. Nicky Mouha. ARX-based Cryptography. Lecture at Tsinghua University, June 2011.

8. Meiqin Wang, Yue Sun, Nicky Mouha, and Bart Preneel. Algebraic Techniques in Differential Cryptanalysis Revisited. ECRYPT II Research Meeting, September 2011.

   – See p. 203.

9. Nicky Mouha, Qingju Wang, Dawu Gu, and Bart Preneel. Differential and Linear Cryptanalysis using Mixed-Integer Linear Programming. COSIC seminar, November 2011.

   – See p. 229.

10. Andrey Bogdanov, Simon Knellwolf, Willi Meier, Nicky Mouha, and Vesselin Velichkov. D.SYM.9: New developments in symmetric key cryptanalysis. ECRYPT II Symlab Report, December 2011.

Publication Chapter

Finding Collisions for a 45-Step Simplified HAS-V

Publication Data

Nicky Mouha, Christophe De Cannière, Sebastiaan Indesteege, and Bart Preneel. Finding Collisions for a 45-Step Simplified HAS-V. In Heung Youl Youm and Moti Yung, editors, *WISA*, volume 5932 of *Lecture Notes in Computer Science*, pages 206–225. Springer, 2009.

Contributions

- Main author. The paper is an extension and generalization of work performed by Christophe De Cannière and Christian Rechberger for SHA-1 [2], applied to a simplified variant of the hash function HAS-V. The use of double conditions was suggested by Sebastiaan Indesteege.

Finding Collisions for a 45-Step Simplified HAS-V*

Nicky Mouha[1,2,†], Christophe De Cannière[1,2,‡], Sebastiaan Indesteege[1,2,§], and Bart Preneel[1,2]

[1] Department of Electrical Engineering ESAT/SCD-COSIC, Katholieke Universiteit Leuven. Kasteelpark Arenberg 10, B-3001 Heverlee, Belgium.
[2] Interdisciplinary Institute for BroadBand Technology (IBBT), Belgium.

Abstract. Recent attacks on hash functions start by constructing a differential characteristic. By finding message pairs that satisfy this characteristic, a collision can be found. This paper describes the method of De Cannière and Rechberger to construct generalized characteristics for SHA-1 in more detail. This method is further generalized and applied to a simplified variant of the HAS-V hash function. Using these techniques, a characteristic for 45 steps is found, requiring an effort of about 2^{46} compression function evaluations to find a colliding message pair. A lot of the message bits can still be freely chosen when using this characteristic, greatly increasing its usefulness.

Keywords: Cryptanalysis, hash function, HAS-V, collision.

1 Introduction

Hash functions are an important building block in cryptography. As described in [15], these are functions h that convert an input m of arbitrary length in a deterministic way to a fixed-length output $h(m)$. It is crucial that a number of security properties are satisfied, one of which is the infeasibility of finding collisions (collision resistance). A collision consists of two input values m, m' where $m \neq m'$ for which $h(m) = h(m')$.

Recent attacks by Wang et al. on the widely used hash functions MD4 [19], MD5 [21], RIPEMD [19] and SHA-1 [20], as well as other hash functions, show that it is possible to find collisions for these hash functions much faster than expected by the birthday paradox [15]. In response to these attacks, NIST has launched a competition to find a new hash function standard [13]. Although a lot

*This work was supported in part by the IAP Program P6/26 BCRYPT of the Belgian State (Belgian Science Policy), and in part by the European Commission through the ICT program under contract ICT-2007-216676 ECRYPT II.

†This author is funded by a research grant of the Institute for the Promotion of Innovation through Science and Technology in Flanders (IWT-Vlaanderen).

‡Postdoctoral Fellow of the Research Foundation – Flanders (FWO)

§F.W.O. Research Assistant, Fund for Scientific Research – Flanders (Belgium).

of cryptanalysis effort is directed to these submissions, we feel that it is still very important to analyze existing hash function standards.

These recent collisions have lead to several practical attacks on network applications. For POP3 [6], it is is possible to mount a password recovery attack. Together with IMAP [1], POP3 [12] is one of the most used protocols for retrieving e-mail. Very recently, a rogue CA certificate was created using a collision for MD5 [17,18]. This certificate allows an attacker to impersonate any website on the Internet secured by HTTPS [16], including websites for banking and e-commerce.

The hash function HAS-V [14] is similar in structure to these hash functions. HAS-V fulfills the need of the KCDSA [7], the Korea Certificate-based Digital Signature Algorithm, to use a hash function with a variable digest size. The only cryptanalytic results on HAS-V known to us are described in [11]. Results using the recent attacks on hash functions have not been published before. The cryptanalysis of a simplified variant of HAS-V is the subject of this paper.

Recent attacks on hash functions focus on the construction of a differential characteristic, that allows collisions to be found with a good probability by finding messages m, m' that satisfy this characteristic. Characteristics are often constructed in an ad hoc way, which does not give any insight into the application of these attacks to other hash functions. This emphasizes the need for automated methods.

One such method, introduced in [2], is further generalized and applied to the simplified HAS-V. Using this method, we found a characteristic for a 45-step collision with an expected work factor of $2^{75.84}$ step function evaluations, which is given in Table 12. Further improvements lead to the better characteristic shown in Table 13, which has a work factor of $2^{51.53}$, making a collision finding attack feasible. If the cost of one step function evaluation is about 2^{-5} compression function evaluations, these work factors are equivalent to about 2^{71} and 2^{46} compression function evaluations, respectively. Note that a lot of bits in the message words can still be freely chosen.

Notation is defined in Table 1. In Sect. 2, a description of a simplified variant of HAS-V is given. An alternative, cyclic description of this hash function is provided as well. The technique for finding NL-characteristics of [2] is further explained, generalized and applied to HAS-V in Sect. 3. Techniques for improving NL-characteristics are laid out in Sect. 4, where good NL-characteristics for a 45-step simplified HAS-V are obtained as well. A conclusion and suggestions for future work are given in Sect. 5.

Appendix A lists the NL-characteristics we obtained. To assist the reader in understanding the more abstract explanation of the graph method in this paper, a simple example is given in Appendix B. Although this method is extensively used to attack SHA-1 in [2], this paper is the first to fully explain it.

2 A Simplified HAS-V

The hash function HAS-V [14] splits a 1024-bit message block into two 512-bit message blocks, which are then processed in two streams. The rounds of each stream alternately use message words of the first and the second 512-bit block. In our simplified variant of HAS-V, the right stream is omitted, as well as rounds in the left stream that depend on message words of the second 512-bit message block. As recent collision finding attacks are applied to hash functions with only one stream, simplifying the hash function in this way allows us to focus more easily on the main concepts of these recent attacks. For the same reason, the optional output tailoring is not applied. All other properties of HAS-V are left intact. A description of this simplified HAS-V is now given.

2.1 Description

The input message is padded and split into 512-bit message blocks. A 3-round compression function with 20 steps per round is applied to each of these 512-bit message blocks. This compression function $g(m, h)$ uses a 160-bit chaining input h and a 512-bit message block m as its inputs. The chaining input h_{n+1} of the next call of the compression function is calculated as $h_n + g(m, h_n)$. Here, the addition is done in blocks of 32-bit words, using a total of five adders modulo 2^{32}. The chaining variables for the first compression function call are set to fixed values, referred to as the IV. They are shown in Table 2. The last chaining input h represents the hash value.

Given a 512-bit message block m, consisting of 16 32-bit message words m_i, four extra message words, referred to as XOR-words, are derived from these message words for every round, as specified in Table 3. The extended message words w_i consist of the message words m_i followed by the four XOR-words.

Table 4 shows how the expanded message words W_t are derived as a reordering of the extended message words w_i for every round.

Figure 1 gives a schematic representation of the HAS-V step function, which is also described by

$$\begin{cases} A_{t+1} \leftarrow (A_t \lll S_t) + f_j(B_t, C_t, D_t, E_t) + W_t + K_t \ , \\ B_{t+1} \leftarrow A_t \ , \\ C_{t+1} \leftarrow B_t \ggg 2 \ , \\ D_{t+1} \leftarrow C_t \ , \\ E_{t+1} \leftarrow D_t \ . \end{cases} \tag{1}$$

Here, f_j represents a Boolean function, different for every round j:

$$\begin{aligned} f_1(B, C, D, E) &\triangleq (B \wedge C) \oplus (\neg B \wedge D) \oplus (C \wedge E) \oplus (D \wedge E) \ , \\ f_2(B, C, D, E) &\triangleq (B \wedge C) \oplus (\neg B \wedge E) \oplus D \ , \\ f_3(B, C, D, E) &\triangleq (\neg B \wedge C) \oplus (B \wedge D) \oplus (C \wedge E) \oplus (D \wedge E) \ . \end{aligned} \tag{2}$$

Table 1 – Notation

notation	description
$x \parallel y$	concatenation of the binary strings x and y
$x \wedge y$	bitwise AND of x and y
$x \vee y$	bitwise OR of x and y
$x \oplus y$	bitwise XOR of x and y
$\neg x$	bitwise NOT of x
$x \lll s$	rotation of x to the left by s positions
$x \ggg s$	rotation of x to the right by s positions
$x + y$	addition of x and y modulo 2^{32} (in text)
$x \boxplus y$	addition of x and y modulo 2^{32} (in figures)
$x[i]$	bit selection: 0 if $(x \wedge 2^i) \equiv 0$, 1 otherwise

Table 2 – The IV values for the simplified HAS-V

	A	B	C	D	E
IV	0x67452301	0xEFCDAB89	0x98BADCFE	0x10325476	0xC3D2E1F0

Table 3 – Calculation of the XOR-words for the simplified HAS-V

	w_{16}	w_{17}	w_{18}	w_{19}
Round 1	$w_0 \oplus w_1 \oplus w_2 \oplus w_3$	$w_4 \oplus w_5 \oplus w_6 \oplus w_7$	$w_8 \oplus w_9 \oplus w_{10} \oplus w_{11}$	$w_{12} \oplus w_{13} \oplus w_{14} \oplus w_{15}$
Round 2	$w_3 \oplus w_6 \oplus w_9 \oplus w_{12}$	$w_{15} \oplus w_2 \oplus w_5 \oplus w_8$	$w_{11} \oplus w_{14} \oplus w_1 \oplus w_4$	$w_7 \oplus w_{10} \oplus w_{13} \oplus w_0$
Round 3	$w_{12} \oplus w_5 \oplus w_{14} \oplus w_7$	$w_0 \oplus w_9 \oplus w_2 \oplus w_{11}$	$w_4 \oplus w_{13} \oplus w_6 \oplus w_{15}$	$w_8 \oplus w_1 \oplus w_{10} \oplus w_3$
Round 4	$w_7 \oplus w_2 \oplus w_{13} \oplus w_8$	$w_3 \oplus w_{14} \oplus w_9 \oplus w_4$	$w_{15} \oplus w_{10} \oplus w_5 \oplus w_0$	$w_{11} \oplus w_6 \oplus w_1 \oplus w_{12}$
Round 5	$w_{15} \oplus w_9 \oplus w_5 \oplus w_3$	$w_{12} \oplus w_8 \oplus w_6 \oplus w_2$	$w_{13} \oplus w_{11} \oplus w_7 \oplus w_1$	$w_{14} \oplus w_{10} \oplus w_4 \oplus w_0$

Table 4 – The message expansion for the simplified HAS-V

t	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Round 1	18	0	1	2	3	19	4	5	6	7	16	8	9	10	11	17	12	13	14	15
Round 2	18	12	5	14	7	19	0	9	2	11	16	4	13	6	15	17	8	1	10	3
Round 3	18	15	9	5	3	19	12	8	6	2	16	13	11	7	1	17	14	10	4	0

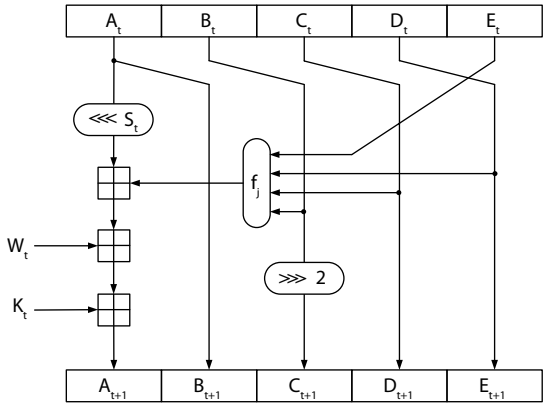


Figure 1 – The HAS-V step function

Table 5 – Constant K_t for the simplified HAS-V

	Round 1	Round 2	Round 3
K_t	0x00000000	0x6ED9EBA1	0xA953FD4E

In every step a constant K_t, different for every round, is added. These are listed in Table 5. The rotation value S_t is different for every step of a round. They are given in Table 6.

2.2 Cyclic Description

Using the step function of the simplified HAS-V, five internal variables A_t, B_t, C_t, D_t and E_t are obtained from five previous internal variables, A_{t-1}, B_{t-1}, C_{t-1}, D_{t-1} and E_{t-1}. As $B_t \equiv A_{t-1}$, $C_t \equiv A_{t-2} \ggg 2$, $D_t \equiv A_{t-3} \ggg 2$ and $E_t \equiv A_{t-4} \ggg 2$, it is sufficient to keep track of only A_t when calculating the step functions. These A_t, preceded by the IV values, are denoted by Q_t. A similar cyclic formulation was proposed for MD5 in [4], however there Q_t refer to values of B_t. The compression function can then be formulated alternatively for $t = 0, \ldots, 59$ as:

$$Q_{t+1} \leftarrow (Q_t \lll S_i) + f_j(Q_{t-1}, Q_{t-2} \ggg 2, Q_{t-3} \ggg 2, Q_{t-4} \ggg 2) + W_t + K_t \ . \quad (3)$$

The values of Q_t for $t = -4, \ldots 0$ are derived from the IV:

$$(Q_{-4}, Q_{-3}, Q_{-2}, Q_{-1}, Q_0) \leftarrow (E \lll 2, D \lll 2, C \lll 2, B, A) \ . \quad (4)$$

Table 6 – Rotation value S_t for the simplified HAS-V

t	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
S_t	5	11	7	13	15	6	13	9	5	11	7	12	8	15	13	8	15	6	7	14

Table 7 – All possible conditions for $(X[i], X'[i])$

	$(0,0)$	$(1,0)$	$(0,1)$	$(1,1)$
?	✓	✓	✓	✓
-	✓	-	-	✓
x	-	✓	✓	-
0	✓	-	-	-
u	-	✓	-	-
n	-	-	✓	-
1	-	-	-	✓
#	-	-	-	-

	$(0,0)$	$(1,0)$	$(0,1)$	$(1,1)$
3	✓	✓	-	-
5	✓	-	✓	-
7	✓	✓	✓	-
A	-	✓	-	✓
B	✓	✓	-	✓
C	-	-	✓	✓
D	✓	-	✓	✓
E	-	✓	✓	✓

It is this cyclic formulation of the simplified HAS-V that will be used from now on in this text.

3 NL-characteristics

For collision attacks, non-linear characteristics (NL-characteristics) start with chaining input and output difference zero. Differences are introduced via the message input, which then cancel themselves out with a sufficiently high probability by following the characteristic. High-probability characteristics are crucial for building fast collision-finding attacks, yet not much is known about their construction. They are often generated manually, using a great deal of intuition and experience. This paper further improves the results from [2], where an automated method is described for constructing these NL-characteristics.

In this paper, NL-characteristics are applied to the simplified HAS-V. An NL-characteristic is a set of conditions $\nabla Q_{-4}, \ldots \nabla Q_T$ and $\nabla W_0 \ldots \nabla W_{T-1}$. Each $\nabla X[i]$ represents a set of possible combinations for $(X[i], X[i]')$, as shown in Table 7. The number of steps T is left variable to be able to study step-reduced versions of this simplified HAS-V. In this paper, results will be obtained for $T = 45$.

3.1 Representation of Conditions on One Bit $\nabla Q_{t+1}[i]$

The step function (3) can be written as follows for every bit, for $0 \leq t < T$ and $0 \leq i < 32$. Indices i are calculated modulo 32. The carry input of the addition is denoted by $C_{t,i}$, the carry output by $C_{t+1,i+1}$.

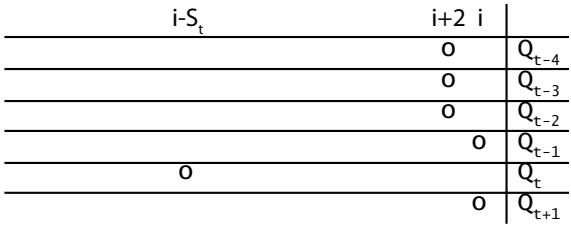


Figure 2 – Calculation of $Q_{t+1}[i]$ from $Q_t[i-S_t]$, $Q_{t-1}[i]$, $Q_{t-1}[i+2]$, $Q_{t-1}[i+2]$ and $Q_{t-1}[i+2]$

$$C_{t+1,i+1} \parallel Q_{t+1}[i] \leftarrow Q_t[i - S_t] + f_j(Q_{t-1}[i], Q_{t-2}[i+2], Q_{t-3}[i+2], Q_{t-4}[i+2]) + W_t[i] + K_t[i] + C_{t,i} \ . \tag{5}$$

To calculate $Q_{t+1}[i]$, the bit positions of the previous state words Q_{t-k} ($0 \leq k < 5$) are schematically represented in Fig. 2.

In the step function for every bit (5), a single, large addition is used with a carry input and output. The resulting carry states $(C_{t+1,i+1}, C'_{t+1,i+1})$ are then the only way in which adjacent bits of the same message word pair (W_t, W'_t) or internal states (Q_{t+1}, Q'_{t+1}) interact. If a particular carry state $(C_{t+1,i+1}, C'_{t+1,i+1})$ cannot occur as the output for the calculation of this bit $(Q_{t+1,i}, Q'_{t+1,i})$, nor as the input of the calculation of the next bit $(Q_{t+2,i+1}, Q'_{t+2,i+1})$, this combination of carries is said to be invalid.

For the calculation of every $\nabla Q_{t+1}[i]$, all valid combinations of $(C_{t,i}, C'_{t,i})$, (Q_{t-k}, Q'_{t-k}) for $0 \leq k < 5$ and (W_t, W'_t) are represented by an edge in Fig. 3. Imposing new conditions on $\nabla Q_{t+1}[i]$ will lead to the elimination of some of these edges.

The Boolean function f_j and the constant bit $K_t[i]$ are fixed for one bit position. Therefore, they are not included on the edges in Fig. 3. Each of the 2^{16} input bits of the edges then completely determines the six output bits $(Q_{t+1}[i], Q'_{t+1}[i])$ and $(C_{t,i+1}, C'_{t,i+1})$.

3.2 Propagation of Conditions for Every Word ∇Q_{t+1}

Initially, all input conditions are allowed for all bits. As this implies that all outputs are allowed for every bit, this is a self-consistent state. However, as soon as some restrictions are imposed on a bit, this may affect other bits. We refer to this mechanism as the propagation of conditions.

To calculate the possible conditions for every word ∇Q_{t+1} for $0 \leq t < T$, it is necessary to do both a forward and a backward propagation over all conditions $\nabla Q_{t+1}[i]$ for $0 \leq i < 32$. In Fig. 3, every possible input combination is shown as an

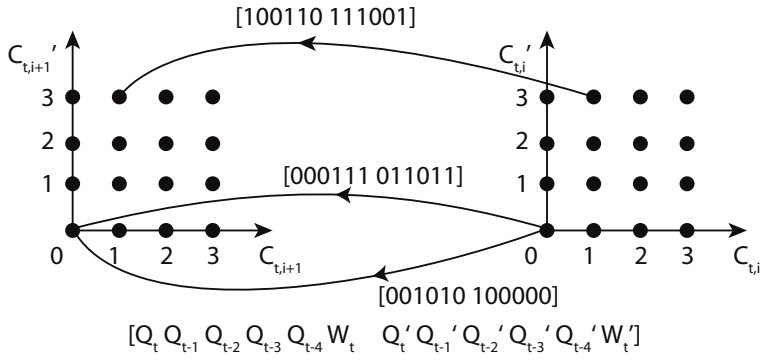


Figure 3 – Explanation of the edges in the graph. When adding four bits (and the carry input), the carry output $C_{t,i}$ can be 0, 1, 2 or 3. The addition of the corresponding four bits of the second message of the collision pair results in the carry $C'_{t,i}$.

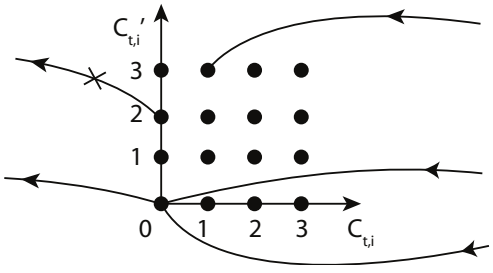


Figure 4 – Removing edges through forward propagation

edge connecting the input carries $(C_{t,i}, C'_{t,i})$ to the output carries $(C_{t,i+1}, C'_{t,i+1})$. Note that there can be multiple edges between two nodes.

In Fig. 4, a forward propagation (for $i = 0, 1, \ldots, 31$) is done where edges are removed if they start at an impossible input carry. In Fig. 5, a backward propagation is performed (for $i = 31, 30, \ldots, 0$) where edges are removed if the output carry is invalid. If necessary, the input conditions $\nabla Q_t[i - S_t]$, $\nabla Q_{t-1}[i]$, $\nabla Q_{t-2}[i+2]$, $\nabla Q_{t-3}[i+2]$,$\nabla Q_{t-4}[i+2]$ and $\nabla W_t[i]$, as well as the output condition $\nabla Q_{t+1}[i]$ in the NL-characteristic are updated. In this way, one word can affect the conditions of another word.

This step is repeated for every word ∇Q_{t+1} for $0 \leq t < T$, until further propagation would not remove additional edges or until at least one condition is inconsistent. Every time a message bit $W_t[i]$ is assigned a new value, message bits $W_{t'}[i]$ that are related by the message expansion, are updated as well if necessary. The remaining valid paths for one word are then shown in Fig. 6.

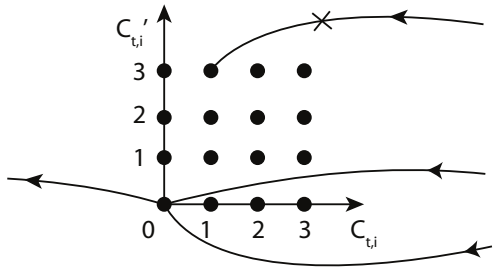


Figure 5 – Removing edges through backward propagation

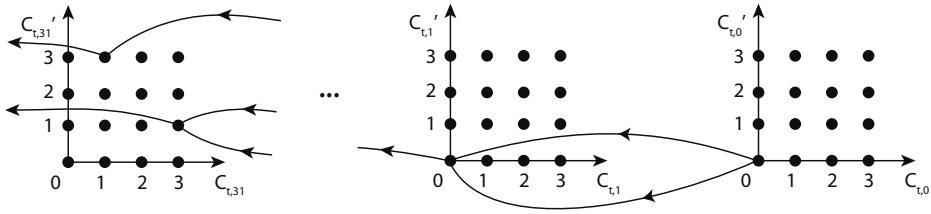


Figure 6 – Remaining valid paths for one word ∇Q_{t+1}

3.3 Double Conditions

Conditions that do not involve one pair of bits, but two pairs of bits, are referred to as "double conditions". The use of these is new to this paper. They are similar to Table 7, except that double conditions apply to four bits instead of two. Thus, there are 2^{16} possible double conditions, instead of 2^4.

For the simplified HAS-V, double conditions can be used in three locations for the calculation of one bit of ∇Q_{t+1}. These are shown in Fig. 7, as the result of the only possibilities of creating an overlap of at least two bits of Fig. 2 with a translated version of this pattern.

The use of the first double condition is explained, the other two cases are analogous. During the calculation of $\nabla Q_{t'+1}[2]$, a double condition is used to represent the possibilities of the joint occurrence of $\nabla Q_{t'+1}[2]$ and $\nabla Q_{t'-1}[2]$. When $\nabla Q_{t+1}[0]$ is calculated, it can be seen that the same double condition now also applies to the joint occurrence of $\nabla Q_{t-2}[2]$ and $\nabla Q_{t-4}[2]$. It is possible that this information leads to the removal of additional edges. If this is the case, the number of iterations needed to construct an NL-characteristic is lowered, and inconsistencies can be found sooner. In our implementation of the search for NL-characteristics, double conditions can be implemented with minimal overhead.

3.4 Work Factor

The work factor N_{w} of an NL-characteristic indicates the expected number of step function evaluations required to find a collision using this characteristic. When building NL-characteristics, the collision search is optimized by lowering the work factor. This concept was introduced in [2].

Message Freedom $F_{\mathrm{W}}(t)$

"Single-message modification" [21] (also known as "single-step modification" [19]) can be used during the search process, as there is still freedom left in the choice of several expanded message words W_t. Due to the constraints imposed by the XOR-words, this is not possible for each of the 20 message word pairs (W_t, W'_t) of the first round. Of the five message word pairs involved in the calculation of each XOR-word, only the first four can be chosen. The last message word pair cannot be freely chosen, but must equal the XOR of the four others.

The message freedom $F_{\mathrm{W}}(t)$ of a characteristic at step t is the number of ways to choose (W_t, W'_t), without violating any (linear) condition imposed by the message expansion, given fixed values of (W_j, W'_j) for $0 \leq j < t$.

The description of the simplified HAS-V indicates that $F_{\mathrm{W}}(t)$ is always 1 for $t = 10, 14, 15, 19$ and $t \geq 20$. For the other values of t, $F_{\mathrm{W}}(t)$ is the product of the number of possibilities for conditions $\nabla W_t[i]$ for $0 \leq i < 32$. This number of possibilities equals the number of checkmarks (✓) for the respective conditions in Table 7.

Uncontrolled Probability $P_{\mathrm{u}}(t)$

The uncontrolled probability $P_{\mathrm{u}}(t)$ of a characteristic at step t is the probability that the output (Q_{t+1}, Q'_{t+1}) follows the characteristic, given that all input pairs (Q_{t-k}, Q'_{t-k}) for $0 \leq k < 5$ and message word pairs (W_t, W'_t) follow this characteristic as well:

$$P_{\mathrm{u}}(t) = P\big((Q_{t+1}, Q'_{t+1}) \in \nabla Q_{t+1} \mid (Q_{t-k}, Q'_{t-k}) \in \nabla Q_{t-k} \text{ for } 0 \leq k < 5, \text{ and } (W_t, W'_t) \in \nabla W_t\big) \ . \quad (6)$$

This probability can be calculated as the number of remaining paths of Fig. 6, divided by the number of paths for which only the input pairs (Q_{t-k}, Q'_{t-k}) for $0 \leq k < 5$ and the message word pairs (W_t, W'_t) follow the characteristic, but not necessarily the output pair (Q_{t+1}, Q'_{t+1}).

Controlled Probability $P_{\mathrm{c}}(t)$

The controlled probability $P_{\mathrm{c}}(t)$ of a characteristic at step t is the probability that there exists at least one pair of message words (W_t, W'_t) following the characteristic,

such that the output (Q_{t+1}, Q'_{t+1}) follows the characteristic, given that all input pairs (Q_{t-k}, Q'_{t-k}) for $0 \leq k < 5$ do as well:

$$P_{\mathrm{c}}(t) = P\big(\exists (W_t, W'_t) \in \nabla W_t : (Q_{t+1}, Q'_{t+1}) \in \nabla Q_{t+1} \mid (Q_{t-k}, Q'_{t-k}) \in \nabla Q_{t-k} \text{ for } 0 \leq k < 5\big) \ . \quad (7)$$

A graph is made for every bit i for the calculation of $(Q_{t+1}[i], Q'_{t+1}[i])$ to determine this probability. Each node of the graph is a carry mask, indicating which of the 16 possible values of $(C_{t,i}, C'_{t,i})$ can occur. Thus, a carry mask can have 2^{16} possible values. Note the analogy with Table 7, where the possible combinations of $(X[i], X[i]')$ are shown.

Let n be the number of possibilities for $(Q_{t-k}[i], Q'_{t-k}[i]) \in \nabla Q_{t-k}[i] \text{for} 0 \leq k < 5$. For each possibility, we run through all carries $(C_{t,i}, C'_{t,i})$ and all message bit pairs $(W_t[i], W'_t[i])$. A binary 16×16 matrix indicates which transition possibilities from $(C_{t,i}, C'_{t,i})$ to $(C_{t,i+1}, C'_{t,i+1})$ can occur.

Using this 16×16 transition matrix, we can calculate the possible carry masks for bit $i+1$ using the carry mask of bit i. For the least significant bit ($i = 0$), only one carry mask is possible: the carry is $(0, 0)$ with probability 1. Each of the edges in the graph has probability $1/n$. Unlike in Fig. 6, there is never more than one edge between two nodes. This step is repeated for every $(Q_{t-k}[i], Q'_{t-k}[i]) \in \nabla Q_{t-k}[i]$ for $0 \leq k < 5$.

This calculation is performed for bits $i = 0 \ldots 31$. We now consider the most significant bit ($i = 31$). One carry mask indicates that none of the carries $(C_{t,31}, C'_{t,31})$ are valid. $P_{\mathrm{c}}(t)$ then equals the sum of all the other carry masks.

Total Work Factor N_{w}

In the collision search tree, the average number of children of a node at step t is $F_{\mathrm{W}}(t) \cdot P_{\mathrm{u}}(t)$. Only a fraction $P_{\mathrm{c}}(t)$ of the nodes at step t have children at all. The search stops as the last step $T-1$ of the compression function is reached. We can thus obtain the following recursive relation for the expected number of nodes $N_{\mathrm{s}}(t)$ at every step of the compression function:

$$N_{\mathrm{s}}(t) = \begin{cases} 1 & \text{for } t = T-1 \ , \\ \max\left(N_{\mathrm{s}}(t+1) \cdot F_{\mathrm{W}}^{-1}(t) \cdot P_{\mathrm{u}}^{-1}(t), P_{\mathrm{c}}^{-1}(t)\right) & \text{for } 0 \leq t < T-1 \ . \end{cases} \quad (8)$$

The total work factor is then given by

$$N_{\mathrm{w}} = \sum_{t=0}^{T-1} N_{\mathrm{s}}(t) \ . \quad (9)$$

In tables, the base 2 logarithms of $F_{\mathrm{W}}(t)$, $P_{\mathrm{u}}(t)$, $P_{\mathrm{c}}(t)$, $N_{\mathrm{s}}(t)$ and $N_{\mathrm{w}}(t)$ are shown.

Table 8 – Lowest Hamming weights found for L-characteristics, not taking the weight of ∇Q_{t+1} for $0 \leq t < 20$ into account

	collision	near-collision	pseudo-collision
40 steps	30	26	27
45 steps	75	68	65

A difference with [2], is that in this work, the double conditions of Sect. 3.3 are also taken into account in the calculation of the work factor. These are assumed to be included in the definitions of $P_{\mathrm{u}}(t)$ and $P_{\mathrm{c}}(t)$. This is because double conditions are used in the actual collision search as well. Implementing this is possible with minimal overhead, and can only improve N_{w}. Experimental results of using these double conditions will be given in Sect. 4.

4 Finding NL-characteristics for 45 Steps

To obtain a good NL-characteristic, Stage 1 of [2] consists of obtaining a sparse L-characteristic to use as a starting point. As can be seen in Table 8, no suitable L-characteristic could be found for 45 steps of the simplified HAS-V. The weight of the ∇Q_{t+1} for the first round is not taken into account, assuming for simplicity that these can all be satisfied by single-message modification.

To overcome this problem, we looked for message differences that are localized at a small number of steps of the internal states ∇Q_{t+1}. The Boolean functions f_j are particularly well suited to allow for NL-characteristics consisting of very short collision regions. It can be seen that both f_1 and f_3 allow any input difference to be either passed on or canceled out at the output. For $f_2(B, C, D, E)$, this is also the case for every input difference, except for an input difference at D, which will always lead to an output difference. The HAS-V specification [14] reveals that this is by design, in an attempt to satisfy the "Strict Avalanche Criterion (SAC)" [22]. As the attacker can choose both messages m, m' of a collision pair, he can control the output differences of the f-function at certain positions (either probabilistically, or by single- or multi-message modification). This allows for more freedom in the construction of NL-characteristics, while still keeping the probability of the characteristic high.

Differences in the message words m_i are only introduced in $m_{12}[0]$ and $m_{14}[0]$. Due to the message expansion, these differences can be found in $W_{16}[0]$, $W_{18}[0]$, $W_{21}[0]$, $W_{23}[0]$. Before and after this collision region, equality is imposed on the internal state words.

In the short collision region, all conditions for $\nabla Q_{t+1}[i]$ are still unrestricted ("?") at Stage 1.

Stages 2 and 3 are the same as in [2]. In Stage 2, unrestricted conditions ("?") are randomly chosen, and the requirement that they are equal ("-") is imposed.

Table 9 – The work factor N_w (in base 2 logarithm) after each of the four stages

	Stage 1	Stage 2	Stage 3	Stage 4
without double conditions	143.87	89.30	81.84	59.92
with double conditions	143.87	81.28	75.84	51.53

This stage is repeated several times, until a characteristic with a sufficiently low work factor is obtained. Further in Stage 2, conditions ("`x`") start to appear, which are replaced by either ("`u`") or ("`n`") when selected. In Stage 3, local optimizations are performed by going over all "`-`" conditions, and replacing them by "`0`" or "`1`" if this improves the work factor. By repeating Stage 3 several times, the work factor gradually decreases. The end result after Stage 3 is shown in Table 12, with corresponding work factor $N_w = 2^{75.84}$.

After Stage 3, adding a single extra condition will never decrease the work factor. It is possible, however, to reduce the work factor even further. This is done in an additional stage, Stage 4, not described in [2]. In Stage 4, not one, but several conditions are added locally, as long as they do not worsen the work factor. If adding multiple conditions improves the work factor, a minimal set of conditions is derived from these, that still lowers the work factor. This set is obtained by relaxing the additional conditions again, one by one, to see if they had any impact on the global work factor. Only the conditions of this minimal set are kept. Experiments show that it is even possible, that relaxing conditions decreases the work factor of the NL-characteristic. This fourth stage is also repeated several times. The end result is shown in Table 13, where a work factor N_w of $2^{51.53}$ is obtained. After the Stage 4, it is not possible to decrease the work factor by adding or relaxing a single condition.

Note that the characteristics obtained after every stage are not necessarily the best possible. Every stage can thus be performed several times, until a characteristic is found that is good enough.

Experimental results indicating the impact of these double conditions on N_w after each of the four stages, are shown in Table 9.

Although time limits did not allow us to find a colliding message pair, we have verified for reduced versions that the complexity estimates accurately reflect the actual search cost, both with and without the inclusion of double conditions.

5 Conclusion and Future Work

This paper shows how techniques developed for SHA-1 in [2] can be further improved and generalized for a simplified variant of the hash function HAS-V. This simplified variant consists of only a single stream.

For 45 steps of this simplified HAS-V, an NL-characteristic is constructed,

requiring about $2^{51.53}$ step function evaluations, or about 2^{46} compression function evaluations, to find a collision. A lot of the message bits can still be freely chosen when using this characteristic.

Stage 1 of method of De Cannière and Rechberger [2], the search for a good L-characteristic, is replaced by the requirement that collisions occur in a very short region. As the method described in this paper can be applied without finding good L-characteristics first, it might be used for hash functions such as RIPEMD-160 [3], for which also no good L-characteristics were found [10].

"Double conditions" are introduced as conditions for two pair of bits. They can be used to speed up the actual collision search.

An extra stage, Stage 4, is introduced to further improve the work factor for finding a collision. It is shown how this additional stage can reduce the work factor from $2^{75.84}$ step function evaluations, or about 2^{71} compression function evaluations, in Table 12, to $2^{51.53}$, or about 2^{46} compression function evaluations, in Table 13.

6 Acknowledgments

The authors would like to thank their colleagues at COSIC, and the symmetric cryptography subgroup in particular, for their useful comments and suggestions. Special thanks go to Vesselin Velichkov, who greatly helped in improving the clarity of this paper.

Several techniques in this paper were already used in the cryptanalysis of SHA-1 by Christophe De Cannière and Christian Rechberger [2], but had not been explained before. The authors are greatly indebted to Christian Rechberger, not only for his useful comments and suggestions, but also for allowing us build upon his previous work for SHA-1.

References

[1] M. Crispin. Internet Message Access Protocol - Version 4rev1. RFC 3501 (Proposed Standard), March 2003. `http://www.ietf.org/rfc/rfc3501.txt`.

[2] C. De Cannière and C. Rechberger. Finding SHA-1 Characteristics: General Results and Applications. In X. Lai and K. Chen, editors, *ASIACRYPT*, volume 4284 of *Lecture Notes in Computer Science*, pages 1–20. Springer, 2006.

[3] H. Dobbertin, A. Bosselaers, and B. Preneel. RIPEMD-160: A Strengthened Version of RIPEMD. In D. Gollmann, editor, *FSE*, volume 1039 of *Lecture Notes in Computer Science*, pages 71–82. Springer, 1996.

[4] P. Hawkes, M. Paddon, and G. G. Rose. Musings on the Wang et al. MD5 Collision. Cryptology ePrint Archive, Report 2004/264, 2004. `http://eprint.iacr.org/`.

[5] S. Indesteege and B. Preneel. Practical Collisions for EnRUPT. In O. Dunkelman, editor, *FSE*, volume 5665 of *Lecture Notes in Computer Science*, pages 246–259. Springer, 2009.

[6] G. Leurent. Message Freedom in MD4 and MD5 Collisions: Application to APOP. In A. Biryukov, editor, *FSE*, volume 4593 of *Lecture Notes in Computer Science*, pages 309–328. Springer, 2007.

[7] C. H. Lim and P. J. Lee. A Study on the Proposed Korean Digital Signature Algorithm. In K. Ohta and D. Pei, editors, *ASIACRYPT*, volume 1514 of *Lecture Notes in Computer Science*, pages 175–186. Springer, 1998.

[8] H. Lipmaa and S. Moriai. Efficient Algorithms for Computing Differential Properties of Addition. In M. Matsui, editor, *FSE*, volume 2355 of *Lecture Notes in Computer Science*, pages 336–350. Springer, 2001.

[9] H. Lipmaa, J. Wallén, and P. Dumas. On the Additive Differential Probability of Exclusive-Or. In B. K. Roy and W. Meier, editors, *FSE*, volume 3017 of *Lecture Notes in Computer Science*, pages 317–331. Springer, 2004.

[10] F. Mendel, N. Pramstaller, C. Rechberger, and V. Rijmen. On the Collision Resistance of RIPEMD-160. In S. K. Katsikas, J. Lopez, M. Backes, S. Gritzalis, and B. Preneel, editors, *ISC*, volume 4176 of *Lecture Notes in Computer Science*, pages 101–116. Springer, 2006.

[11] F. Mendel and V. Rijmen. Weaknesses in the HAS-V Compression Function. In K.-H. Nam and G. Rhee, editors, *ICISC*, volume 4817 of *Lecture Notes in Computer Science*, pages 335–345. Springer, 2007.

[12] J. Myers and M. Rose. Post Office Protocol - Version 3. RFC 1939 (Standard), May 1996. `http://www.ietf.org/rfc/rfc1939.txt`.

[13] National Institute of Standards and Technology. Announcing Request for Candidate Algorithm Nominations for a New Cryptographic Hash Algorithm (SHA-3) Family. *Federal Register*, 27(212):62212–62220, November 2007. `http://csrc.nist.gov/groups/ST/hash/documents/FR_Notice_Nov07.pdf`.

[14] N. K. Park, J. H. Hwang, and P. J. Lee. HAS-V: A New Hash Function with Variable Output Length. In D. R. Stinson and S. E. Tavares, editors, *Selected Areas in Cryptography*, volume 2012 of *Lecture Notes in Computer Science*, pages 202–216. Springer, 2000.

[15] B. Preneel. *Analysis and design of cryptographic hash functions*. PhD thesis, Katholieke Universiteit Leuven, 1993.

[16] E. Rescorla. HTTP Over TLS. RFC 2818 (Informational), May 2000. `http://www.ietf.org/rfc/rfc2818.txt`.

[17] A. Sotirov, M. Stevens, J. Appelbaum, A. Lenstra, D. A. Molnar, D. A. Osvik, and B. de Weger. MD5 considered harmful today: Creating a rogue CA certificate, December 2008. 25th Chaos Communications Congress, Berlin, Germany.

[18] M. Stevens, A. K. Lenstra, and B. de Weger. Chosen-Prefix Collisions for MD5 and Colliding X.509 Certificates for Different Identities. In M. Naor, editor, *EUROCRYPT*, volume 4515 of *Lecture Notes in Computer Science*, pages 1–22. Springer, 2007.

[19] X. Wang, X. Lai, D. Feng, H. Chen, and X. Yu. Cryptanalysis of the Hash Functions MD4 and RIPEMD. In R. Cramer, editor, *EUROCRYPT*, volume 3494 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2005.

[20] X. Wang, Y. L. Yin, and H. Yu. Finding Collisions in the Full SHA-1. In V. Shoup, editor, *CRYPTO*, volume 3621 of *Lecture Notes in Computer Science*, pages 17–36. Springer, 2005.

[21] X. Wang and H. Yu. How to Break MD5 and Other Hash Functions. In R. Cramer, editor, *EUROCRYPT*, volume 3494 of *Lecture Notes in Computer Science*, pages 19–35. Springer, 2005.

[22] A. F. Webster and S. E. Tavares. On the Design of S-Boxes. In H. C. Williams, editor, *CRYPTO*, volume 218 of *Lecture Notes in Computer Science*, pages 523–534. Springer, 1985.

A NL-characteristics

The NL-characteristics obtained after Stage 3 of Sect. 4 are shown in Table 12. After Stage 4, Table 13 is obtained. The work factor N_w improves from $2^{75.84}$ to $2^{51.53}$.

B A Two-bit Example

B.1 Introduction

Let n denote the word size in bits. We will write the differential probability of addition modulo 2^n as $\mathrm{xdp}^+(\alpha, \beta \to \gamma)$, where α, β and γ are bitstrings, most significant bit first. The best known method to find xdp^+ was an exponential-in-n

calculation, before Lipmaa and Moriai introduced their algorithm in [8]. In [9], it was shown how xdp^+ can be calculated as a series of matrix multiplications in linear time in n.

In this section, we will calculate the $\mathrm{xdp}^+(11, 01 \to 10)$ by representing the addition as a graph and applying dynamic programming. We then show the relation of this graph method with [9], as both algorithms can be implemented in $\mathcal{O}(n)$ by using matrix multiplications. Afterwards, we mention several improvements and extensions to the graph method. Although this two-bit example may seem contrived, we found a fully worked-out example to be very useful to help understand the more abstract explanation of Fig. 3-6 in Sect. 3. There, an extension of the graph method is used to represent the step update function of HAS-V.

B.2 Visualizing $\mathrm{xdp}^+(11, 01 \to 10)$ in a Graph

For $\mathrm{xdp}^+(\alpha_1 \parallel \alpha_0, \beta_1 \parallel \beta_0 \to \gamma_1 \parallel \gamma_0) \triangleq \mathrm{xdp}^+(11, 01 \to 10)$, we consider two additions, $z = x + y$ and $z' = x' + y'$, as shown in Fig. 8. For this particular xdp^+, we define the input differences for the least significant bits ($\alpha_0 = x_0 \oplus x'_0 = 1$ and $\beta_0 = y_0 \oplus y'_0 = 1$), and for the most significant bits ($\alpha_1 = x_1 \oplus x'_1 = 1$ and $\beta_1 = y_1 \oplus y'_1 = 0$). We assume that all valid inputs x, x' and y, y' are uniformly distributed. We then find $\mathrm{xdp}^+(11, 01 \to 10)$ as the probability that the output has difference $\gamma_0 = z_0 \oplus z'_0 = 0$ and $\gamma_1 = z_1 \oplus z'_1 = 1$.

The calculation for the least significant bits is shown in Table 10. As there is no carry input for the least significant bits, we only consider $C_0 = C'_0 = 0$. We list all values that satisfy the input conditions (α_0 and β_0) for the least significant bit. Note that the output condition ($\gamma_0 = z_0 \oplus z'_0 = 0$) is satisfied as well for all valid inputs $(C_0, C'_0, x_0, y_0, x'_0, y'_0)$.

Table 10 – The summation for the least significant bits (z_0, z'_0), where $\alpha_0 = x_0 \oplus x'_0 = 1$ and $\beta_0 = y_0 \oplus y'_0 = 1$

C_0	C'_0	x_0	y_0	x'_0	y'_0	C_1	C'_1	z_0	z'_0	α_0	β_0	γ_0
0	0	0	0	1	1	0	1	0	0	1	1	0
0	0	0	1	1	0	0	0	1	1	1	1	0
0	0	1	0	0	1	0	0	1	1	1	1	0
0	0	1	1	0	0	1	0	0	0	1	1	0

We then draw each of these input values as the four rightmost edges in the graph of Fig. 9. Every edge is labeled with the input conditions $[x_0 \ y_0 \ \ x'_0 \ y'_0]$, and starts at (C_0, C_0). Together, these uniquely determine (z_0, z'_0) and (C_1, C_1). For now, the reader can ignore that some lines are dashed.

Next, we do the calculation for the most significant bits, as shown in Table 11. We again list all values that satisfy the input conditions (α_1 and β_1). Note that now, several carry inputs (C_1, C'_1) are possible. The output condition ($\gamma_1 = z_1 \oplus z'_1 = 1$) is not always satisfied, implying that $\mathrm{xdp}^+(11, 01 \to 10) < 1$.

Table 11 – The summation for the most significant bits (z_1, z_1'), where $\alpha_1 = x_1 \oplus x_1' = 1$ and $\beta_1 = y_1 \oplus y_1' = 0$

C_1	C_1'	x_1	y_1	x_1'	y_1'	C_2	C_2'	z_1	z_1'	α_1	β_1	γ_1
0	0	0	0	1	0	0	0	0	1	1	0	1
0	0	0	1	1	1	0	1	1	0	1	0	1
0	0	1	0	0	0	0	0	1	0	1	0	1
0	0	1	1	0	1	1	0	0	1	1	0	1
1	0	0	0	1	0	0	0	1	1	1	0	0
1	0	0	1	1	1	1	1	0	0	1	0	0
1	0	1	0	0	0	1	0	0	0	1	0	0
1	0	1	1	0	1	1	0	1	1	1	0	0
0	1	0	0	1	0	0	1	0	0	1	0	0
0	1	0	1	1	1	0	1	1	1	1	0	0
0	1	1	0	0	0	0	0	1	1	1	0	0
0	1	1	1	0	1	1	1	0	0	1	0	0

Again, we draw each of the inputs of Table 11 as edges in Fig. 9. To improve the readability, we use three separate coordinate systems on top of each other on the left of the figure. These should in fact overlap: the same nodes are represented three times. For example, four edges end in $(C_2, C_2') = (0, 0)$. If the output pairs are valid ($\gamma_1 = z_1 \oplus z_1' = 1$), we use full lines, and if they are invalid ($\gamma_1 = z_1 \oplus z_1' = 0$), dashed lines are used.

Due to backward propagation (explained in Fig. 5), the inputs $[x_0 \;\; y_0 \;\; x_0' \;\; y_0']$ with values [00 11] and [11 00] become dashed lines as well: they will eventually result in an incorrect output difference $\gamma_1 = 0$. The probability $\mathrm{xdp}^+(11, 01 \to 10)$ is then equal to the number of paths in the graph with valid inputs (x, x', y, y') and outputs z, z' (full lines), divided by the number of paths that have valid inputs $(x, x'y, y')$ (full or dashed lines). This ratio is equal to 8/16, or 1/2.

Note that storing this graph does not require a lot of memory. For every bit in the n-bit addition, we need to store 2^6 bits. Each of these 2^6 bits is either set to 1 if the input $(C_0, C_0', x_0, y_0, x_0', y_0')$ is valid, and 0 otherwise. For the entire n-bit addition, we thus need to store only $2^6 n$ bits; the memory requirement is $\mathcal{O}(n)$.

B.3 Calculating $\mathrm{xdp}^+(11, 01 \to 10)$ Using Matrix Multiplications

Similar to [9], we can calculate xdp^+ as a series of matrix multiplications. The graph of Fig. 9 can be seen as a first-order Markov chain. We have $[1 \;\; 0 \;\; 0 \;\; 0]^T$ as the initial distribution, as the input carry of the addition is $C_0 = C_0' = 0$ with probability 1. All three other input carries (C_0, C_0') have probability 0.

As the input conditions for $[x_0 \;\; y_0 \;\; x_0' \;\; y_0']$ are given, these specify the transition matrix of the Markov chain. Every column contains the transition probabilities for one carry input (C_0, C_0') to every carry output (C_1, C_1'). We left-multiply

the initial distribution by this transition matrix. We do the same for every subsequent bit of the n-bit addition.

Lastly, we sum all probabilities (by left-multiplying by [1 1 1 1]): the carry outputs (C_n, C'_n) are not used, so all of them are valid. This gives us the total probability of xdp^+.

B.4 Extending the Graph Method

As the reader may have noticed, the matrices of the previous section are larger than those in [9]. This is because we do not take the symmetry into account: although the value of $C_i \oplus C'_i$ would be sufficient, we keep track of the values of (C_i, C'_i). This symmetry exists because we restrict the input differences α, β and the output differences γ to exclusive-or differences.

The graph based method of the previous section, however, can also support the signed differences that were used for the cryptanalysis of MD5 [21], and as well as all the other generalized conditions of Table 7.

It is straightforward to generalize the graph method to the addition of three or more words. In this case, we extend each of the n adders of Fig. 8 to three or more input bits. This will increase the maximal values of the carry (C_i, C'_i): for example, the addition of four bits (and the carry input) can have a maximal carry output of 3.

In this case, value of the carry (C_i, C'_i) is equal to all output bits of the adder at position i, except the least significant bit. This can be seen as a variant of Fig. 8, where three or more bits are input to every adder.

In fact, the method can be generalized for any combination of additions, XORs and Boolean functions, as long as no rotations are present (except at the input or output). It is this calculation that was used for every step of SHA-1 in [2], and is also used for every step of HAS-V in this paper. By constructing higher-order Markov chains, the graph method was used in [5] to efficiently calculate the differential probability of a multiplication by 9, given by $\mathrm{xdp}^+(x, x \ll 3)$.

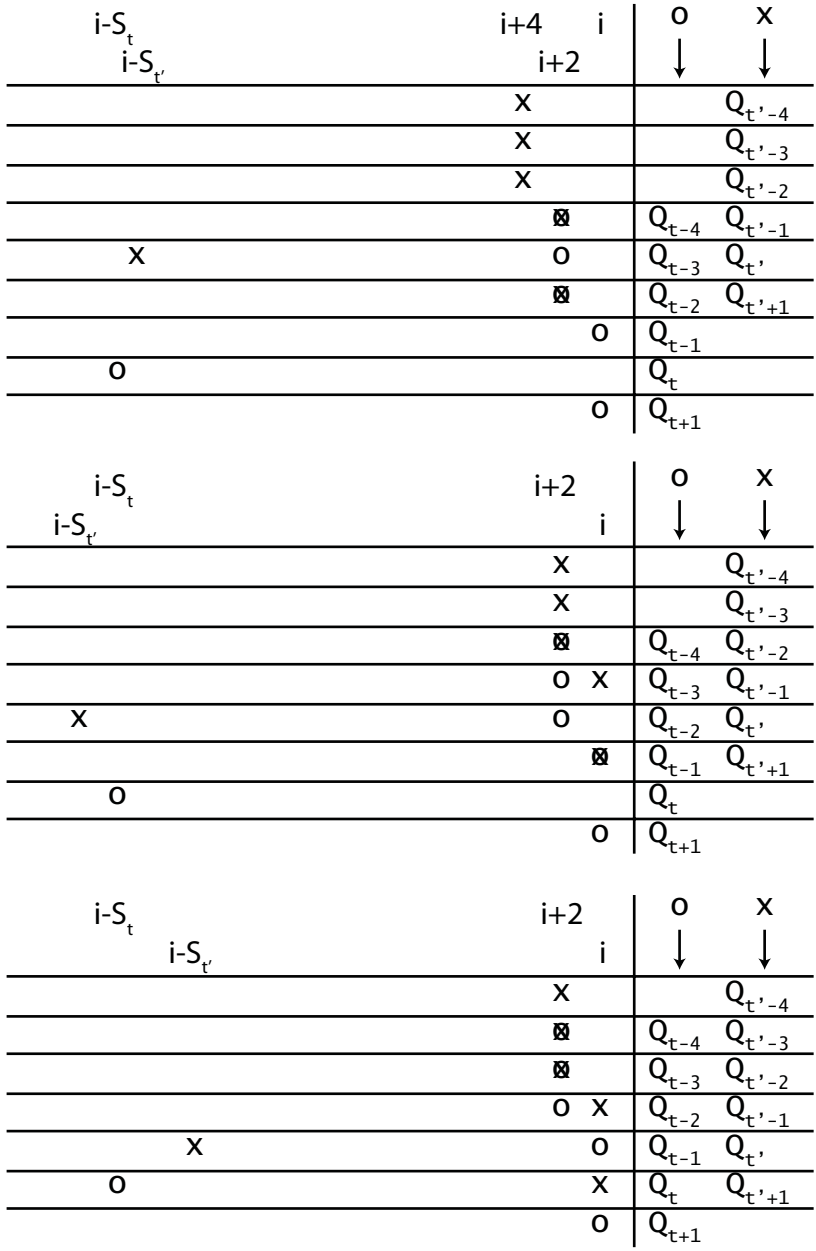


Figure 7 – Double conditions for the HAS-V step function, obtained as the only possible overlaps of at least two bits in Fig. 2 with a translated version of this pattern

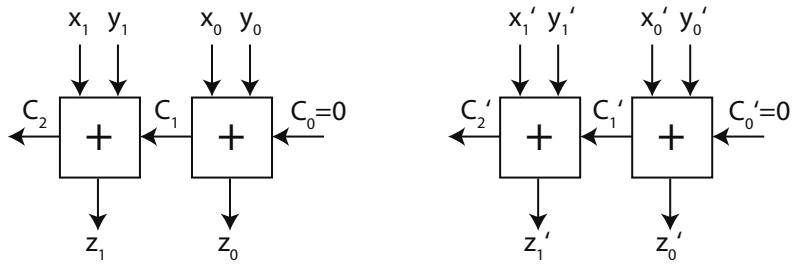


Figure 8 – Calculating $z = x + y$ and $z' = x' + y'$. All variables with subscripts represent one bit.

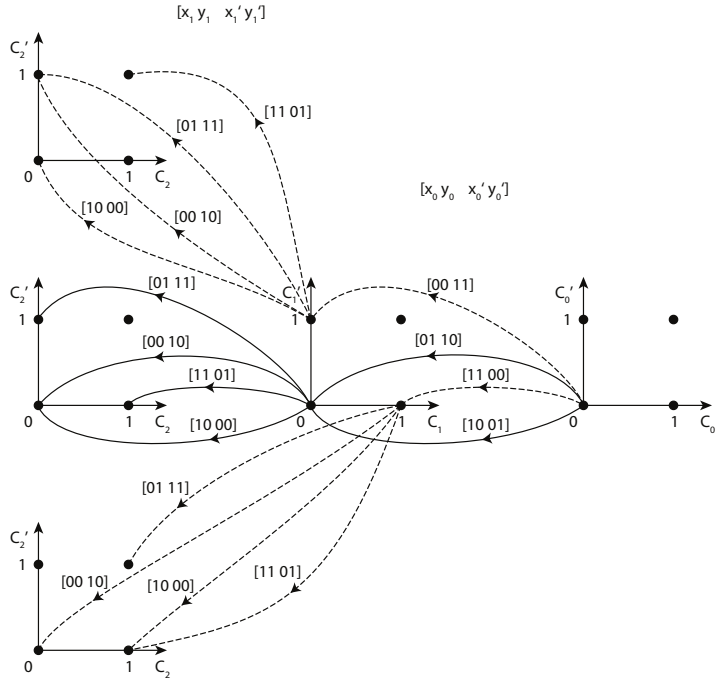


Figure 9 – Graph representation to calculate $\mathrm{xdp}^+(11, 01 \to 10)$. Only valid input pairs are shown. Full lines are used for the eight paths that have valid output pairs ($\gamma_1 = z_1 \oplus z_1' = 1$), and dashed lines are used for paths with invalid output pairs ($\gamma_1 = z_1 \oplus z_1' = 0$). As there are eight of each, the ratio gives $\mathrm{xdp}^+(11, 01 \to 10) = 8/16 = 1/2$. The three coordinate systems on top of each other on the left represent the same nodes three times. This makes the drawing more readable, however note that, for example, four edges end in $(C_2, C_2') = (0, 0)$.

Table 12 – NL-characteristic of 45 steps after Stage 3, work factor $N_{\mathrm{w}} = 2^{75.84}$

t	∇Q_{t+1}	∇W_t	F_{W}	$P_{\mathrm{u}}(t)$	$P_{\mathrm{c}}(t)$	$N_{\mathrm{s}}(t)$
-5	00001111010010111000011111000011					
-4	01000000110010010101000111011000					
-3	01100010111010110111001111111010					
-2	11101111110011011010101110001001					
-1	01100111010001010010001100000001					
0	----------	----------	32	0.00	0.00	0.00
1	----------	----------	32	0.00	0.00	0.00
2	----------	----------	32	0.00	0.00	0.00
3	----------	----------	32	0.00	0.00	0.00
4	----------	----------	32	0.00	0.00	0.00
5	----------	----------	32	0.00	0.00	0.00
6	----------	----------	32	0.00	0.00	0.00
7	----------	----------10	30	0.00	0.00	0.00
8	----------	----------	32	0.00	0.00	0.00
9	----------	----------	32	0.00	0.00	0.00
10	----------	----------	0	0.00	0.00	0.00
11	----------	----------	32	0.00	0.00	0.00
12	----------	----------	32	0.00	0.00	0.00
13	----------0---------	----------	32	-1.00	0.00	22.60
14	----------010001101100-	----------	0	-12.00	0.00	53.60
15	----------001011---1-0	----------	0	-9.00	0.00	41.60
16	---0----------unnnnnnnnnnnn	1----------u	30	-14.00	-1.00	32.60
17	0--1----------0100u111010	----------	32	-13.00	0.00	48.60
18	0--1------1---------uu-uu0001110	----------000-----u	28	-19.77	-7.77	67.60
19	01-unn--nnu----------11000nn10	0----------	0	-19.00	-1.97	75.83
20	n-u0uu001-000----------0-0000-10	----------1-101--00	0	-14.30	-2.30	56.83
21	u-u-0n11----0---------11-010--10	1----------u	0	-18.98	-6.42	42.53
22	--1-110---011----------0-n----11	----------10	0	-10.00	-1.00	23.56
23	--0-0-10----------n-0---0	----------000-----u	0	-7.56	-1.61	13.56
24	--1-1--1----------1----	----------	0	-4.00	0.00	6.00
25	----------0----	----------	0	-1.00	0.00	2.00
26	----------1----	----------	0	-1.00	0.00	1.00
27	----------	----------	0	0.00	0.00	0.00
28	----------	----------	0	0.00	0.00	0.00
29	----------	----------	0	0.00	0.00	0.00
30	----------	----------	0	0.00	0.00	0.00
31	----------	----------	0	0.00	0.00	0.00
32	----------	----------	0	0.00	0.00	0.00
33	----------	----------	0	0.00	0.00	0.00
34	----------	0----------	0	0.00	0.00	0.00
35	----------	----------	0	0.00	0.00	0.00
36	----------	----------	0	0.00	0.00	0.00
37	----------	----------	0	0.00	0.00	0.00
38	----------	----------	0	0.00	0.00	0.00
39	----------	----------	0	0.00	0.00	0.00
40	----------	----------	0	0.00	0.00	0.00
41	----------	0----------	0	0.00	0.00	0.00
42	----------	----------	0	0.00	0.00	0.00
43	----------	----------10	0	0.00	0.00	0.00
44	----------	----------	0	0.00	0.00	0.00

Table 13 – NL-characteristic of 45 steps after Stage 4, work factor $N_{\mathrm{w}} = 2^{51.53}$

t	∇Q_{t+1}	∇W_t	F_{W}	$P_{\mathrm{u}}(t)$	$P_{\mathrm{c}}(t)$	$N_{\mathrm{s}}(t)$
-5	00001111010010111000011111000011					
-4	01000000110010010101000111011000					
-3	01100010111010110111001111111010					
-2	11101111110011011010101110001001					
-1	01100111010001010010001100000001					
0	----------	----------	32	0.00	0.00	0.00
1	----------	----------	32	0.00	0.00	0.00
2	----------	----------	32	0.00	0.00	0.00
3	----------	----------	32	0.00	0.00	0.00
4	----------	----------	32	0.00	0.00	0.00
5	----------	----------	32	0.00	0.00	0.00
6	----------	----------	32	0.00	0.00	0.00
7	----------	----00----------10	28	0.00	0.00	0.00
8	----------	----------	32	0.00	0.00	0.00
9	----------	----------	32	0.00	0.00	0.00
10	----------	----------	0	0.00	0.00	0.00
11	----------0----------	----------	32	-1.00	0.00	0.00
12	----------0-0--------	----------	32	-2.00	0.00	0.00
13	----------1-00-----0--	----------	32	-4.00	0.00	23.48
14	----------0100011011001	----------	0	-13.00	0.00	51.48
15	------00----------0001011011110	----------	0	-17.00	0.00	38.48
16	-1-0--110----------unnnnnnnnnnnn	10----------1-00---0u	25	-17.83	-4.24	21.48
17	01-1--1----------110100u111010	----------0-------	31	-18.00	0.00	28.65
18	00-1-1110011111-----uu1uu0001110	----------01000--1--u	25	-20.00	-1.00	41.65
19	01-unn--nnu1111------1011000nn10	000-00000100000------1-100000000	0	-11.58	-8.59	46.65
20	n-u0uu0010000---------0000000-10	000001011110---1------0101011000	0	-6.10	-4.62	35.07
21	u-u-0n11--110---------11-01---10	10----------1-00---0u	0	-10.03	-1.00	28.96
22	--1-110---011----------0-n----11	----00----------10	0	-7.46	-0.12	18.93
23	--0-0-10----------n-0---0	----------01000--1--u	0	-5.48	0.00	11.48
24	--1-1--1----------1----	----------	0	-4.00	0.00	6.00
25	----------0----	----------	0	-1.00	0.00	2.00
26	----------1----	----------	0	-1.00	0.00	1.00
27	----------	----------	0	0.00	0.00	0.00
28	----------	----------	0	0.00	0.00	0.00
29	----------	----------	0	0.00	0.00	0.00
30	----------	----------	0	0.00	0.00	0.00
31	----------	----------	0	0.00	0.00	0.00
32	----------	----------0-------	0	0.00	0.00	0.00
33	----------	----------	0	0.00	0.00	0.00
34	----------	000-00000100000------1-100000000	0	0.00	0.00	0.00
35	----------	----------	0	0.00	0.00	0.00
36	----------	----------	0	0.00	0.00	0.00
37	----------	----------	0	0.00	0.00	0.00
38	----------	----------	0	0.00	0.00	0.00
39	----------	----------	0	0.00	0.00	0.00
40	----------	----------	0	0.00	0.00	0.00
41	----------	000-00000100000------1-100000000	0	0.00	0.00	0.00
42	----------	----------	0	0.00	0.00	0.00
43	----------	----00----------10	0	0.00	0.00	0.00
44	----------	----------	0	0.00	0.00	0.00

Publication Chapter

Cryptanalysis of the ESSENCE Family of Hash Functions

Publication Data

Nicky Mouha, Gautham Sekar, Jean-Philippe Aumasson, Thomas Peyrin, Søren S. Thomsen, Meltem Sönmez Turan, and Bart Preneel. Cryptanalysis of the ESSENCE Family of Hash Functions. In Feng Bao, Moti Yung, Dongdai Lin, and Jiwu Jing, editors, *Inscrypt*, volume 6151 of *Lecture Notes in Computer Science*, pages 15–34. Springer, 2009.

Contributions

- Main author. Devised the attack on 31 rounds. The distinguishing attacks on 14 rounds are by Gautham Sekar.

Cryptanalysis of the ESSENCE Family of Hash Functions*

Nicky Mouha[1,2,†], Gautham Sekar[1,2,‡], Jean-Philippe Aumasson[3,§], Thomas Peyrin[4], Søren S. Thomsen[5], Meltem Sönmez Turan[6], and Bart Preneel[1,2]

[1] Department of Electrical Engineering ESAT/SCD-COSIC, Katholieke Universiteit Leuven, Kasteelpark Arenberg 10, B-3001 Heverlee, Belgium.
[2] Interdisciplinary Institute for BroadBand Technology (IBBT), Belgium.
[3] FHNW, Windisch, Switzerland.
[4] Ingenico, France.
[5] Department of Mathematics, Technical University of Denmark, Matematiktorvet 303S, DK-2800 Kgs. Lyngby, Denmark.
[6] Computer Security Division, National Institute of Standards and Technology, USA.
{nicky.mouha,gautham.sekar,bart.preneel}@esat.kuleuven.be, jeanphilippe.aumasson@gmail.com, thomas.peyrin@gmail.com, ssth@win.dtu.dk, meltem.turan@nist.gov

Abstract. ESSENCE is a family of cryptographic hash functions, accepted to the first round of NIST's SHA-3 competition. This paper presents the first known attacks on ESSENCE. We present a semi-free-start collision attack on 31 out of 32 rounds of ESSENCE-512, invalidating the design claim that at least 24 rounds of ESSENCE are secure against differential cryptanalysis. We develop a novel technique to satisfy the first nine rounds of the differential characteristic. Non-randomness in the outputs of the feedback function F is used to construct several distinguishers on a 14-round ESSENCE block cipher and the corresponding compression function, each requiring only 2^{17} output bits. This observation is extended to key-recovery attacks on the block cipher. Next, we show that the omission of round constants allows slid pairs and fixed points to be found. These attacks are independent of the number of rounds. Finally, we suggest several countermeasures against these attacks, while still keeping the design simple and easy to analyze.

Keywords: Cryptanalysis, hash function, ESSENCE, semi-free-start collision, distinguisher, key-recovery, slide attack.

*This work was supported in part by the IAP Program P6/26 BCRYPT of the Belgian State (Belgian Science Policy), and in part by the European Commission through the ICT program under contract ICT-2007-216676 ECRYPT II.

†This author is funded by a research grant of the Institute for the Promotion of Innovation through Science and Technology in Flanders (IWT-Vlaanderen).

‡This author is supported by an ADAPID project.

§Supported by the Swiss National Science Foundation under project no. 113329.

1 Introduction

Recent attacks by Wang et al. on the widely used hash functions MD4 [16], MD5 [17], RIPEMD [16] and SHA-1 [18], as well as other hash functions, show that collisions for these hash functions can be found much faster than expected by the birthday paradox [15].

In search for a new secure hash function standard, NIST announced the SHA-3 hash function competition [12]. The ESSENCE family of cryptographic hash functions, designed by Martin [8], advanced to the first round of this competition. It is a family of block cipher-based hash functions using the Merkle-Damgård mode of operation. The ESSENCE family uses simple algorithms that are easily parallelizable and well-established mathematical principles. ESSENCE comes with a proof of security against linear and differential cryptanalysis, that until this paper remained unchallenged.

First, we describe several undesired properties of the ESSENCE L function. These can be used to build a semi-free-start collision attack [11, pp. 371–372] on 31 out of 32 rounds of the ESSENCE-512 compression function using a differential characteristic. This directly invalidates the design claim that at least 24 rounds of ESSENCE are resistant against differential cryptanalysis [8]. To build our attack, we describe a novel technique to satisfy the conditions imposed by the characteristic in the first nine rounds. We do not know of a similar technique in existing literature.

Then, we find that the ESSENCE compression functions use a non-linear feedback function F that is unbalanced. We first exploit this to build efficient distinguishers on 14-round versions of the ESSENCE block ciphers as well as the compression functions. These distinguishers require only 2^{17} output bits. We then show how to use these results to recover the key with a few known plaintexts and a computational effort less than that of exhaustive search. We also show that, under some circumstances, the attacks on 14-round ESSENCE could be extended to the full 32-round block cipher and compression function.

Following this, we observe that the omission of round constants in ESSENCE leads to several attacks that cannot be prevented by increasing the number of rounds. A slide attack can be applied to any number of rounds of the ESSENCE compression function. We also find fixed points for any number of rounds of the ESSENCE block cipher, that lead to a compression function output of zero.

ESSENCE was not advanced to the second round of the SHA-3 competition; however, its appealing features (like design simplicity and hardware efficiency) make any effort on tweaking it appear worthwhile. Therefore, in this paper, we also suggest some countermeasures to thwart the aforesaid attacks.

In later work, Naya-Plasencia et al. [13] present different results on ESSENCE. Our paper presents not only differential cryptanalysis but also distinguishing attacks and slide attacks. Furthermore, some of our techniques can easily be generalized to other block ciphers and hash functions.

The paper is organized as follows. Section 2 describes the compression function

of ESSENCE. In Sect. 3, we define and calculate the branching number of the linear L function for both linear and differential cryptanalysis. As the branching number turns out to be quite low, we use this observation to build a semi-free-start collision attack for 31 out of 32 rounds in Sect. 4. To satisfy the first nine rounds of the differential characteristic of the semi-free-start collision attack, we develop a technique in Sect. 5. Our distinguishers that exploit the weakness of F function are presented in Sect. 6. In the same section, we also show how our distinguishing attacks can be converted into key-recovery attacks on the block ciphers. Following this, we show how the omission of round constants allows us to find slid pairs (Sect. 7) and fixed points (Sect. 8) for any number of rounds. Finally, Sect. 9 enlists our countermeasures and Sect. 10 concludes the paper.

2 Description of the Compression Function of ESSENCE

The inputs to the compression function of ESSENCE are an eight-word chaining value and an eight-word message block, where each word is 32 or 64 bits in length, for ESSENCE-224/256 and ESSENCE-384/512 respectively. The compression function uses a permutation E, that in turn uses a nonlinear feedback function F, a linear transformation L, some XORs and word shifts.

The message block $m = (m_0, \ldots, m_7)$ forms the initial value of an eight-word state $k = (k_0, \ldots, k_7)$. In the case of the block cipher, m is the key $k = (k_0, \ldots, k_7)$. Similarly, the chaining value $c = (c_0, \ldots, c_7)$ is the initial chaining value of an eight-word state $r = (r_0, \ldots, r_7)$. In the case of the block cipher, c is the plaintext. Both states are iterated N times. The designer recommends N to be a multiple of 8, $N \geq 24$ for resistance to differential and linear cryptanalysis and $N = 32$ as a measure of caution [8]. Figure 1 illustrates one round of ESSENCE. The

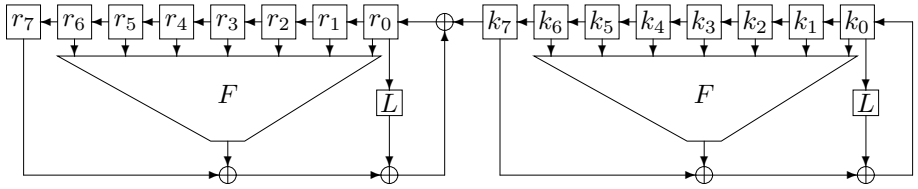


Figure 1 – One round of ESSENCE; each r_n and k_n $(n = 0, \ldots, 7)$ is a 32- or 64-bit word

compression function uses a Davies-Meyer feed-forward (see Fig. 2). That is, at the end of N rounds, the value $r_7||r_6||r_5||r_4||r_3||r_2||r_1||r_0$ is XORed with the initial chaining value. The result is the $r_7||r_6||r_5||r_4||r_3||r_2||r_1||r_0$ for the next iteration.

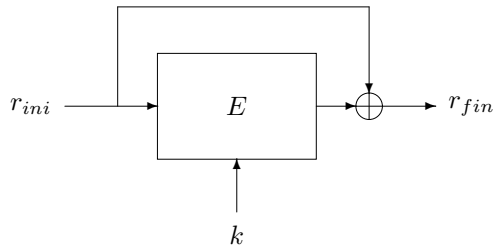


Figure 2 – The compression function of ESSENCE; E is the round function of ESSENCE when iterated N times, k denotes the message block, r_{ini} denotes the initial value of $r_7||r_6||r_5||r_4||r_3||r_2||r_1||r_0$ and r_{fin} denotes the value of r for the next iteration

3 Branching Number of the L Function

The L function of ESSENCE is a linear transformation from 32 (or 64) bits to 32 (or 64) bits and it is the only component that causes diffusion between the different bit positions of a word. Therefore, its properties are very important for both linear and differential cryptanalysis.

In this section, we focus on the branching number of the L function for both linear and differential cryptanalysis. Let the branching number for differential cryptanalysis be the minimum number of non-zero input and output differences for the L function. These branching numbers are 10 and 16 for the 32-bit and 64-bit L functions respectively. If we were to consider only one-bit differences at either the input or the output of L, these numbers would be 14 and 27 respectively.

The branching number for linear cryptanalysis can be defined as the (non-zero) minimum number of terms in a linear equation relating the input and output bits of the L function. These branching numbers are 10 and 17 for the 32-bit and 64-bit L function respectively. Considering linear relations that involve only one bit at the input or one bit at the output, we would find branching numbers of 12 and 26 respectively.

Although one-bit differences are spread out well by the L function, this is clearly not the case for differences in multiple bits. This problem is most severe with the 64-bit L function. In the next section, we will show how this property can be used to build narrow trails for all digest sizes of ESSENCE.

4 A 31-Round Semi-Free-Start Collision Attack For ESSENCE-512

In this section, we will focus only on ESSENCE-512 for the sake of brevity and clarity. As the strategy is not specific to any particular digest size, these results

can easily be generalized to all digest sizes of ESSENCE.

Although the ESSENCE L function spreads out one-bit differences very well, the previous section showed that this is not the case for differences in multiple bits. We therefore propose to use the differential characteristic of Table 1, to obtain 31-round semi-free-start collisions for ESSENCE-512.

To construct narrow trails, we use the non-zero difference A with the lowest possible Hamming weight. For this difference, we impose the condition $(\neg A) \wedge L(A) = 0$, where $\neg$ represents the negation operation and all logical operations are to be performed bitwise. This can be formulated as follows: if there is a difference at the output of the L function at a particular bit position, there must be a difference at the input of L at this bit position as well. This requirement is necessary, as the F function can absorb or propagate an input difference at the output, but if no input difference is present, then there won't be an output difference either at this particular bit position. This places a restriction on the output difference of the L function for this bit position.

There exist exactly 8 differences A with a weight of 17 and lower weight differences A do not exist. These differences are available in Appendix A, along with a method to calculate them efficiently.

The last two columns of Table 1 provide an estimate of the probability that the characteristic is satisfied for every round. For these, we have assumed that the F function propagates or absorbs an input difference with equal probability. A more accurate calculation of these probabilities takes into account that the shift register causes input values of the F function to be reused.

We find that this probability is different for bit positions where A and $L(A)$ both contain a difference, and for bit positions where only A contains a difference. As such, of all differences A with weight 17, we select the difference that has the highest weight of $L(A)$. Five such differences exist, and we arbitrarily select the difference with the smallest absolute value, $A =$ `0A001021903036C3`. The corresponding $L(A) =$ `0200100180301283` has weight 11. As such, we find that rounds 10 to 16 of the key schedule, and rounds 18 to 24 of the compression function, each have a probability of $2^{-8.415 \cdot 6 - 8 \cdot 11} = 2^{-138.49}$. For rounds 18 to 23 of the key schedule, we find a probability of $2^{-7.193 \cdot 6 - 7 \cdot 11} = 2^{-120.16}$.

To find semi-free-start collisions, we first search for a message pair that satisfies the key expansion characteristic, and then afterwards search for a chaining value pair that satisfies the compression function characteristic. These two searches can be decoupled, as the chaining value does not influence the key schedule. As such, the probabilities for the message pairs and IV pairs can be summed up instead of multiplied.

As will be shown in the next section, only two round function calls are required to find a message (or IV) that satisfies the first nine rounds of the key expansion (or compression function). To find a pair of messages (or IVs) that satisfy the differential characteristic, we use the same depth-first search algorithm that was introduced for SHA-1 in [2]. The memory requirements of this search algorithm are negligible. We assume that the cost of visiting a node in this search tree is equiva-

lent to one round function call, or 2^{-5} compression function calls. The complexity calculation of [2] then shows that a 31-round semi-free-start collision can be found using the characteristic of Table 1 after $2^{138.49+120.16+1-5} + 2^{138.49+1-5} = 2^{254.65}$ equivalent compression function calls. This is faster than a generic birthday attack, which requires about 2^{256} compression function evaluations.

5 Finding Message Pairs for the First Nine Rounds

To find messages that satisfy the first few rounds of the characteristic, single-message modification [17] cannot be used. This is because the entire message is loaded into the r-registers before the round function is applied, instead of injecting one message word every round. We therefore propose to use another technique, that turns out to be even more efficient than single-message modification. This concept is explained for the key schedule only, as it is completely analogous for the compression function.

In this section, we will adopt a stream-based notation for the round function. Denote the initial eight-word state $(k_7, k_6, k_5, k_4, k_3, k_2, k_1, k_0)$ as $(x_{-2}, x_{-1}, x_0, x_1, x_2, x_3, x_4, x_5)$. After clocking one for one round, the value of the register k_0 is represented by x_6, and so on. In this text, we will not make a distinction between linear and affine equations, and use the term "linear equation" for any equation that contains no monomials of a degree more than one.

Finding a pair of messages that satisfy the characteristic, can be seen as solving a set of non-linear equations defined by the round function. Solving a set of non-linear equations is a difficult problem in general. This is even more the case as we are not looking for a single solution, but for a very large set of solutions.

What we can do, however, is impose linear conditions on the variables x_0 to x_{12}, in such a way that the round function behaves as a linear function. We then obtain a set of linear equations, of which every solution corresponds to a message pair that follows the first nine rounds of the characteristic. Enumerating the solutions of this linear space has a negligible computation cost compared to a round function evaluation.

For every solution, we have to apply the round function twice to obtain x_{13} and x_{14}. These are guaranteed to follow the characteristic as well. They serve as a starting point to satisfy the conditions of the remaining characteristic in a probabilistic way. After reaching round 31, we can calculate x_{-2} and x_{-1} by applying two inverse round functions. These values will always satisfy the characteristic.

Let $A[j]$ denote the j-th significant bit ($j = 0$ denotes the least significant bit or LSB) of A. The only non-linear function of ESSENCE is the F function. As the F function operates on every bit in parallel, the linear conditions that have to be added, depend on the values $A[j]$ and $L(A)[j]$ at every bit position j. The equations we use are given in Appendix B. Note that for bit positions j where $A[j] = 0$, it is not a problem if $x_0[j]$ or $x_{12}[j]$ are represented by a non-linear expression, as these bits are not involved in any of the linear conditions anyway.

Table 1 – A 31-round semi-free-start collision differential characteristic for the ESSENCE-512 compression function; differences from R to Y are arbitrary, 0 represents the zero difference, $A =$ `0A001021903036C3`

Round	Register R	Register K	Pr for CV	Pr for m
0	0 0 0 0 0 0 0 0	A 0 0 0 0 0 0 0	1	1
1	0 0 0 0 0 0 0 A	0 0 0 0 0 0 0 A	1	1
2	0 0 0 0 0 0 A 0	0 0 0 0 0 0 A 0	2^{-17}	2^{-17}
3	0 0 0 0 0 A 0 0	0 0 0 0 0 A 0 0	2^{-17}	2^{-17}
4	0 0 0 0 A 0 0 0	0 0 0 0 A 0 0 0	2^{-17}	2^{-17}
5	0 0 0 A 0 0 0 0	0 0 0 A 0 0 0 0	2^{-17}	2^{-17}
6	0 0 A 0 0 0 0 0	0 0 A 0 0 0 0 0	2^{-17}	2^{-17}
7	0 A 0 0 0 0 0 0	0 A 0 0 0 0 0 0	2^{-17}	2^{-17}
8	A 0 0 0 0 0 0 0	A 0 0 0 0 0 0 0	2^{-17}	2^{-17}
9	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 A	1	1
10	0 0 0 0 0 0 0 0	0 0 0 0 0 0 A 0	1	2^{-17}
11	0 0 0 0 0 0 0 0	0 0 0 0 0 A 0 0	1	2^{-17}
12	0 0 0 0 0 0 0 0	0 0 0 0 A 0 0 0	1	2^{-17}
13	0 0 0 0 0 0 0 0	0 0 0 A 0 0 0 0	1	2^{-17}
14	0 0 0 0 0 0 0 0	0 0 A 0 0 0 0 0	1	2^{-17}
15	0 0 0 0 0 0 0 0	0 A 0 0 0 0 0 0	1	2^{-17}
16	0 0 0 0 0 0 0 0	A 0 0 0 0 0 0 0	1	2^{-17}
17	0 0 0 0 0 0 0 A	0 0 0 0 0 0 0 A	1	1
18	0 0 0 0 0 0 A 0	0 0 0 0 0 0 A 0	2^{-17}	2^{-17}
19	0 0 0 0 0 A 0 0	0 0 0 0 0 A 0 0	2^{-17}	2^{-17}
20	0 0 0 0 A 0 0 0	0 0 0 0 A 0 0 0	2^{-17}	2^{-17}
21	0 0 0 A 0 0 0 0	0 0 0 A 0 0 0 0	2^{-17}	2^{-17}
22	0 0 A 0 0 0 0 0	0 0 A 0 0 0 0 0	2^{-17}	2^{-17}
23	0 A 0 0 0 0 0 0	0 A 0 0 0 0 0 0	2^{-17}	2^{-17}
24	A 0 0 0 0 0 0 0	A 0 0 0 0 0 0 R	2^{-17}	1
25	0 0 0 0 0 0 0 0	0 0 0 0 0 0 R S	1	1
26	0 0 0 0 0 0 0 0	0 0 0 0 0 R S T	1	1
27	0 0 0 0 0 0 0 0	0 0 0 0 R S T U	1	1
28	0 0 0 0 0 0 0 0	0 0 0 R S T U V	1	1
29	0 0 0 0 0 0 0 0	0 0 R S T U V W	1	1
30	0 0 0 0 0 0 0 0	0 R S T U V W X	1	1
31	0 0 0 0 0 0 0 0	R S T U V W X Y	1	1

As the equations in Appendix B show, we need to add 10 equations for every bit position j where $A[j] = 1$, and 6 equations if $A[j] = 0$. Also, to represent the 64-bit values x_8 to x_{12} resulting from the round function, we need to add $5 \cdot 64$ additional equations for outputs of the round function. In total, we obtain a set of $10 \cdot 17 + 6 \cdot (64 - 17) + 5 \cdot 64 = 772$ linear equations in $13 \cdot 64 = 832$ binary variables.

We build this system of equations by successively adding $10+5 = 15$ or $6+5 = 11$ more equations for every bit position j. With some small probability, the system of equations becomes inconsistent. If this happens, we add a different set of linear equations for this bit position. Even this may fail with some probability, in which case we add a linearization of the F function using $7+5$ instead of $6+5$ equations for this particular bit position. This may or may not decrease the number of solutions slightly, but it allows us to avoid backtracking.

For one particular run, using only the equations mentioned in Appendix C, we find a consistent system of 772 linear equations in 832 binary variables. The number of linearly independent equations turns out to be 771. As such, we have found $2^{832-771}/2 = 2^{60}$ pairs of messages that satisfy the first 9 rounds. We divide by two to avoid counting the same pair twice. If more than 2^{60} pairs of messages are needed, we can simply run this program again to find the next set of messages. As including these 771 equations would use up a lot of space, we give only one of the 2^{60} message pairs in Table 6.

This technique is very similar to the techniques of multi-message modification [17], tunneling [7], neutral bits [1] and the amplified boomerang attack [6]. These 2^{60} messages correspond to 60 auxiliary differential paths for the amplified boomerang attack. No results are known to us where these auxiliary differential paths were also obtained in a fully automated way.

6 Distinguishing Attacks

Our motivational observation is that the non-linear feedback function F is unbalanced. Exploiting this, we first construct distinguishers on 14-round ESSENCE (both the block cipher and the compression function) and then for the full 32-round ESSENCE. Towards the end of this section, we present key-recovery attacks on the ESSENCE family of block ciphers. These attacks can be seen as an immediate consequence of our distinguishing attacks.

6.1 Weakness in the Feedback Function of ESSENCE

In [8], the designer notes that the security of the algorithms is heavily dependent on F, as it is the only nonlinear function in ESSENCE. This gave us some motivation to study the properties of F. The function F takes seven 32-bit or 64-bit words (say, $a, \ldots, g$) as inputs and produces a 32-bit or 64-bit word as the output. The function

works in a bitsliced manner. The exact description of F is largely irrelevant to our analysis; hence, we refer the interested reader to Appendix D.

Let $F(a, b, c, d, e, f, g)[j]$ denote the j-th significant bit ($j = 0$ denotes the LSB) of $F(a, b, c, d, e, f, g)$. Our motivational observation is the following (confirmed both experimentally and from the tables in Appendix D of [8]).

Observation 1: If $a, \ldots, g$ are uniformly distributed, then

$$Pr[F(a, b, c, d, e, f, g)[j] = 0] = \frac{1}{2} + \frac{1}{2^7} \ . \tag{1}$$

6.2 Distinguishers on 14-Round ESSENCE

In this section, we use Observation 1 to build distinguishers on 14 rounds of ESSENCE. First, we consider the block cipher, then the compression function.

Let $k_n[j]$, $r_n[j]$ and $L(r_n)[j]$ respectively denote the j-th significant bits ($j = 0$ denotes the LSB) of k_n, r_n and $L(r_n)$. In the beginning, suppose the key k and the initial value r are such that $k_0[0] = r_0[0]$. Then, after 7 rounds, $k_7[0] = r_7[0]$. Now, if after the 7th round, $L(r_0)[0] = 0$ and $F(r_6, r_5, r_4, r_3, r_2, r_1, r_0)[0] = 0$ (from Observation 1, this occurs with $0.5 + 2^{-7}$ probability[7]), then after the 8th round, we will have $r_0[0] = 0$. Note that the condition $L(r_0)[0] = 0$ after the 7th round is the same as the condition $L(r_1)[0] = 0$ after the 8th round. Therefore, when the key and the plaintext are initially related in the form $k_0[0] = r_0[0]$, and when the outputs after 8 rounds satisfy the condition $L(r_1)[0] = 0$ (this occurs with probability 1/2), then $Pr[r_0[0] = 0] = 1/2 + 2^{-7}$. Now, r_0 and r_1 after the 8th round are respectively equal to r_6 and r_7 after the 14th round. Hence, when the key and the plaintext are related in the form $k_0[0] = r_0[0]$, and when the outputs after 14 rounds satisfy the condition $L(r_7)[0] = 0$, then

$$Pr[r_6[0] = 0] = \frac{1}{2} + \frac{1}{2^7} \ . \tag{2}$$

6.3 The Distinguisher

A distinguisher is an algorithm that distinguishes one probability distribution from another. In cryptography, a distinguisher is an algorithm that distinguishes a stream of bits from a stream of bits uniformly distributed at random (i.e., bitstream generated by an ideal cipher).

Our distinguisher on ESSENCE is constructed by collecting n outputs $r_6[0]$, after 14 rounds, generated by as many keys (so that the n samples are independent) such that $k_0[0] = r_0[0]$ initially. Let D_0 and D_1 denote the distributions

[7]The bit $L(r_0)[0]$ is the XOR-sum of $r_0[0]$ and several other bits of r_0. We assume that all $r_0[j]$ are independent and uniformly distributed. Then the condition $L(r_0)[0] = 0$ does not affect $Pr[r_0[0] = 0]$ and therefore the bias in $Pr[F(r_6, r_5, r_4, r_3, r_2, r_1, r_0)[0] = 0]$ is also unaffected.

of the outputs from 14-round ESSENCE block cipher and a random permutation, respectively. Given $L(r_7)[0] = 0$, let p_0 and p_1 respectively denote the probability that $r_6[0] = 0$ holds given the outputs are collected from 14-round ESSENCE and the probability that $r_6[0] = 0$ holds given the outputs are generated by a random source. That is, $p_0 = 1/2 + 2^{-7}$ (from (2)) and $p_1 = 1/2$. Then, $\mu_0 = n \cdot p_0$ and $\mu_1 = n \cdot p_1$ are the respective means of D_0 and D_1. Similarly, $\sigma_0 = \sqrt{n \cdot p_0 \cdot (1 - p_0)}$ and $\sigma_1 = \sqrt{n \cdot p_1 \cdot (1 - p_1)}$ denote the respective standard deviations of D_0 and D_1. When n is large, both these binomial distributions can be approximated with the normal distribution. Now, if $|\mu_0 - \mu_1| > 2(\sigma_0 + \sigma_1)$, i.e., $n > 2^{16}$, the output of the cipher can be distinguished from a random permutation with a success probability of 0.9772 (since the cumulative distribution function of the normal distribution gives the value 0.9772 at $\mu + 2\sigma$) provided $L(r_7)[0] = 0$. To test whether n is large enough for the normal approximation to the binomial distribution to hold, we use a commonly employed rule of thumb: $n \cdot p > 5$ and $n \cdot (1 - p) > 5$, where $p \in \{p_0, p_1\}$. A simple calculation proves that both the above inequalities hold when $n = 2^{16}$. Since the condition $L(r_7)[0] = 0$ holds with 0.5 probability, we need to generate $2 \cdot 2^{16} = 2^{17}$ samples of $r_6[0]$ from as many keys (such that $k_0[0] = r_0[0]$ initially) to build the distinguisher with a success probability of 0.9772. Our simulations support this result.

6.4 Distinguishers using Biases in Other Bits

Since the function F operates on its input bits in a bitsliced manner, it is easy to see that the distinguisher presented for the LSB of r_6 also works for more significant bits. In other words, if initially $k_0[j] = r_0[j]$, for any j in $\{0, \ldots, 31\}$, then with 2^{16} samples of $r_6[j]$ at the the end of 14 rounds, it is possible to distinguish 14-round ESSENCE block cipher from a random permutation with a success probability of 0.9772.

6.5 Distinguishers for the Compression Function

The ESSENCE compression function is a Davies-Meyer construction in which the output of the block cipher is XORed with the initial chaining value. In other words, the output of the compression function is the XOR-sum of the values of $r_7||r_6||r_5||r_4||r_3||r_2||r_1||r_0$ before and after applying the permutation E. This XOR-sum is the chaining value $r_7||r_6||r_5||r_4||r_3||\ r_2||r_1||r_0$ for the next iteration. As we assume that an attacker can observe both the chaining value input and the compression function output, it is trivial to undo the Davies-Meyer feedforward and apply the distinguishers of the 14-round block cipher.

These observations are extended to 32-round ESSENCE in Appendix E.

6.6 Key-Recovery Attacks

In this section, we show that the distinguishing attacks on the ESSENCE family of block ciphers can be converted into key-recovery attacks.

Let us say that we have n known plaintexts. Considering that the plaintexts are initially loaded directly into the r-registers [9], we expect $n/2$ plaintexts to have $r_0[j] = 0$. Without loss of generality, let us consider this partition of the plaintext space where $r_0[j] = 0$. Now, from our analysis in Sect. 6.2, we can collect statistics on $L(r_7)[j] \oplus r_6[j]$ at the end of the 14 rounds and observe its tendency for sufficiently large n — if $L(r_7)[j] \oplus r_6[j] = 0$ more often, then the key bit $k_0[j] = 0$; likewise, if $L(r_7)[j] \oplus r_6[j] = 1$ more often, then the key bit $k_0[j] = 1$ (the results are swapped if we begin with plaintexts in which $r_0[j] = 1$).

Using a similar analysis, we can recover the rest of the key bits in k_0. The number of known plaintexts required is 2^{15}. This is obtained as follows, using standard linear cryptanalysis [10]. We are interested in finding whether, after 14 rounds, the number of times that $L(r_7)[j] \oplus r_6[j] = 0$ holds is greater than $n/4$. Accordingly, we determine the key bit $k_0[j]$. Unlike in the distinguishing attacks, a confidence interval for the uniform distribution is not required. From [10] we obtain that the success probability of this method is 0.9772 when $n/2 = |p - 1/2|^{-2}$, where p is the probability that $L(r_7)[j] \oplus r_6[j] = 0$ (or 1). Substituting $p = 1/2 \pm 2^{-7}$ in the above formula for n, we get $n = 2^{15}$. It follows that the probability that this recovered key word (k_0) is correct is $(0.9772)^{32} \approx 0.48$. The other 224 bits of the key can be exhaustively searched. Thereby, we expect that $2^{224}/0.48 \approx 2^{225.1}$ keys have to be tested before the correct key is obtained with guaranteed success. This key-recovery attack can also be applied on the block cipher of ESSENCE-224 (which is identical to the block cipher of ESSENCE-256) with the same complexities. For the block ciphers of ESSENCE-384/512, we require 2^{15} known plaintexts and a computational effort equivalent to testing $2^{448}/(0.9772)^{64} \approx 2^{450.1}$ keys (where exhaustive search requires testing 2^{512} keys) for guaranteed success.

These observations are extended to 32-round ESSENCE in Appendix F.

7 Slide Attack

In this part of the study, we provide an efficient method to find two inputs (c, m) and (c', m') such that their output (after feed-forward) r and r' are shifted versions of each other; i.e., if $r_i = r'_{i+1}$ for $0 \le i < 7$.

The necessary conditions on (c, m) and (c', m') are

1. $c_i = c'_{i+1}$ for $0 \le i \le 7$,
2. $c'_0 = m_7 \oplus c_7 \oplus F(c_6, \ldots, c_0) \oplus L(c_0)$,
3. $m_i = m'_{i+1}$ for $0 \le i \le 7$,

4. $m'_0 = m_7 \oplus F(m_6, \ldots, m_0) \oplus L(m_0)$.

If these conditions hold, then after 32 rounds (and XORing with the initial value), the output of the compression function satisfies $r_i = r'_{i+1}$ for $0 \leq i < 7$.

As an example, let $m_i = 0$ for all i. Then we must choose $m'_i = 0$ for all $i > 0$, and $m'_0 = 1^n$ where 1^n represents the 32-bit or 64-bit unsigned integer of which all bits are set. Let $c_i = 0$ for all i, let $c'_i = 0$ for all $i > 0$, and let $c'_0 = 1^n$. Then, the two outputs of the compression function (with $N = 32$) are:

c	00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
c'	FFFFFFFF 00000000 00000000 00000000 00000000 00000000 00000000 00000000
m	00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
m'	FFFFFFFF 00000000 00000000 00000000 00000000 00000000 00000000 00000000
R	6B202EF2 BB610A07 97E43146 9BD34AE3 C8BC7CBF B8EE4A3C B6118DC5 775F7BBF
R'	C07ABCFA 6B202EF2 BB610A07 97E43146 9BD34AE3 C8BC7CBF B8EE4A3C B6118DC5

For every choice of (c, m), an input (c', m') such that this property on the compression function outputs is obtained can be found in time equivalent to about one compression function evaluation. Hence, in total about 2^{512} pairs of inputs producing slid pairs can be found by the above method. This observation can easily be extended to slide the output by $2, 3, \ldots, 7$ steps.

7.1 Slid Pairs with Identical Chaining Values

It is also possible to find slid pairs with $c = c'$. Let the initial state of the register R be of the form $(c_0, c_0, \ldots, c_0)$, where c_0 is selected randomly. For a message block m of the form $(m_0, m_1, \ldots, m_7)$ where $m_7 = F(c_0, \ldots, c_0) \oplus L(c_0)$ and the rest of the m_i's are selected arbitrarily, select m' as $(m'_0, m'_1, \ldots, m'_7)$, such that $m'_{i+1} = m_i$ for $i = 0, 1, 2, \ldots, 6$ and $m'_0 = m_7 \oplus F(m_6, \ldots, m_0) \oplus L(m_0)$. Then, the outputs of the compression function for m and m' also satisfy $r_i = r'_{i+1}$ for $0 \leq i < 7$. It is possible to select c in 2^{32} different ways, and for each selected c, we can choose $2^{7 \cdot 32}$ different message blocks, therefore the number of such slid pairs is 2^{256}. As an example, assume $c_0 =$ `243f6a88`, which is the truncated fractional part of π, and all "free" message words are zero.

c, c'	243F6A88 243F6A88 243F6A88 243F6A88 243F6A88 243F6A88 243F6A88 243F6A88
m	00000000 00000000 00000000 00000000 00000000 00000000 00000000 F6B1EB63
m'	094E149C 00000000 00000000 00000000 00000000 00000000 00000000 00000000
R	BE31AA01 EB6E9F07 EAD99889 6FE79B44 391CCD35 67FDB8B6 FC3AA0F6 6E80148E
R'	F86D77C6 BE31AA01 EB6E9F07 EAD99889 6FE79B44 391CCD35 67FDB8B6 FC3AA0F6

8 Fixed Points for the ESSENCE Block Cipher

If a fixed point for one round of the ESSENCE block cipher can be found, this automatically leads to a fixed point for all 32 steps of the block cipher. After

applying the Davies-Meyer feed-forward, the resulting compression function output will then be zero.

If two different fixed points are found, this would lead to a free-start collision. This free-start collision is preserved after the output padding is applied.

For a fixed point for one round, $c_0 = c_1 = \ldots = c_7$ and $m_0 = m_1 = \ldots = m_7$ should hold. This is obvious: after one step, all register values move one place, but must have the same value as in the previous step to form a fixed point. Moreover, the round update functions should satisfy the following equations.

$$F(c_0, c_0, c_0, c_0, c_0, c_0, c_0) \oplus c_0 \oplus L(c_0) \oplus m_0 = c_0 \ ,$$
$$F(m_0, m_0, m_0, m_0, m_0, m_0, m_0) \oplus m_0 \oplus L(m_0) = m_0 \ .$$

Solving the equations, one gets the following values for ESSENCE-256 and ESSENCE-512:

	ESSENCE-256	ESSENCE-512
c_0	993AE9B9	D5B330380561ECF7
m_0	307A380C	10AD290AFFB19779

Using similar methods, we have found that the only fixed points for two, three or four rounds is the same fixed point for one round applied two, three or four times respectively. We have not been able to extend this result for more rounds. As such, we have not been able to find a free-start collisions using this technique. Depending how the compression function is used, however, it might be undesirable that we can easily find inputs that fix the compression function output to zero.

9 Measures to Improve the Security of ESSENCE

From the analysis in Sect. 3–6, it is clear that ESSENCE has weaknesses in L and F.

The concatenation of both the input and output of the L function can be seen as an error-correcting code with $[n, k] = [64, 32]$ or $[128, 64]$. The branching number is then equal to the error-correcting code of these dimensions with the highest minimum weight. Best known results from coding theory [5] can be used to construct an L function with a branching number for both linear and differential cryptanalysis of 12 or 22 respectively. Better codes may exist according to currently known upper bounds for the minimum weight, but have so far not been found.

A search can be made for variants of these codes (possibly with a slightly lower branching number) that satisfy all design criteria for the L function. Although the resulting function will always be linear, it may however not be possible to implement it as an LFSR.

In (5), the function F is in algebraic normal form (ANF). We know that the coefficient of the maximum degree monomial in this ANF is equal to the XOR-sum

of all the entries in the truth table of F. To thwart the attacks in Sect. 6 and Appendix F, it is necessary that F is balanced. Discarding the maximum degree monomial is a possible solution.

Other countermeasures include increasing the number of rounds and adding round constants. In Sect. 7 and Sect. 8, we saw how the omission of round constants allowed slid pairs and fixed points to be found. Increasing the number of rounds does not thwart these attacks, but it increases the security margin against the semi-free-start collision attacks of this paper.

10 Conclusions and Open Problems

In this paper, we first presented a semi-free-start collision attack on 31 out of 32 rounds with a complexity of $2^{254.65}$ compression function evaluations. We find messages that satisfy the first nine rounds of the differential characteristic of the semi-free-start collision attack as the solution of a large set of linear equations. We found that six linear input conditions are sufficient to make F behave as a linear function in Table 5. It is an open problem if solutions using fewer equations exist.

We also presented a set of distinguishers on 14-round ESSENCE. The distinguishers can be applied to the block cipher as well as the compression function. Each of the distinguishers on 14-round ESSENCE requires 2^{17} output bits. The distinguishers work on all digest sizes of ESSENCE with the same complexity. It has also been shown how the distinguishing attacks can be turned into key-recovery attacks.

We then showed how the omission of round constants allowed slid pairs and fixed points to be found. This cannot be prevented by increasing the number of rounds.

Finally, we suggested some measures to improve the security of ESSENCE. These suggestions are rather preliminary and need to be worked on further in order to obtain a more secure family of hash functions.

11 Acknowledgments

The authors would like to thank Christophe De Cannière, Sebastiaan Indesteege, Gaëtan Leurent, Willi Meier, Tomislav Nad, María Naya-Plasencia, Vincent Rijmen and Andrea Röck for their useful comments and suggestions.

Special thanks go out to the designer of ESSENCE, Jason Worth Martin, who not only gave us useful feedback, but was also very supportive when we wanted to make these results public. He has verified the correctness of the results in this paper.

Part of this work was performed at the Hash Function Retreat, hosted by the Graz University of Technology as an initiative of the SymLab group of the ECRYPT II project. We are very grateful to Mario Lamberger, Florian Mendel,

Tomislav Nad, Christian Rechberger, Vincent Rijmen and Martin Schläffer for their excellent organization of this retreat.

María Naya-Plasencia, Andrea Röck, Thomas Peyrin, Jean-Philippe Aumasson, Gaëtan Leurent and Willi Meier have obtained other, non-overlapping results on ESSENCE [13] in parallel with these results. Their paper uses different characteristics and another way of finding conforming messages.

References

[1] E. Biham and R. Chen. Near-Collisions of SHA-0. In M. K. Franklin, editor, *CRYPTO*, volume 3152 of *Lecture Notes in Computer Science*, pages 290–305. Springer, 2004.

[2] C. De Cannière and C. Rechberger. Finding SHA-1 Characteristics: General Results and Applications. In X. Lai and K. Chen, editors, *ASIACRYPT*, volume 4284 of *Lecture Notes in Computer Science*, pages 1–20. Springer, 2006.

[3] I. Dinur and A. Shamir. Side Channel Cube Attacks on Block Ciphers. Cryptology ePrint Archive, Report 2009/127, 2009. `http://eprint.iacr.org/`.

[4] S. Dziembowski and K. Pietrzak. Leakage-Resilient Cryptography. In *FOCS*, pages 293–302. IEEE Computer Society, 2008.

[5] M. Grassl. Tables of Linear Codes and Quantum Codes. `http://www.codetables.de/`, June 2008.

[6] A. Joux and T. Peyrin. Hash Functions and the (Amplified) Boomerang Attack. In A. Menezes, editor, *CRYPTO*, volume 4622 of *Lecture Notes in Computer Science*, pages 244–263. Springer, 2007.

[7] V. Klima. Tunnels in Hash Functions: MD5 Collisions Within a Minute. Cryptology ePrint Archive, Report 2006/105, 2006.

[8] J. W. Martin. ESSENCE: A Family of Cryptographic Hashing Algorithms. `http://www.math.jmu.edu/~martin/essence/Supporting_Documentation/essence_compression.pdf`, 2008.

[9] J. W. Martin. Personal communication, 2009.

[10] M. Matsui. Linear Cryptoanalysis Method for DES Cipher. In *EUROCRYPT*, pages 386–397, 1993.

[11] A. Menezes, P. C. van Oorschot, and S. A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1997.

[12] National Institute of Standards and Technology. Announcing Request for Candidate Algorithm Nominations for a New Cryptographic Hash Algorithm (SHA-3) Family. *Federal Register*, 27(212):62212–62220, November 2007. `http://csrc.nist.gov/groups/ST/hash/documents/FR_Notice_Nov07.pdf`.

[13] M. Naya-Plasencia, A. Roeck, T. Peyrin, J.-P. Aumasson, G. Leurent, and W. Meier. Cryptanalysis of ESSENCE. Unpublished, 2009.

[14] K. Pietrzak. A Leakage-Resilient Mode of Operation. In A. Joux, editor, *EUROCRYPT*, volume 5479 of *Lecture Notes in Computer Science*, pages 462–482. Springer, 2009.

[15] B. Preneel. *Analysis and design of cryptographic hash functions*. PhD thesis, Katholieke Universiteit Leuven, 1993.

[16] X. Wang, X. Lai, D. Feng, H. Chen, and X. Yu. Cryptanalysis of the Hash Functions MD4 and RIPEMD. In R. Cramer, editor, *EUROCRYPT*, volume 3494 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2005.

[17] X. Wang and H. Yu. How to Break MD5 and Other Hash Functions. In R. Cramer, editor, *EUROCRYPT*, volume 3494 of *Lecture Notes in Computer Science*, pages 19–35. Springer, 2005.

[18] X. Wang, H. Yu, and Y. L. Yin. Efficient Collision Search Attacks on SHA-0. In V. Shoup, editor, *CRYPTO*, volume 3621 of *Lecture Notes in Computer Science*, pages 1–16. Springer, 2005.

A Finding the Lowest Weight Difference A

We wish to find a difference A that satisfies

$$(\neg A) \wedge L(A) = 0 \tag{3}$$

and

$$\mathrm{hw}(A) \leq w \ , \tag{4}$$

where $\mathrm{hw}(A)$ is the number of bits set in A and w is to be as small as possible.

We proceed as follows. Let w represent the (still unknown) weight of the lowest weight difference A. We then split w into two integers w_0 and w_1, such that $w_0 + w_1 = w$ and $|w_1 - w_0| \leq 1$. Let L^{-1} represent the inverse L function, such that $L^{-1}(L(x)) = x$. Let $M(x) = L(x) \oplus x$. The design of ESSENCE guarantees that M is invertible, as L is not allowed to have any eigenvalues in the ground field.

First step: We enumerate all x where $\mathrm{hw}(x) \leq w_0$. After calculating $A = L^{-1}(x)$, we check (3) and (4).

Table 2 – All differences A with $\mathrm{hw}(A) = 17$ that satisfy (3); there are no solutions where $\mathrm{hw}(A) < 17$ and (3)

A
2461822430680025
48C3044860D0004A
91860890C1A00094
0A001021903036C3
1400204320606D86
2800408640C0DB0C
5000810C8181B618
A001021903036C30

Second step: We enumerate all y where $\mathrm{hw}(y) \leq w_1$. After calculating $A = M^{-1}(y)$, we check if (3) and (4).

Equation (3) implies that the bit positions where $L(A)$ is 1, is always a subset of bit positions where A is 1. Therefore, we only have to consider two cases: the case where the set of bit positions where $L(A)$ is 1 contains no more than w_0 elements, and the case where the set bit positions where $L(A)$ is 0 and A is 1 contains not more than w_1 elements. As $w_0 + w_1 = w$, these two steps are guaranteed to find all A that satisfy (3) and (4). If no solution is found, we increase w by one and perform the two steps again, enumerating only the new values of x and y.

The total complexity of this search is $\left(\sum_{i=0}^{w_0} C_i^{64}\right) + \left(\sum_{j=0}^{w_1} C_j^{64}\right)$. As we find $w = 17$ here, the total number of 64-bit linear function evaluations is $\left(\sum_{i=0}^{8} C_i^{64}\right) + \left(\sum_{j=0}^{9} C_j^{64}\right) \approx 2^{35}$. This calculation can be performed in less than a minute on a recent desktop computer. The solutions are shown in Table 2.

B Making F Behave as a Linear Transformation

We consider three separate cases, depending on the values of A and $L(A)$ for a particular bit position j.

If $A[j] = 1$, we can enumerate all possible input conditions, such that F behaves linearly and has the required differential behavior. Because we enumerate all possibilities, we obtain an optimal result: it is not possible to add fewer than 10 linear equations. All existing solutions where 10 linear equations are added, are shown in Table 3 (for $L(A)[j] = 1$) and Table 4 (for $L(A)[j] = 0$).

If $A[j] = 0$: the differential behavior is always satisfied: if there is no input difference, there will not be an output difference either. We found that adding 6 equations is sufficient. We do not rule out the possibility that fewer than 6 equations are sufficient. The solutions we found are given in Table 4.

We will omit the index j, so that x_0 to x_{12} represent one-bit variables. The

Table 3 – Making F linear and imposing the required differential behavior for position j where $A[j] = L(A)[j] = 1$ can be done by adding no more than 10 linear equations; exactly four such solutions exist

	Solution 1	Solution 2	Solution 3	Solution 4
	$x_0 \oplus x_2 = 1$	$x_1 = 1$	$x_1 = 1$	$x_1 = 1$
	$x_1 = 0$	$x_2 \oplus x_5 = 0$	$x_2 \oplus x_5 = 0$	$x_2 = 1$
	$x_3 = 1$	$x_2 \oplus x_7 = 1$	$x_2 \oplus x_7 = 1$	$x_3 = 0$
	$x_4 = 1$	$x_2 \oplus x_8 = 0$	$x_2 \oplus x_8 = 0$	$x_4 = 1$
	$x_5 = 1$	$x_2 \oplus x_9 = 0$	$x_2 \oplus x_9 = 0$	$x_5 = 1$
	$x_7 = 0$	$x_2 \oplus x_{12} = 1$	$x_3 = 0$	$x_7 = 0$
	$x_8 = 1$	$x_3 = 0$	$x_4 = 1$	$x_8 = 1$
	$x_9 = 0$	$x_4 = 1$	$x_{10} = 0$	$x_9 = 1$
	$x_{10} = 0$	$x_{10} = 0$	$x_{11} = 0$	$x_{10} = 0$
	$x_{12} = 1$	$x_{11} = 0$	$x_{12} = 1$	$x_{11} = 0$
$F(x_0, \ldots, x_6) =$	$x_6 \oplus 1$	$x_0 \oplus x_6$	$x_0 \oplus x_6$	$x_0 \oplus x_6$
$F(x_1, \ldots, x_7) =$	$x_2 \oplus 1$	$x_2 \oplus 1$	$x_2 \oplus 1$	0
$F(x_2, \ldots, x_8) =$	0	$x_2 \oplus 1$	$x_2 \oplus 1$	0
$F(x_3, \ldots, x_9) =$	0	x_5	x_5	1
$F(x_4, \ldots, x_{10}) =$	1	1	1	1
$F(x_5, \ldots, x_{11}) =$	1	0	0	0
$F(x_6, \ldots, x_{12}) =$	0	$x_7 \oplus 1$	0	$x_{12} \oplus 1$

expressions $F(x_0, \ldots, x_6)$ and $F(x_6, \ldots, x_{12})$ are not added to the system of linear equations of the attack, as this is not necessary. They are only mentioned to show that their differential behavior is correct.

C A Message Pair for the First Nine Rounds

We give a message pair that satisfies the first 9 rounds of the characteristic of Table 1 in Table 6.

D The Feedback Function F

We denote the field of two elements by $\mathbb{F}_2$. The nonlinear feedback function, F, of ESSENCE-224/256 (respectively ESSENCE-384/512) takes seven 32-bit (respectively 64-bit) input words and outputs a single 32-bit (respectively 64-bit) word

Table 4 – Making F linear and imposing the required differential behavior for position j where $A[j] = 1$ and $L(A)[j] = 0$ can be done by adding no more than 10 linear equations; exactly one such solution exists

	Solution 1
	$x_0 \oplus x_2 = 0$
	$x_1 = 0$
	$x_3 = 1$
	$x_4 = 1$
	$x_5 = 1$
	$x_7 = 0$
	$x_8 = 1$
	$x_9 = 0$
	$x_{10} = 0$
	$x_{12} = 1$
$F(x_0, \ldots, x_6) =$	1
$F(x_1, \ldots, x_7) =$	$x_2 \oplus 1$
$F(x_2, \ldots, x_8) =$	0
$F(x_3, \ldots, x_9) =$	0
$F(x_4, \ldots, x_{10}) =$	1
$F(x_5, \ldots, x_{11}) =$	1
$F(x_6, \ldots, x_{12}) =$	0

as follows:

$$\begin{aligned} F(a,b,c,d,e,f,g) = & abcdefg + abcdef + abcefg + acdefg + abceg + abdef + \\ & abdeg + abefg + acdef + acdfg + acefg + adefg + bcdfg + \\ & bdefg + cdefg + abcf + abcg + abdg + acdf + adef + adeg + \\ & adfg + bcde + bceg + bdeg + cdef + abc + abe + abf + abg + \\ & acg + adf + adg + aef + aeg + bcf + bcg + bde + bdf + beg + \\ & bfg + cde + cdf + def + deg + dfg + ad + ae + bc + bd + \\ & cd + ce + df + dg + ef + fg + a + b + c + f + 1 \ , \end{aligned} \tag{5}$$

where the multiplication and addition are taken in $\mathbb{F}_2$ (i.e., they are the same as bitwise XOR and bitwise AND, respectively).

E Distinguishing Attacks on the Full 32-Round ESSENCE-256

The attacks described in Sect. 6.2 can be easily extended to the full ESSENCE-256 block cipher. Let us suppose the key k and the plaintext are related such that

Table 5 – Making F linear for position j where $A[j] = L(A)[j] = 0$ can be done by adding no more than 6 linear equations; at least six such solutions exist

	Solution 1	Solution 2	Solution 3
	$x_3 = 0$ $x_4 = 0$ $x_5 = 1$ $x_6 = 0$ $x_7 = 1$ $x_9 = 1$	$x_3 = 0$ $x_4 = 0$ $x_5 = 1$ $x_6 = 1$ $x_7 = 1$ $x_8 = 1$	$x_3 = 1$ $x_4 = 1$ $x_5 = 1$ $x_6 = 1$ $x_7 = 1$ $x_8 = 1$
$F(x_1, \ldots, x_7) =$	$x_1 \oplus 1$	x_2	x_1
$F(x_2, \ldots, x_8) =$	$x_2 \oplus x_8 \oplus 1$	$x_2 \oplus 1$	x_2
$F(x_3, \ldots, x_9) =$	$x_8 \oplus 1$	$x_9 \oplus 1$	x_9
$F(x_4, \ldots, x_{10}) =$	x_8	0	$x_9 \oplus x_{10} \oplus 1$
$F(x_5, \ldots, x_{11}) =$	$x_8 \oplus x_{10} \oplus 1$	$x_{10} \oplus x_{11} \oplus 1$	$x_{10} \oplus x_{11} \oplus 1$

	Solution 4	Solution 5	Solution 6
	$x_4 = 0$ $x_5 = 1$ $x_6 = 1$ $x_7 = 0$ $x_8 = 1$ $x_9 = 0$	$x_4 = 0$ $x_5 = 1$ $x_6 = 1$ $x_7 = 0$ $x_8 = 1$ $x_{10} = 0$	$x_4 = 0$ $x_5 = 1$ $x_6 = 1$ $x_7 = 0$ $x_8 = 1$ $x_{11} = 1$
$F(x_1, \ldots, x_7) =$	$x_1 \oplus x_2 \oplus 1$	$x_1 \oplus x_2 \oplus 1$	$x_1 \oplus x_2 \oplus 1$
$F(x_2, \ldots, x_8) =$	$x_3 \oplus 1$	$x_3 \oplus 1$	$x_3 \oplus 1$
$F(x_3, \ldots, x_9) =$	0	x_9	x_9
$F(x_4, \ldots, x_{10}) =$	$x_{10} \oplus 1$	$x_9 \oplus 1$	$x_9 \oplus x_{10} \oplus 1$
$F(x_5, \ldots, x_{11}) =$	$x_{10} \oplus 1$	$x_9 \oplus 1$	$x_9 \oplus x_{10} \oplus 1$

Table 6 – A message pair satisfying the first 9 rounds of the characteristic of Table 1

i	m_i	m'_i	$m_i \oplus m'_i$
0	FFFFFFFFFFFFFFFF	FFFFFFFFFFFFFFFF	0000000000000000
1	1A001021983836CB	1A001021983836CB	0000000000000000
2	5809832A1DEA2458	5809832A1DEA2458	0000000000000000
3	8AEF5FEBEB9FDAAB	8AEF5FEBEB9FDAAB	0000000000000000
4	32F9D8578015D297	32F9D8578015D297	0000000000000000
5	0D031372423B91AC	0D031372423B91AC	0000000000000000
6	B804AC08CD97E348	B804AC08CD97E348	0000000000000000
7	E8BB8E649DC3B35F	E2BB9E450DF3859C	0A001021903036C3

after 18 rounds, $r_0[0] = k_0[0]$. Given this, using similar arguments as those used to derive (2), we obtain that at the end of 32 rounds, if $L(r_7)[0] = 0$, then

$$Pr[r_6[0] = 0] = \frac{1}{2} + \frac{1}{2^7} \ . \tag{6}$$

We can thus construct a distinguisher by collecting 2^{17} outputs $r_6[0]$, after 32 rounds, generated by as many keys (so that the samples are independent) given that after 18 rounds, $k_0[0] = r_0[0]$. In other words, the adversary first tests whether $k_0[0] = r_0[0]$ after 18 rounds. If this condition is satisfied, she collects the output $r_6[0]$ after 32 rounds provided $L(r_7)[0] = 0$. Therefore, this constitutes a known-key distinguishing attack which one may view as an attack on a large set of weak keys. Alternatively, the attack scenario may be such that two bits of the internal state after 18 rounds are leaked to the adversary. A similar assumption was made in [3], as a model for certain side-channel attacks. More generally, this scenario is captured by the notion of leakage resilience [4, 14], i.e., security when "even a bounded amount of arbitrary (adversarially chosen) information on the internal state (...) is leaked during computation" [4]. Although this assumption leads to trivial attacks (e.g., observe the full internal state of AES at the penultimate rounds), it assists to evaluate security against a wider range of adversaries, and to better understand the resilience of algorithms against "extreme" adversaries.

Since the condition $k_0[0] = r_0[0]$ (after 18 rounds) holds with 0.5 probability, the attacker would need to examine with $2^{17} \cdot 2 = 2^{18}$ randomly generated keys to mount the distinguishing attack with a success probability of 0.9772.

It is easy to see that distinguishers of the same complexity can be built by collecting any other bit of r_6 (after 32 rounds) because F operates in a bitsliced manner. As in Sect. 6.5, when the attacker can observe both the chaining value input and the compression function output, the above distinguishers can be applied onto the compression function as well.

F Key-Recovery Attacks on 32-Round ESSENCE

In Appendix E, we extended the distinguisher on 14-round ESSENCE-256 to 32 rounds by selecting plaintexts based upon the intermediate value of $r_0[j]$ and $k_0[j]$ at round 18. This result may be viewed in terms of a known plaintext key-recovery attack against a vulnerable implementation of the ESSENCE-256 block cipher. Let us say that we are attacking such an implementation of the 32-round ESSENCE-256 block cipher where through some means (side-channel analysis, cache pollution, etc.) we can read bit j of r_0 after 18 rounds. Like in Sect. 6.6, we focus on a subset of 2^{14} plaintexts where $r_0[j] = 0$ (or 1) for all 2^{14} texts after 18 rounds. Applying the same analysis as in Sect. 6.6 to the remaining 14 rounds gives us the value of $k_0[j]$ at round 18. If our vulnerable implementation allows us to read all the bit positions of r_0, then with probability 0.48, we can recover the full key-word k_0 at round 18. Since the key schedule is a bijection (and easily invertible) we can recover the original key with minimal effort. Again, a similar analysis can be applied to the other members of the ESSENCE family of block ciphers.

Publication Chapter

The Differential Analysis of S-Functions

Publication Data

Nicky Mouha, Vesselin Velichkov, Christophe De Cannière, and Bart Preneel. The Differential Analysis of S-Functions. In Alex Biryukov, Guang Gong, and Douglas R. Stinson, editors, *Selected Areas in Cryptography*, volume 6544 of *Lecture Notes in Computer Science*, pages 36–56. Springer, 2010.

Contributions

- Main author.

The Differential Analysis of S-functions*,†

Nicky Mouha‡, Vesselin Velichkov§, Christophe De Cannière¶, and Bart Preneel

[1] Department of Electrical Engineering ESAT/SCD-COSIC,
Katholieke Universiteit Leuven. Kasteelpark Arenberg 10, B-3001 Heverlee, Belgium.
[2] Interdisciplinary Institute for BroadBand Technology (IBBT), Belgium.
{Nicky.Mouha,Vesselin.Velichkov,Christophe.DeCanniere}@esat.kuleuven.be

Abstract. An increasing number of cryptographic primitives use operations such as addition modulo 2^n, multiplication by a constant and bitwise Boolean functions as a source of non-linearity. In NIST's SHA-3 competition, this applies to 6 out of the 14 second-round candidates. In this paper, we generalize such constructions by introducing the concept of S-functions. An S-function is a function that calculates the i-th output bit using only the inputs of the i-th bit position and a finite state $S[i]$. Although S-functions have been analyzed before, this paper is the first to present a fully general and efficient framework to determine their differential properties. A precursor of this framework was used in the cryptanalysis of SHA-1. We show how to calculate the probability that given input differences lead to given output differences, as well as how to count the number of output differences with non-zero probability. Our methods are rooted in graph theory, and the calculations can be efficiently performed using matrix multiplications.

Keywords: Differential cryptanalysis, S-function, xdp^+, $\text{xdp}^{\times C}$, $\text{adp}^{\oplus}$, counting possible output differences, ARX.

1 Introduction

Since their introduction to cryptography, differential cryptanalysis [7] and linear cryptanalysis [26] have shown to be two of the most important techniques in both the design and cryptanalysis of symmetric-key cryptographic primitives.

*The framework proposed in this paper is accompanied by a software toolkit, available at http://www.ecrypt.eu.org/tools/s-function-toolkit

†This work was supported in part by the IAP Program P6/26 BCRYPT of the Belgian State (Belgian Science Policy), and in part by the European Commission through the ICT program under contract ICT-2007-216676 ECRYPT II.

‡This author is funded by a research grant of the Institute for the Promotion of Innovation through Science and Technology in Flanders (IWT-Vlaanderen).

§DBOF Doctoral Fellow, K.U.Leuven, Belgium.

¶Postdoctoral Fellow of the Research Foundation – Flanders (FWO).

Differential cryptanalysis was introduced by Biham and Shamir in [7]. For block ciphers, it is used to analyze how input differences in the plaintext lead to output differences in the ciphertext. If this happens in a non-random way, this can be used to build a distinguisher or even a key-recovery attack.

The analysis of how differences propagate through elementary components of cryptographic designs is therefore essential to differential cryptanalysis. As typical S-boxes are no larger than 8×8, this analysis can be done by building a difference distribution table. Such a difference distribution table lists the number of occurrences of every combination of input and output differences.

The combination of S-box layers and permutation layers with good cryptographic properties, are at the basis of the wide-trail design. The wide-trail design technique is used in AES [10] to provide provable resistance against both linear and differential cryptanalysis attacks.

However, not all cryptographic primitives are based on S-boxes. Another option is to use only operations such as addition modulo 2^n, exclusive or (xor), Boolean functions, bit shifts and bit rotations. For Boolean functions, we assume that the same Boolean function is used for each bit position i of the n-bit input words.

Each of these operations is very well suited for implementation in software, but building a difference distribution table becomes impractical for commonly used primitives where $n = 32$ or $n = 64$. Examples using such constructions include the XTEA block cipher [32], the Salsa20 stream cipher family [5], as well as the hash functions MD5, SHA-1, and 6 out of 14 second-round candidates[3] of NIST's SHA-3 hash function competition [31].

In this paper, we present the first known fully general framework to analyze these constructions efficiently. It is inspired by the cryptanalysis techniques for SHA-1 by De Cannière and Rechberger [12] (clarified in [30]), and by methods introduced by Lipmaa, Wallén and Dumas [23]. The framework is used to calculate the probability that given input differences lead to given output differences, as well as to count the number of output differences with non-zero probability. Our methods are based on graph theory, and the calculations can be efficiently performed using matrix multiplications. We show how the framework can be used to analyze several commonly used constructions.

Notation is defined in Table 1. Section 2 defines the concept of an S-function. This type of function can be analyzed using the framework of this paper. The differential probability xdp^+ of addition modulo 2^n, when differences are expressed using xor, is analyzed in Sect. 3. We show how to calculate xdp^+ with an arbitrary number of inputs. In Sect 4, we study the differential probability $\mathrm{adp}^\oplus$ of xor when differences are expressed using addition modulo 2^n. Counting the number of output differences with non-zero probability is the subject of Sect. 5. We conclude in Sect. 6. The matrices obtained for xdp^+ are listed in Appendix A. We show

[3]The hash functions BLAKE [4], Blue Midnight Wish [14], CubeHash [6], Shabal [8], SIMD [20] and Skein [13] can be analyzed using the general framework that is introduced in this paper.

Table 1 – Notation

Notation	Description
$x \parallel y$	concatenation of the strings x and y
$\|A\|$	number of elements of set A
$x \ll s$	shift of x to the left by s positions
$x \gg s$	shift of x to the right by s positions
$x \lll s$	rotation of x to the left by s positions
$x \ggg s$	rotation of x to the right by s positions
$x + y$	addition of x and y modulo 2^n (in text)
$x \boxplus y$	addition of x and y modulo 2^n (in figures)
$x[i]$	selection: bit (or element) at position i of word x, where $i = 0$ is the least significant bit (element)

all possible subgraphs for xdp^+ in Appendix B. In Appendix C, we extend xdp^+ to an arbitrary number of inputs. The computation of $\text{xdp}^{\times C}$ is explained in Appendix D.

2 S-Functions

In this section, we define S-functions, the type of functions that can be analyzed using our framework. In order to show the broad range of applicability of the proposed technique, we give several examples of functions that follow our definition.

An S-function (short for "state function") accepts n-bit words $a_1, a_2, \ldots, a_k$ and a list of states $S[i]$ (for $0 \leq i < n$) as input, and produces an n-bit output word b in the following way:

$$(b[i], S[i+1]) = f(a_1[i], a_2[i], \ldots, a_k[i], S[i]), \quad 0 \leq i < n \ . \tag{1}$$

Initially, we set $S[0] = 0$. Note that f can be any arbitrary function that can be computed using only input bits $a_1[i], a_2[i], \ldots, a_k[i]$ and state $S[i]$. For conciseness, the same function f is used for every bit $0 \leq i < n$. Our analysis, however, does not require functions f to be the same, and not even to have the same number of inputs. A schematic representation of an S-function is given in Fig. 1.

Examples of S-functions include addition, subtraction and multiplication by a constant (all modulo 2^n), exclusive-or (xor) and bitwise Boolean functions. Although this paper only analyzes constructions with one output b, the extension to multiple outputs is straightforward. Our technique therefore also applies to larger constructions, such as the Pseudo-Hadamard Transform used in SAFER [1] and Twofish [34], and first analyzed in [21].

With a minor modification, the concept of S-functions allows that the inputs $a_1, a_2, \ldots, a_k$ and the output b are rotated (or reordered) as well. This corresponds to rotating (or reordering) the bits of the input and output of f. This results in

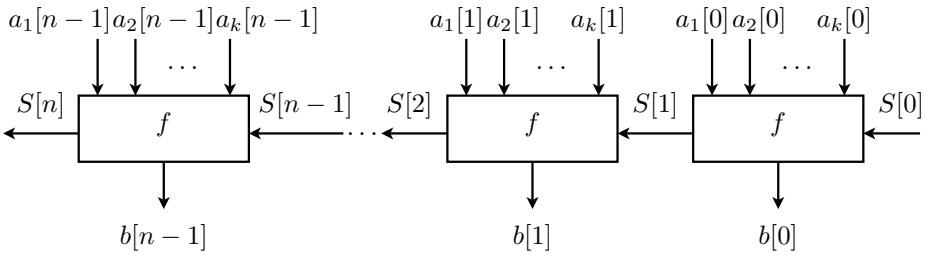


Figure 1 – Representation of an S-function

exactly the same S-function, but the input and output variables are relabeled accordingly. An entire step of SHA-1 as well as the `MIX` primitive of the block cipher RC2 can therefore be seen as an S-function. If the extension to multiple output bits is made, this applies as well to an entire step of SHA-2: for every step of SHA-2, two 32-bit registers are updated.

Every S-function is also a *T-function*, but the reverse is not always true. Proposed by Klimov and Shamir [19], a T-function is a mapping in which the i-th bit of the output depends only on bits $0, 1, \ldots, i$ of the input. Unlike a T-function, the definition of an S-function requires that the dependence on bits $0, 1, \ldots, i-1$ of the input can be described by a finite number of states (independent of n). Therefore, squaring modulo 2^n is a T-function, but not an S-function.

In [11], Daum introduced the concept of a *narrow T-function*. A w-narrow T-function computes the i-th output bit based on some information of length w bits computed from all previous input bits. An S-function, however, requires only the i-th input bit and a state $S[i]$ to calculate the i-th output bit and the next state $S[i+1]$. There is a subtle difference between narrow T-functions and S-functions. If the number of states is finite and not dependent on the word length n, it may not always be possible for a narrow T-function to compute $S[i+1]$ from the previous state $S[i]$ and the i-th input bit.

It is possible to simulate every S-function using a *finite-state machine* (FSM), also known as a finite-state automaton (FSA). This finite-state machine has k inputs $a_1[i], a_2[i], \ldots, a_k[i]$, and one state for every value of $S[i]$. The output is $b[i]$. The FSM is clocked n times, for $0 \leq i < n$. From (1), we see that the output depends on both the current state and the input. The type of FSM we use is therefore a Mealy machine [27].

The straightforward hardware implementation of an S-function corresponds to a *bit-serial* design. Introduced by Lyon in [24, 25], a bit-serial hardware architecture treats all n bits in sequence on a single hardware unit. Every bit requires one clock cycle to be processed.

The S-function framework can also be used in differential cryptanalysis, when the inputs and outputs are xor- or additive differences. Assume that every input pair (x_1, x_2) satisfies a difference $\Delta^{\bullet}x$, using some group operator $\bullet$. Then, if both

x_1 and $\Delta^\bullet x$ are given, we can calculate $x_2 = x_1 \bullet \Delta^\bullet x$. It is then straightforward to define a function to calculate the output values and the output difference as well. This approach will become clear in the following sections, when we calculate the differential probabilities xdp$^+$ and adp$^\oplus$ of modular addition and xor respectively.

3 Computation of xdp$^+$

3.1 Introduction

In this section, we study the differential probability xdp$^+$ of addition modulo 2^n, when differences are expressed using xor. Until [22], no algorithm was published to compute xdp$^+$ faster than exhaustive search over all inputs. In [22], the first algorithm with a linear time in the word length n was proposed. If n-bit computations can be performed, the time complexity of this algorithm becomes sublinear in n.

In [23], xdp$^+$ is expressed using the mathematical concept of rational series. It is shown that this technique is more general, and can also be used to calculate the differential probability adp$^\oplus$ of xor, when differences are expressed using addition modulo 2^n.

In this paper, we present a new technique for the computation of xdp$^+$, using graph theory. The main advantage of the proposed method over existing techniques, is that it is not only more general, but also allows results to be obtained in a fully automated way. The only requirement is that both the operations and the input and output differences of the cryptographic component can be written as the S-function of Sect. 2. In the next section, we introduce this technique to calculate the probability xdp$^+$.

3.2 Defining the Probability xdp$^+$

Given n-bit words $x_1, y_1, \Delta^\oplus x, \Delta^\oplus y$, we calculate $\Delta^\oplus z$ using

$$x_2 \leftarrow x_1 \oplus \Delta^\oplus x \ , \tag{2}$$

$$y_2 \leftarrow y_1 \oplus \Delta^\oplus y \ , \tag{3}$$

$$z_1 \leftarrow x_1 + y_1 \ , \tag{4}$$

$$z_2 \leftarrow x_2 + y_2 \ , \tag{5}$$

$$\Delta^\oplus z \leftarrow z_2 \oplus z_1 \ . \tag{6}$$

We then define $\mathrm{xdp}^+(\alpha, \beta \rightarrow \gamma)$ as

$$\mathrm{xdp}^+(\alpha, \beta \rightarrow \gamma) = \frac{|\{(x_1, y_1) : \Delta^\oplus x = \alpha, \Delta^\oplus y = \beta, \Delta^\oplus z = \gamma\}|}{|\{(x_1, y_1) : \Delta^\oplus x = \alpha, \Delta^\oplus y = \beta\}|} \ , \tag{7}$$

$$= 4^{-n} |\{(x_1, y_1) : \Delta^\oplus x = \alpha, \Delta^\oplus y = \beta, \Delta^\oplus z = \gamma\}| \ , \tag{8}$$

as there are $2^n \cdot 2^n = 4^n$ combinations for the two n-bit words (x_1, y_1).

3.3 Constructing the S-Function for xdp^+

We rewrite (2)-(6) on a bit level, using the formulas for multiple-precision addition in radix 2 [28, §14.2.2]:

$$x_2[i] \leftarrow x_1[i] \oplus \Delta^{\oplus} x[i] \ , \tag{9}$$
$$y_2[i] \leftarrow y_1[i] \oplus \Delta^{\oplus} y[i] \ , \tag{10}$$
$$z_1[i] \leftarrow x_1[i] \oplus y_1[i] \oplus c_1[i] \ , \tag{11}$$
$$c_1[i+1] \leftarrow (x_1[i] + y_1[i] + c_1[i]) \gg 1 \ , \tag{12}$$
$$z_2[i] \leftarrow x_2[i] \oplus y_2[i] \oplus c_2[i] \ , \tag{13}$$
$$c_2[i+1] \leftarrow (x_2[i] + y_2[i] + c_2[i]) \gg 1 \ , \tag{14}$$
$$\Delta^{\oplus} z[i] \leftarrow z_2[i] \oplus z_1[i] \ , \tag{15}$$

where carries $c_1[0] = c_2[0] = 0$. Let us define

$$S[i] \leftarrow (c_1[i], c_2[i]) \ , \tag{16}$$
$$S[i+1] \leftarrow (c_1[i+1], c_2[i+1]) \ . \tag{17}$$

Then, (9)-(15) correspond to the S-function

$$(\Delta^{\oplus} z[i], S[i+1]) = f(x_1[i], y_1[i], \Delta^{\oplus} x[i], \Delta^{\oplus} y[i], S[i]), \quad 0 \le i < n \ . \tag{18}$$

Because we are adding two words in binary, both carries $c_1[i]$ and $c_2[i]$ can be either 0 or 1.

3.4 Computing the Probability xdp^+

In this section, we use the S-function (18), defined by (9)-(15), to compute xdp^+. We explain how this probability can be derived from the number of paths in a graph, and then show how to calculate xdp^+ using matrix multiplications.

Graph Representation.

For $0 \le i \le n$, we will represent every state $S[i]$ as a vertex in a graph (Fig. 2). This graph consists of several subgraphs, containing only vertices $S[i]$ and $S[i+1]$ for some bit position i. We repeat the following for all combinations of $(\alpha[i], \beta[i], \gamma[i])$:

Set $\alpha[i] \leftarrow \Delta^{\oplus} x[i]$ and $\beta[i] \leftarrow \Delta^{\oplus} y[i]$. Then, we loop over all values of $(x_1[i], y_1[i], S[i])$. For each combination, $\Delta^{\oplus} z[i]$ and $S[i]$ are uniquely determined by (18). We draw an edge between $S[i]$ and $S[i+1]$ in the subgraph, if and only if $\Delta^{\oplus} z[i] = \gamma[i]$. Note that several edges may have the same set of endpoints.

For completeness, all subgraphs for xdp^+ are given in Appendix B. Let α, β, γ be given. As shown in Fig. 2, we construct a full graph containing all vertices $S[i]$ for $0 \le i \le n$, where the edges between these vertices correspond to those of the subgraphs for $\alpha[i], \beta[i], \gamma[i]$.

Theorem 1. *Let P be the set of all paths from $(c_1[0], c_2[0]) = (0,0)$ to any of the four vertices $(c_1[n], c_2[n]) \in \{(0,0),(0,1),(1,0),(1,1)\}$ (see Fig. 2). Then, there is exactly one path in P for every pair (x_1, y_1) of the set in the definition of* xdp$^+$*, given by (8).*

Proof. Given $x_1[i]$, $y_1[i]$, $\Delta^{\oplus}x[i]$, $\Delta^{\oplus}y[i]$, $c_1[i]$ and $c_2[i]$, the values of $\Delta^{\oplus}z[i]$, $c_1[i+1]$ and $c_2[i+1]$ are uniquely determined by (9)-(15). All paths in P start at $(c_1[0], c_2[0]) = (0,0)$, and only consist of vertices $(c_1[i], c_2[i])$ for $0 \le i \le n$ that satisfy (9)-(15). Furthermore, edges for which $\Delta^{\oplus}z[i] \neq \gamma[i]$ are not in the graph, and therefore not part of any path P. Thus by construction, P contains every pair (x_1, y_1) of the set in (8) exactly once. □

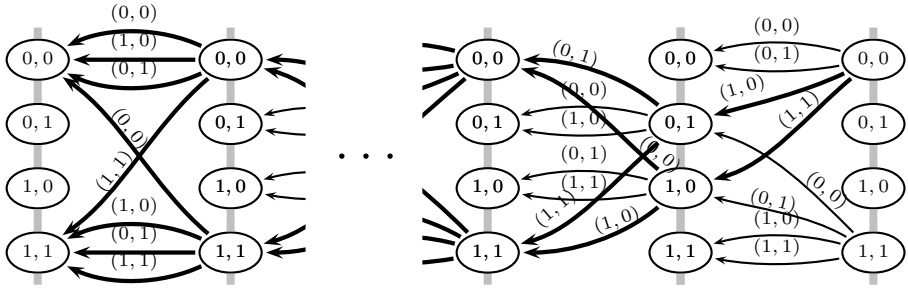


Figure 2 – An example of a full graph for xdp$^+$. Vertices $(c_1[i], c_2[i]) \in \{(0,0),(0,1),(1,0),(1,1)\}$ correspond to states $S[i]$. There is one edge for every input pair (x_1, y_1). All paths that satisfy input differences α, β and output difference γ are shown in bold. They define the set of paths P of Theorem 1.

Multiplication of Matrices.

The differential $(\alpha[i], \beta[i] \to \gamma[i])$ at bit position i is written as a bit string $w[i] \leftarrow \alpha[i] \parallel \beta[i] \parallel \gamma[i]$. Each $w[i]$ corresponds to a subgraph of Appendix B. As this subgraph is a bipartite graph, we can construct its biadjacency matrix $A_{w[i]} = [x_{kj}]$, where x_{kj} is the number of edges that connect vertices $j = S[i]$ and $k = S[i+1]$. These matrices are given in Appendix A.

Let the number of states $S[i]$ be N. Define $1 \times N$ matrix $L = [\ 1 \quad 1 \quad \cdots \quad 1\ ]$ and $N \times 1$ matrix $C = [\ 1 \quad 0 \quad \cdots \quad 0\ ]^T$. For any directed acyclic graph, the number of paths between two vertices can be calculated as a matrix multiplication [9]. We can therefore calculate the number of paths P as

$$|P| = LA_{w[n-1]} \cdots A_{w[1]} A_{w[0]} C \ . \tag{19}$$

Using (8), we find that $\mathrm{xdp}^+(\alpha, \beta \rightarrow \gamma) = 4^{-n}|P|$. Therefore, we can define $A^*_{w[i]} = A_{w[i]}/4$, and obtain

$$\mathrm{xdp}^+(\alpha, \beta \rightarrow \gamma) = LA^*_{w[n-1]} \cdots A^*_{w[1]} A^*_{w[0]} C \ . \tag{20}$$

As such, we obtain a similar expression as in [23], where the xdp^+ was calculated using the concept of rational series. Our matrices $A^*_{w[i]}$ are of size 4×4 instead of 2×2, however. We now give a simple algorithm to reduce the size of our matrices.

3.5 Minimizing the Size of the Matrices for xdp^+.

Corresponding to (20), we can define a non-deterministic finite-state automaton (NFA) with states $S[i]$ and inputs $w[i]$. Compared to a deterministic finite-state automaton, the transition *function* is replaced by a transition *relation*. There are several choices for the next state, each with a certain probability. This NFA can be minimized as follows.

First, we remove non-accessible states. A state is said to be non-accessible, if it can never be reached from the initial state $S[0] = 0$. This can be done using a simple algorithm to check for connectivity, with a time complexity that is linear in the number of edges.

Secondly, we merge indistinguishable states. The method we propose, is similar to the FSM reduction algorithms found independently by [17] and [29]. Initially, we assign all states $S[i]$ to one equivalence class $T[i] = 0$. We try to partition this equivalence class into smaller classes, by repeating the following steps:

- We iterate over all states $S[i]$.
- For every input $w[i]$ and every equivalence class $T[i]$, we sum the transition probabilities to every state $S[i]$ of this equivalence class.
- If these sums are different for two particular states $S[i]$, we partition them into different equivalence classes $T[i]$.

The algorithm stops when the equivalence classes $T[i]$ cannot be partitioned further.

In the case of xdp^+, we find that all states are accessible. However, there are only two indistinguishable states: $T[i] = 0$ and $T[i] = 1$ when $(c_1[i], c_2[i])$ are elements of the sets $\{(0,0),(1,1)\}$ and $\{(0,1),(1,0)\}$ respectively. Our algorithm shows how matrices $A^*_{w[i]}$ of (20) can be reduced to matrices $A'_{w[i]}$ of size 2×2. These matrices are the same as in [23], but they have now been obtained in an automated way. For completeness, they are given again in Appendix A. Our approach also allows a new interpretation of matrices $A'_{w[i]}$ in the context of S-functions (18): every matrix entry defines the transition probability between two sets of states, where all states of one set were shown to be equivalent by the minimization algorithm.

3.6 Extensions of xdp^+

In this section, we show how S-functions not only lead to expressions to calculate $\mathrm{xdp}^+(\alpha, \beta \to \gamma)$, but can be applied to related constructions as well.

Multiple Inputs $\mathrm{xdp}^+(\alpha, \beta, \ldots \to \gamma)$.

Using the framework of this paper, we can easily calculate xdp^+ for more than two (independent) inputs. This calculation can be used, for example, in the differential cryptanalysis of XTEA [32] using xor differences. In [15], a 3-round iterative characteristic $(\alpha, 0) \to (\alpha, 0)$ is used, where $\alpha =$ `0x80402010`. In the third round of the characteristic, there are two consecutive applications of addition modulo 2^n. Separately, these result in probabilities $\mathrm{xdp}^+(\alpha, 0 \to \alpha) = 2^{-3}$ and $\mathrm{xdp}^+(\alpha, \alpha \to 0) = 2^{-3}$. It is shown in [15] that the joint probability $\mathrm{xdp}^+(\alpha, 0, \alpha \to 0)$ is higher than the product of the probabilities $2^{-3} \cdot 2^{-3} = 2^{-6}$, and is estimated to be $2^{-4.755}$. Using the techniques presented in this paper, we evaluate the exact joint probability to be 2^{-3}. We also verified this experimentally. The calculations are detailed in Appendix C. This result can be trivially confirmed using the commutativity property of addition: $\mathrm{xdp}^+(\alpha, \alpha \to 0) \cdot \mathrm{xdp}^+(0, 0 \to 0) = \mathrm{xdp}^+(\alpha, \alpha \to 0) = 2^{-3}$. Nevertheless, our method is more general and can be used for any input difference.

Multiplication by a Constant $\mathrm{xdp}^{\times C}$.

A problem related to xdp^+, is the differential probability of multiplication by a constant C where differences are expressed by xor. We denote this probability by $\mathrm{xdp}^{\times C}$. In the hash function Shabal [8], multiplications by 3 and 5 occur. EnRUPT [33] uses a multiplication by 9. In the cryptanalysis of EnRUPT [18], a technique is described to calculate $\mathrm{xdp}^{\times 9}$. This technique is based on a precursor of the framework in this paper. In Appendix D, we show how each of these probabilities can be calculated efficiently, using the framework of this paper. The example of $\mathrm{xdp}^{\times 3}$ is fully worked out.

Pseudo-Hadamard Transform $\mathrm{xdp}^{\mathrm{PHT}}$.

The Pseudo-Hadamard Transform (PHT) is defined as $\mathrm{PHT}(x_1, x_2) = (2x_1 + x_2, x_1 + x_2)$. It is a reversible operation, used to provide diffusion in several cryptographic primitives, including block ciphers SAFER [1] and Twofish [34]. Its differential properties were first studied in [21]. If we allow an S-function to be constructed with two outputs b_1 and b_2, the analysis of this construction becomes straightforward using the techniques of this paper.

Step Functions of the MD4 Family.

The MD4 family consists of several hash functions, including MD4, MD5, SHA-1, SHA-2 and HAS-160. Currently, the most commonly used hash functions worldwide are MD5 and SHA-1. The step functions of MD4, HAS-160 and SHA-1 can each be represented as an S-function. This applies as well to the `MIX` primitive of the block cipher RC2. They can therefore also be analyzed using our framework. The calculation of the uncontrolled probability $P_u(i)$ in the cryptanalysis of SHA-1 [12,30] uses a precursor of the techniques in this paper. By making the extension to multiple outputs, the same analysis can be made as well for the step function of SHA-2.

4 Computation of $\mathrm{adp}^{\oplus}$

4.1 Introduction

In this section, we study the differential probability $\mathrm{adp}^{\oplus}$ of xor when differences are expressed using addition modulo 2^n. The best known algorithm to compute $\mathrm{adp}^{\oplus}$ was exhaustive search over all inputs, until an algorithm with a linear time in n was proposed in [23].

We show how the technique introduced in Sect. 3 for xdp^{+} can also be applied to $\mathrm{adp}^{\oplus}$. Using this, we confirm the results of [23]. The approach we introduced in this section is conceptually much easier than [23], and can easily be generalized to other constructions with additive differences.

4.2 Defining the Probability $\mathrm{adp}^{\oplus}$

Given n-bit words $x_1, y_1, \Delta^+x, \Delta^+y$, we calculate Δ^+z using

$$x_2 \leftarrow x_1 + \Delta^+x \ , \tag{21}$$
$$y_2 \leftarrow y_1 + \Delta^+y \ , \tag{22}$$
$$z_1 \leftarrow x_1 \oplus y_1 \ , \tag{23}$$
$$z_2 \leftarrow x_2 \oplus y_2 \ , \tag{24}$$
$$\Delta^+z \leftarrow z_2 - z_1 \ . \tag{25}$$

Similar to (8), we define $\mathrm{adp}^{\oplus}(\alpha, \beta \to \gamma)$ as

$$\mathrm{adp}^{\oplus}(\alpha, \beta \to \gamma) = \frac{|\{(x_1, y_1) : \Delta^+x = \alpha, \Delta^+y = \beta, \Delta^+z = \gamma\}|}{|\{(x_1, y_1) : \Delta^+x = \alpha, \Delta^+y = \beta\}|} \ , \tag{26}$$
$$= 4^{-n}|\{(x_1, y_1) : \Delta^+x = \alpha, \Delta^+y = \beta, \Delta^+z = \gamma\}| \ , \tag{27}$$

as there are $2^n \cdot 2^n = 4^n$ combinations for the two n-bit words (x_1, y_1).

4.3 Constructing the S-function for $\mathrm{adp}^{\oplus}$

We rewrite (21)-(25) on a bit level, again using the formulas for multiple-precision addition and subtraction in radix 2 [28, §14.2.2]:

$$x_2[i] \leftarrow x_1[i] \oplus \Delta^+x[i] \oplus c_1[i] \ , \tag{28}$$
$$c_1[i+1] \leftarrow (x_1[i] + \Delta^+x[i] + c_1[i]) \gg 1 \ , \tag{29}$$
$$y_2[i] \leftarrow y_1[i] \oplus \Delta^+y[i] \oplus c_2[i] \ , \tag{30}$$
$$c_2[i+1] \leftarrow (y_1[i] + \Delta^+y[i] + c_2[i]) \gg 1 \ , \tag{31}$$
$$z_1[i] \leftarrow x_1[i] \oplus y_1[i] \ , \tag{32}$$
$$z_2[i] \leftarrow x_2[i] \oplus y_2[i] \ , \tag{33}$$
$$\Delta^+z[i] \leftarrow (z_2[i] \oplus z_1[i] \oplus c_3[i])[0] \ , \tag{34}$$
$$c_3[i+1] \leftarrow (z_2[i] - z_1[i] + c_3[i]) \gg 1 \ , \tag{35}$$

where carries $c_1[0] = c_2[0] = 0$ and borrow $c_3[0] = 0$. We assume all variables to be integers in two's complement notation, all shifts are signed shifts. Let us define

$$S[i] \leftarrow (c_1[i], c_2[i], c_3[i]) \ , \tag{36}$$
$$S[i+1] \leftarrow (c_1[i+1], c_2[i+1], c_3[i+1]) \ . \tag{37}$$

Then (28)-(35) correspond to the S-function

$$(\Delta^+z[i], S[i+1]) = f(x_1[i], y_1[i], \Delta^+x[i], \Delta^+y[i], S[i]), \qquad 0 \le i < n \ . \tag{38}$$

Both carries $c_1[i]$ and $c_2[i]$ can be either 0 or 1; borrow $c_3[i]$ can be either 0 or -1.

4.4 Computing the Probability $\mathrm{adp}^{\oplus}$

Using the description of the S-function (38), the calculation of $\mathrm{adp}^{\oplus}$ follows directly from Sect. 3.4. We obtain eight matrices $A_{w[i]}$ of size 8×8. After applying the minimization algorithm of Sect. 3.5, the size of the matrices remains unchanged. Here, we use the expression $-4 \cdot c_3[i] + 2 \cdot c_2[i] + c_1[i]$ as an index to order the states $S[i]$. The matrices we obtain are then permutation similar to those of [23]; their states $S'[i]$ can be related to our states $S[i]$ by permutation σ:

$$\sigma = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 0 & 4 & 2 & 6 & 1 & 5 & 3 & 7 \end{pmatrix} \ . \tag{39}$$

We calculate the number of paths using (19). From (27), we get $\mathrm{adp}^{\oplus}(\alpha, \beta \to \gamma) = 4^{-n}|P|$. Therefore, we can define $A^*_{w[i]} = A_{w[i]}/4$, and obtain

$$\mathrm{adp}^{\oplus}(\alpha, \beta \to \gamma) = LA^*_{w[n-1]} \cdots A^*_{w[1]} A^*_{w[0]} C \ . \tag{40}$$

5 Counting Possible Output Differences

5.1 Introduction

In the previous sections, we showed for several constructions how to calculate the probability that given input differences lead to a given output difference. A related problem is to calculate the number of *possible* output differences, when the input differences are given. We say that an output difference is possible, if it occurs with a non-zero probability.

First, we describe a naive algorithm to count the number of output differences. It has a time complexity that is exponential in the word length n. We investigate both improvements in existing literature, as well as cryptanalysis results where such a calculation is necessary.

Then, we introduce a new algorithm. We found it to be the first in existing literature with a time complexity that is linear in n. We show that our algorithm can be used for all constructions based on S-functions.

5.2 Algorithm with a Exponential Time in n

Generic Exponential-in-n Time Algorithm.

A naive, but straightforward algorithm works as follows. All output differences with non-zero probability can be represented in a search tree. Every level in this tree contains nodes of one particular bit position, with the least significant bit at the top level. This tree is traversed using depth-first search. For each output difference with non-zero probability that is found, we increment a counter for the number of output differences by one. When all nodes are traversed, this counter contains the total number of possible output differences. The time complexity of this algorithm is exponential in n, the memory complexity is linear in n.

Improvement for $\mathrm{xdc}^+(\alpha, \beta)$.

We introduce the notation $\mathrm{xdc}^+(\alpha, \beta)$ for the number of output xor-differences of addition modulo 2^n, given input xor-differences α and β. In [3], xdc^+ was used to build a key-recovery attack on top of a boomerang distinguisher for 32-round Threefish-512 [13]. They introduced a new algorithm to calculate xdc^+. The correctness of this algorithm is proven in the full version of [3], i.e. [2]. The algorithm, however, only works if one of the inputs contains either no difference, or a difference only in the most significant bit. Also, it does not generalize to other types of differences. The time complexity of this algorithm is exponential in the number of non-zero input bits, and the memory complexity is linear in the number of non-zero input bits. As a result, it is only usable in practice for sparse input differences. We were unable to find any other work on this problem in existing literature.

5.3 Algorithm with a Linear Time in n

In Sect. 3 and 4, we showed how to calculate the probability of an output difference using both graph theory and matrix multiplications. We now present a similar method to calculate the number of possible output differences. First, the general algorithm is explained. It is applicable to any type of construction based on S-functions. Then, we illustrate how the matrices for xdp^+ can be turned into matrices for xdc^+. This paper is the first to present an algorithm for this problem with a linear-in-n time complexity. We also extend the results to $\text{adp}^{\oplus}$. Our strategy is similar to the calculation of the controlled probability $P_c(i)$, used in the cryptanalysis of SHA-1 [12, 30].

Graph Representation.

As in Sect. 3.4, we will again construct a graph. Let N be the number of states $|T[i]|$ that we obtained in Sect. 3.5. For xdp^+, we found $N = 2$. We will now construct larger subgraphs, where the nodes do not represent states $T[i]$, but elements of its power set $\mathcal{P}(T[i])$. This power set $\mathcal{P}(T[i])$ contains 2^N elements, ranging from the empty set $\varnothing$ to set of all states $\{0, 1, \ldots, N-1\}$. In automata theory, this technique is known as the subset construction [16, §2.3.5]. It converts the non-deterministic finite-state automaton (NFA) of Sect. 3.5 into a deterministic finite-state automaton (DFA).

For every subgraph, the input difference bits $\alpha[i]$ and $\beta[i]$ are fixed. We then define exactly one edge for every output bit $\gamma[i]$ from every set in $\mathcal{P}(T[i])$ to the corresponding set of next states in $\mathcal{P}(T[i+1])$. The example in the next section will clarify this step.

Theorem 2. *Let P be the set of all paths that start in $\{0\}$ at position $i = 0$ and end in a non-empty set at position $i = n$. Then, the number of paths $|P|$ corresponds to the number of possible output differences.*

Proof. All paths P start in $\{0\}$ at $i = 0$, and end in a non-empty set at $i = n$. For a given output difference bit, there is exactly one edge leaving from a non-empty set of states to another non-empty set of states. Therefore by construction, every possible output difference corresponds to exactly one path in P. □

Multiplication of Matrices.

The differential $(\alpha[i], \beta[i])$ at bit position i is written as a bit string $w[i] \leftarrow \alpha[i] \parallel \beta[i]$. As in Sect. 3.4, we construct the biadjacency matrices of these subgraphs. They will be of size $2^N \times 2^N$. As we are only interested in possible output differences, these matrices can be reduced to matrices $B_{w[i]}$ of size $(2^N - 1) \times (2^N - 1)$ by removing the empty set $\varnothing$.

Define $1 \times (2^N - 1)$ matrix $L = [\ 1 \quad 1 \quad \cdots \quad 1\ ]$ and $(2^N - 1) \times 1$ matrix $C = [\ 1 \quad 0 \quad \cdots \quad 0\ ]^T$. Similar to (19), we obtain the number of possible output

differences as

$$|P| = LB_{w[n-1]} \cdots B_{w[1]} B_{w[0]} C \ . \tag{41}$$

The time complexity of (41) is linear in the word length n.

We note that these matrices can have large dimensions. However, this is often not a problem in practice, as they are typically very sparse. If we keep track of only non-zero elements, there is little memory required to store vectors, and fast algorithms exist for sparse matrix-vector multiplications. Also, the size of the matrices can be minimized using Sect. 3.5.

5.4 Computing the Number of Output Differences xdc^+

In the minimized matrices for xdp^+ (given in [23] and again in Appendix A), we refer to the states corresponding to the first and the second column as $S[i] = 0$ and $S[i] = 1$ respectively. Then, the subgraphs for xdc^+ can be constructed as in Fig. 3. Regardless of the value of the output bit, edges leaving from the empty set $\varnothing$ at i will always arrive at the empty set at $i + 1$. Assume that the input differences are $\alpha[i] = \beta[i] = 0$, and that we are in state $S[i] = 1$, represented in Fig. 3 as $\{1\}$. Recall that the matrices for xdp^+ are

$$A'_{000} = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}, \ A'_{001} = \frac{1}{2} \begin{bmatrix} 0 & 1 \\ 0 & 1 \end{bmatrix}, \tag{42}$$

for output differences $\gamma[i] = 0$ and $\gamma[i] = 1$ respectively. To find out which states can be reached from state $S[i] = 1$, we multiply both matrices to the right by $\begin{bmatrix} 0 & 1 \end{bmatrix}^T$. We obtain

$$A'_{000} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \ A'_{001} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \frac{1}{2} \begin{bmatrix} 1 \\ 1 \end{bmatrix}. \tag{43}$$

We see that we cannot reach a valid next state if $\gamma[i] = 0$, so there is an edge between $\{1\}$ at i and $\varnothing$ at $i+1$ for $\gamma[i] = 0$. If $\gamma[i] = 1$, both states can be reached. Therefore, we draw an edge between $\{1\}$ at i and $\{0, 1\}$ at $i + 1$ for $\gamma[i] = 1$. The other edges of Fig. 3 can be derived in a similar way.

Matrices $B_{00}, B_{01}, B_{10}, B_{11}$ of (41) can be derived from Fig. 3 as

$$B_{00} = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 1 & 1 \end{bmatrix}, \ B_{01} = B_{10} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 1 & 2 \end{bmatrix}, \ B_{11} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix} \ . \tag{44}$$

If the input differences are very sparse or very dense, (41) can be sped up by using

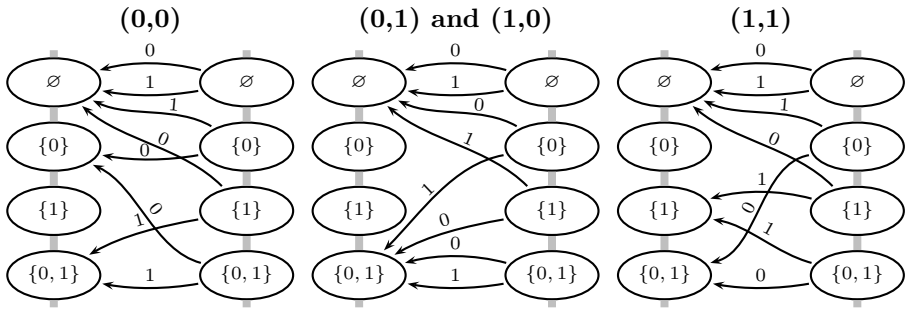


Figure 3 – All possible subgraphs for xdc^+. Vertices correspond to valid sets of states $S[i]$. There is one edge for every output difference bit $\gamma[i]$. Above each subgraph, the value of $(\alpha[i], \beta[i])$ is given in bold.

the following expressions for the powers of matrices:

$$B_{00}^k = \begin{bmatrix} 1 & k-1 & k \\ 0 & 0 & 0 \\ 0 & 1 & 1 \end{bmatrix}, \qquad B_{01}^k = B_{10}^k = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 2^{k-1} & 2^{k-1} & 2^k \end{bmatrix},$$
$$B_{11}^k = \begin{bmatrix} 0 & 0 & 0 \\ k-1 & 1 & k \\ 1 & 0 & 1 \end{bmatrix}. \tag{45}$$

This way, we obtain an algorithm with a time complexity that is linear in the number of non-zero input bits. As such, our algorithm always outperforms the naive exponential time algorithm, as well as the exponential time algorithm of [3] that only works for some input differences.

Let $L = \begin{bmatrix} 1 & 1 \end{bmatrix}$ and $C = \begin{bmatrix} 1 & 0 \end{bmatrix}^T$. We illustrate our method by recalculating the example given in [3]:

$$\mathrm{xdc}^+(\texttt{0x1000010402000000}, \texttt{0x0000000000000000}) \tag{46}$$
$$= L \cdot B_{00}^3 \cdot B_{10} \cdot B_{00}^{19} \cdot B_{10} \cdot B_{00}^5 \cdot B_{10} \cdot B_{00}^8 \cdot B_{10} \cdot B_{00}^{25} \cdot C \tag{47}$$
$$= 5880 \tag{48}$$

5.5 Calculation of $\mathrm{adc}^\oplus$

We can also calculate $\mathrm{adc}^\oplus$, which is the number of output differences for xor, when all differences are expressed using addition modulo 2^n. As the matrices $A_{w[i]}^*$ for $\mathrm{adp}^\oplus$ are of dimension 8×8, the matrices $B_{w[i]}$ of $\mathrm{adc}^\oplus$ would be of dimension $(2^8 - 1) \times (2^8 - 1) = 255 \times 255$. However, we find that only 24 out of 255 states are accessible. Furthermore, we find that all 24 accessible states are equivalent to 2 states. In the end, we obtain the following 2×2 matrices:

$$B_{00} = \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix}, \quad B_{01} = B_{10} = B_{11} = \begin{bmatrix} 0 & 0 \\ 1 & 2 \end{bmatrix} . \tag{49}$$

These matrices $B_{w[i]}$ are consistent with Theorem 2 of [23]. Although the end result is simple, this example encompasses many of the techniques presented in this paper.

6 Conclusion

In Sect. 2, we introduced the concept of an S-function, for which we build a framework in this paper. In Sect. 3, we analyzed the differential probability xdp^+ of addition modulo 2^n, when differences are expressed using xor. This probability was derived using graph theory, and calculated using matrix multiplications. We showed not only how to derive the matrices in an automated way, but also give an algorithm to minimize their size. The results are consistent with [23]. This technique was extended to an arbitrary number of inputs and to several related constructions, including an entire step of SHA-1. A precursor of the methods in this section was already used for the cryptanalysis of SHA-1 [12, 30]. We are unaware of any other fully systematic and efficient framework for the differential cryptanalysis of S-functions using xor differences.

Using the proposed framework, we studied the differential probability $\text{adp}^{\oplus}$ of xor when differences are expressed using addition modulo 2^n in Sect 4. To the best of our knowledge, this paper is the first to obtain this result in a constructive way. We verified that our matrices correspond to those obtained in [23]. As these techniques can easily be generalized, this paper provides the first known systematic treatment of the differential cryptanalysis of S-functions using additive differences.

Finally, in Sect. 5, we showed how the number of output differences with non-zero probability can be calculated. An exponential-in-n algorithm was already used for this problem in the cryptanalysis of Threefish [3]. As far as we know, this paper is the first to present an algorithm for this with a time complexity that is linear in the number of non-zero bits.

Acknowledgments. The authors would like to thank their colleagues at COSIC, and Vincent Rijmen in particular, for the fruitful discussions, as well as the anonymous reviewers for their detailed comments and suggestions. Thanks to James Quah for pointing out an error in one of the matrices of Appendix A, and for several suggestions on how to improve the text.

References

[1] R. J. Anderson, editor. *Fast Software Encryption, Cambridge Security Workshop, Cambridge, UK, December 9-11, 1993, Proceedings*, volume 809 of *Lecture Notes in Computer Science*. Springer, 1994.

[2] J.-P. Aumasson, C. Calik, W. Meier, O. Özen, R. C.-W. Phan, and K. Varıcı. Improved Cryptanalysis of Skein. Cryptology ePrint Archive, Report 2009/438, 2009. http://eprint.iacr.org/.

[3] J.-P. Aumasson, Çagdas Çalik, W. Meier, O. Özen, R. C.-W. Phan, and K. Varıcı. Improved Cryptanalysis of Skein. In M. Matsui, editor, *ASIACRYPT*, volume 5912 of *Lecture Notes in Computer Science*, pages 542–559. Springer, 2009.

[4] J.-P. Aumasson, L. Henzen, W. Meier, and R. C.-W. Phan. SHA-3 proposal BLAKE. Submission to the NIST SHA-3 Competition (Round 3), 2010. http://131002.net/blake/blake.pdf.

[5] D. J. Bernstein. The Salsa20 Family of Stream Ciphers. In M. J. B. Robshaw and O. Billet, editors, *The eSTREAM Finalists*, volume 4986 of *Lecture Notes in Computer Science*, pages 84–97. Springer, 2008.

[6] D. J. Bernstein. CubeHash specification (2.B.1). Submission to the NIST SHA-3 Competition (Round 2), 2009. http://cubehash.cr.yp.to/submission2/spec.pdf.

[7] E. Biham and A. Shamir. Differential Cryptanalysis of DES-like Cryptosystems. *J. Cryptology*, 4(1):3–72, 1991.

[8] E. Bresson, A. Canteaut, B. Chevallier-Mames, C. Clavier, T. Fuhr, A. Gouget, T. Icart, J.-F. Misarsky, M. Naya-Plasencia, P. Paillier, T. Pornin, J.-R. Reinhard, C. Thuillet, and M. Videau. Shabal, a Submission to NIST's Cryptographic Hash Algorithm Competition. Submission to the NIST SHA-3 Competition (Round 2), 2008. http://ehash.iaik.tugraz.at/uploads/6/6c/Shabal.pdf.

[9] E. W. Chittenden. On the Number of Paths in a Finite Partially Ordered Set. *The American Mathematical Monthly*, 54(7):404–405, 1947. http://www.jstor.org/stable/2304391.

[10] J. Daemen and V. Rijmen. *The Design of Rijndael: AES - The Advanced Encryption Standard*. Springer, 2002.

[11] M. Daum. Narrow T-Functions. In H. Gilbert and H. Handschuh, editors, *FSE*, volume 3557 of *Lecture Notes in Computer Science*, pages 50–67. Springer, 2005.

[12] C. De Cannière and C. Rechberger. Finding SHA-1 Characteristics: General Results and Applications. In X. Lai and K. Chen, editors, *ASIACRYPT*, volume 4284 of *Lecture Notes in Computer Science*, pages 1–20. Springer, 2006.

[13] N. Ferguson, S. Lucks, B. Schneier, D. Whiting, M. Bellare, T. Kohno, J. Callas, and J. Walker. The Skein Hash Function Family. Submission to the NIST SHA-3 Competition (Round 3), 2010. `http://www.skein-hash.info/sites/default/files/skein1.3.pdf`.

[14] D. Gligoroski, V. Klima, S. J. Knapskog, M. El-Hadedy, J. Amundsen, and S. F. Mjølsnes. Cryptographic Hash Function BLUE MIDNIGHT WISH. Submission to the NIST SHA-3 Competition (Round 2), 2009. `http://people.item.ntnu.no/~danilog/Hash/BMW-SecondRound/Supporting_Documentation/BlueMidnightWishDocumentation.pdf`.

[15] S. Hong, D. Hong, Y. Ko, D. Chang, W. Lee, and S. Lee. Differential Cryptanalysis of TEA and XTEA. In J. I. Lim and D. H. Lee, editors, *ICISC*, volume 2971 of *Lecture Notes in Computer Science*, pages 402–417. Springer, 2003.

[16] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation (3rd Edition)*. Addison-Wesley, 2006.

[17] D. A. Huffman. The synthesis of sequential switching circuits. *Journal of the Franklin Institute*, 257(3):161–190, 1954.

[18] S. Indesteege and B. Preneel. Practical Collisions for EnRUPT. *J. Cryptology*, 24(1):1–23, 2011.

[19] A. Klimov and A. Shamir. Cryptographic Applications of T-Functions. In M. Matsui and R. J. Zuccherato, editors, *Selected Areas in Cryptography*, volume 3006 of *Lecture Notes in Computer Science*, pages 248–261. Springer, 2003.

[20] G. Leurent, C. Bouillaguet, and P.-A. Fouque. SIMD Is a Message Digest. Submission to the NIST SHA-3 Competition (Round 2), 2009. `http://www.di.ens.fr/~leurent/files/SIMD.pdf`.

[21] H. Lipmaa. On Differential Properties of Pseudo-Hadamard Transform and Related Mappings. In A. Menezes and P. Sarkar, editors, *INDOCRYPT*, volume 2551 of *Lecture Notes in Computer Science*, pages 48–61. Springer, 2002.

[22] H. Lipmaa and S. Moriai. Efficient Algorithms for Computing Differential Properties of Addition. In M. Matsui, editor, *FSE*, volume 2355 of *Lecture Notes in Computer Science*, pages 336–350. Springer, 2001.

[23] H. Lipmaa, J. Wallén, and P. Dumas. On the Additive Differential Probability of Exclusive-Or. In B. K. Roy and W. Meier, editors, *FSE*, volume 3017 of *Lecture Notes in Computer Science*, pages 317–331. Springer, 2004.

[24] R. F. Lyon. Two's Complement Pipeline Multipliers. *IEEE Transactions on Communications*, 24(4):418–425, April 1976.

[25] R. F. Lyon. A bit-serial architectural methodology for signal processing. In J. P. Gray, editor, *VLSI-81*, pages 131–140. Academic Press, 1981.

[26] M. Matsui and A. Yamagishi. A New Method for Known Plaintext Attack of FEAL Cipher. In *EUROCRYPT*, pages 81–91, 1992.

[27] G. H. Mealy. A method for synthesizing sequential circuits. *Bell Systems Technical Journal*, 34:1045–1079, 1955.

[28] A. Menezes, P. C. van Oorschot, and S. A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1997.

[29] E. F. Moore. Gedanken experiments on sequential machines. *Automata Studies*, pages 129–153, 1956.

[30] N. Mouha, C. De Cannière, S. Indesteege, and B. Preneel. Finding Collisions for a 45-Step Simplified HAS-V. In H. Y. Youm and M. Yung, editors, *WISA*, volume 5932 of *Lecture Notes in Computer Science*, pages 206–225. Springer, 2009.

[31] National Institute of Standards and Technology. Announcing Request for Candidate Algorithm Nominations for a New Cryptographic Hash Algorithm (SHA-3) Family. *Federal Register*, 27(212):62212–62220, November 2007. `http://csrc.nist.gov/groups/ST/hash/documents/FR_Notice_Nov07.pdf`.

[32] R. M. Needham and D. J. Wheeler. TEA extensions. Computer Laboratory, Cambridge University, England, 1997. `http://www.movable-type.co.uk/scripts/xtea.pdf`.

[33] S. O'Neil, K. Nohl, and L. Henzen. EnRUPT Hash Function Specification. Submission to the NIST SHA-3 Competition (Round 1), 2008. `http://enrupt.com/SHA3/Supporting_Documentation/EnRUPT_Specification.pdf`.

[34] B. Schneier, J. Kelsey, D. Whiting, D. Wagner, C. Hall, and N. Ferguson. *The Twofish encryption algorithm: a 128-bit block cipher*. John Wiley & Sons, Inc., New York, NY, USA, 1999.

A Matrices for xdp$^+$

The four distinct matrices $A_{w[i]}$ obtained for xdp$^+$ in Sect. 3.4 are given in (50). The remaining matrices can be derived using $A_{001} = A_{010} = A_{100}$ and $A_{011} =$

$A_{101} = A_{110}$.

$$A_{000} = \begin{bmatrix} 3 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 3 \end{bmatrix}, \qquad A_{001} = \begin{bmatrix} 0 & 1 & 1 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 1 & 1 & 0 \end{bmatrix},$$
$$A_{011} = \begin{bmatrix} 2 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 2 \end{bmatrix}, \qquad A_{111} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 3 & 0 \\ 0 & 3 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}. \tag{50}$$

Similarly, we give the four distinct matrices $A'_{w[i]}$ of Sect. 3.4 in (51). The remaining matrices satisfy $A'_{001} = A'_{010} = A'_{100}$ and $A'_{011} = A'_{101} = A'_{110}$.

$$A'_{000} = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}, \; A'_{001} = \frac{1}{2}\begin{bmatrix} 0 & 1 \\ 0 & 1 \end{bmatrix}, \; A'_{011} = \frac{1}{2}\begin{bmatrix} 1 & 0 \\ 1 & 0 \end{bmatrix}, \; A'_{111} = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}. \tag{51}$$

B All Possible Subgraphs for xdp^+

All possible subgraphs for xdp^+ are given in Fig. 4.

C Computation of xdp^+ with Multiple Inputs.

In Sect. 3, we showed how to compute the probability $\mathrm{xdp}^+(\alpha, \beta \to \gamma)$, by introducing S-functions and using techniques based on graph theory and matrix multiplications. Similarly, we can also evaluate the probability $\mathrm{xdp}^+(\alpha[i], \beta[i], \zeta[i], \ldots \to \gamma[i])$ for multiple inputs. We illustrate this for the simplest case of three inputs. We follow the same basic steps from Sect. 3 and Sect. 4: construct the S-function, construct the graph and derive the matrices, minimize the matrices, and multiply them to compute the probability.

Let us define

$$S[i] \leftarrow (c_1[i], c_2[i]) \ , \tag{52}$$
$$S[i+1] \leftarrow (c_1[i+1], c_2[i+1]) \ . \tag{53}$$

Then, the S-function corresponding to the case of three inputs x, y, q and output z is:

$$(\Delta^{\oplus} z[i], S[i+1]) = f(x_1[i], y_1[i], q_1[i], \Delta^{\oplus} x[i], \Delta^{\oplus} y[i], \Delta^{\oplus} q[i], S[i]). \quad 0 \le i < n \ . \tag{54}$$

Because we are adding three words in binary, the values for the carries $c_1[i]$ and $c_2[i]$ are both in the set $\{0, 1, 2\}$. The differential $(\alpha[i], \beta[i], \zeta[i] \to \gamma[i])$ at bit position i is written as a bit string $w[i] \leftarrow \alpha[i] \parallel \beta[i] \parallel \zeta[i] \parallel \gamma[i]$. Using this S-function and the corresponding graph, we build the matrices $A_{w[i]}$. After we apply

the minimization algorithm (removing inaccessible states and combining equivalent states) we obtain the following minimized matrices. The remaining matrices satisfy $A_{0001} = A_{0010} = A_{0100} = A_{1000}$, $A_{0011} = A_{0101} = A_{0110} = A_{1001} = A_{1010} = A_{1100}$ and $A_{0111} = A_{1011} = A_{1101} = A_{1110}$.

$$A_{0000} = \begin{bmatrix} 4 & 0 & 0 & 2 \\ 0 & 0 & 8 & 0 \\ 0 & 0 & 0 & 0 \\ 4 & 0 & 0 & 6 \end{bmatrix}, \quad A_{0001} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 4 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 \end{bmatrix}, \quad A_{0011} = \begin{bmatrix} 2 & 0 & 0 & 0 \\ 4 & 0 & 4 & 4 \\ 0 & 0 & 2 & 0 \\ 2 & 0 & 2 & 4 \end{bmatrix},$$

$$A_{0111} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 4 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 3 & 0 & 0 \end{bmatrix}, \quad A_{1111} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 8 & 0 & 0 & 0 \\ 0 & 0 & 4 & 2 \\ 0 & 0 & 4 & 6 \end{bmatrix}.$$

D Computation of xdp$^{\times 3}$

Given n-bit words $x_1, \Delta^{\oplus} x$, we can calculate $\Delta^{\oplus} z$ using

$$x_2 \leftarrow x_1 \oplus \Delta^{\oplus} x \ , \tag{55}$$
$$z_1 \leftarrow x_1 \cdot 3 = (x_1 \ll 1) + x_1 \ , \tag{56}$$
$$z_2 \leftarrow x_2 \cdot 3 = (x_2 \ll 1) + x_2 \ , \tag{57}$$
$$\Delta^{\oplus} z \leftarrow z_2 \oplus z_1 \ . \tag{58}$$

We then define $\mathrm{xdp}^{\times 3}(\alpha \to \gamma)$ as

$$\mathrm{xdp}^{\times 3}(\alpha \to \gamma) = \frac{|\{x_1 : \Delta^{\oplus} x = \alpha, \Delta^{\oplus} z = \gamma\}|}{|\{x_1 : \Delta^{\oplus} x = \alpha\}|} \ , \tag{59}$$
$$= 2^{-n} |\{x_1 : \Delta^{\oplus} x = \alpha, \Delta^{\oplus} z = \gamma\}| \ , \tag{60}$$

as there are 2^n values for the n-bit word x_1.

The left shift by one requires one bit of both $x_1[i]$ and $x_2[i]$ to be stored for the calculation of the next output bit. For this, we will use $d_1[i]$ and $d_2[i]$. In general, shifting to the left by i positions requires the i previous inputs to be stored. Therefore, (55)-(58) correspond to the following bit level expressions:

$$x_2[i] \leftarrow x_1[i] \oplus \Delta^{\oplus} x[i] \ , \tag{61}$$
$$z_1[i] \leftarrow x_1[i] \oplus d_1[i] \oplus c_1[i] \ , \tag{62}$$
$$c_1[i+1] \leftarrow (x_1[i] + d_1[i] + c_1[i]) \gg 1 \ , \tag{63}$$
$$d_1[i+1] \leftarrow x_1[i] \ , \tag{64}$$
$$z_2[i] \leftarrow x_2[i] \oplus d_2[i] \oplus c_2[i] \ , \tag{65}$$
$$c_2[i+1] \leftarrow (x_2[i] + d_2[i] + c_2[i]) \gg 1 \ , \tag{66}$$
$$d_2[i+1] \leftarrow x_2[i] \ , \tag{67}$$
$$\Delta^{\oplus} z[i] \leftarrow z_2[i] \oplus z_1[i] \ , \tag{68}$$

where $c_1[0] = c_2[0] = d_1[0] = d_2[0] = 0$. Let us define

$$S[i] \leftarrow (c_1[i], c_2[i], d_1[i], d_2[i]) \ , \tag{69}$$

$$S[i+1] \leftarrow (c_1[i+1], c_2[i+1], d_1[i+1], d_2[i+1]) \ . \tag{70}$$

Then (61)-(68) correspond to the S-function

$$(\Delta^{\oplus} z[i], S[i+1]) = f(x_1[i], \Delta^{\oplus} x[i], S[i]), \qquad 0 \le i < n \ . \tag{71}$$

Each of $c_1[i]$, $c_2[i]$, $d_1[i]$, $d_2[i]$ can be either 0 or 1. After minimizing the 16 states $S[i]$, we obtain only 4 indistinguishable states. Define again 1×4 matrix $L = [\ 1 \quad 1 \quad 1 \quad 1\ ]$ and 4×1 matrix $C = [\ 1 \quad 0 \quad 0 \quad 0\ ]^T$. The differential $(\alpha[i] \rightarrow \gamma[i])$ at bit position i is written as a bit string $w[i] \leftarrow \alpha[i] \parallel \gamma[i]$. Then $\mathrm{xdp}^{\times 3}$ is equal to

$$\mathrm{xdp}^{\times 3}(\alpha \rightarrow \gamma) = L A^*_{w[n-1]} \cdots A^*_{w[1]} A^*_{w[0]} C \ , \tag{72}$$

where

$$A^*_{00} = \frac{1}{2}\begin{bmatrix} 1 & 0 & 2 & 0 \\ 0 & 0 & 0 & 2 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \qquad A^*_{01} = \frac{1}{2}\begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix},$$

$$A^*_{10} = \frac{1}{2}\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}, \qquad A^*_{11} = \frac{1}{2}\begin{bmatrix} 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 2 & 1 \end{bmatrix}. \tag{73}$$

We now illustrate this calculation by example. Let $\alpha =$ `0x12492489` and $\gamma =$ `0x3AEBAEAB`. Then $\mathrm{xdp}^{\times 3}(\alpha \rightarrow \gamma) = 2^{-15}$, whereas $\mathrm{xdp}^{+}(\alpha, \alpha \ll 1 \rightarrow \gamma) = 2^{-25}$. From this example, we see that approximating the probability calculation of multiplication by a constant using xdp^{+}, can give a result that is completely different from the actual probability. This motivates the need for the technique that we present in this section. We note there is no loss in generality when we analyze $\mathrm{xdp}^{\times 3}$: the same technique can be automatically applied for $\mathrm{xdp}^{\times \mathrm{C}}$, where C is an arbitrary constant.

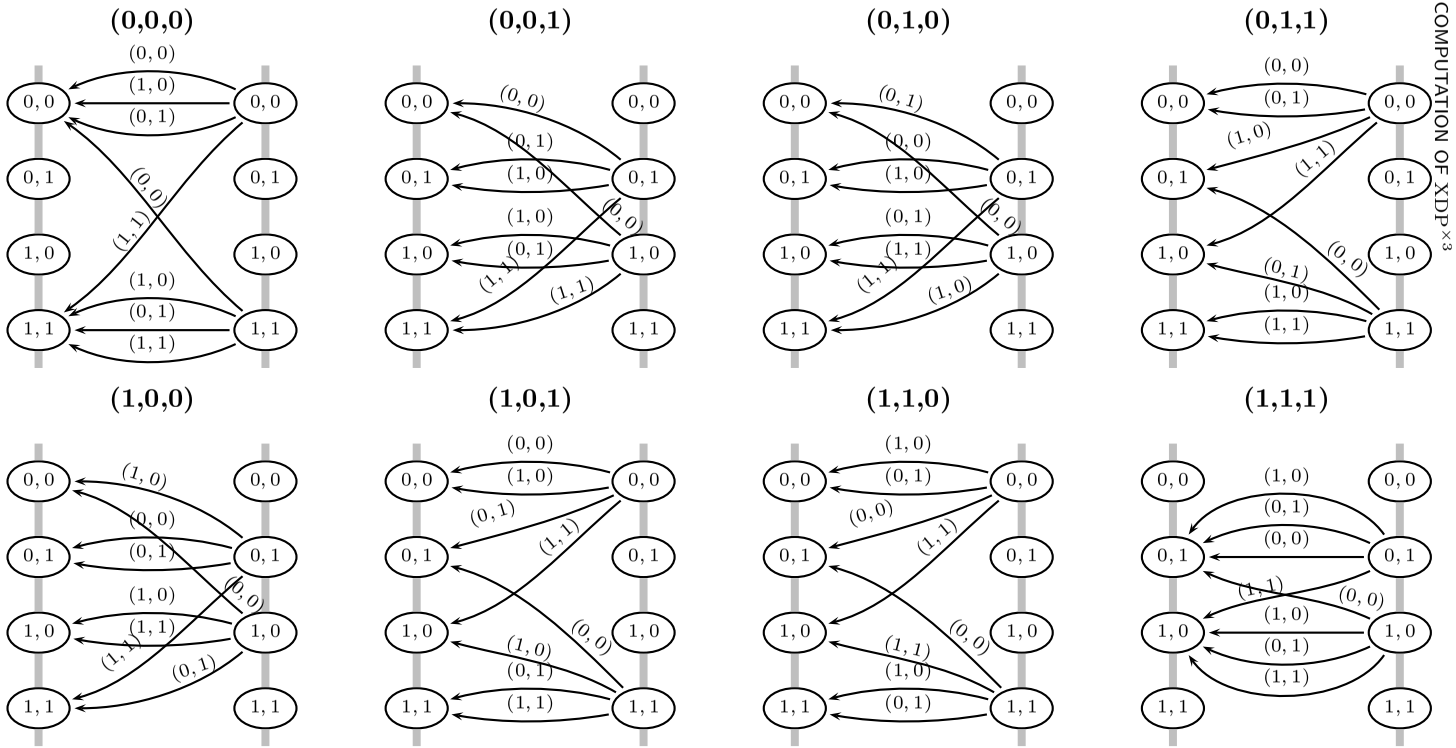


Figure 4 – All possible subgraphs for xdp$^+$. Vertices $(c_1[i], c_2[i])$ correspond to states $S[i]$. There is one edge for every input pair (x_1, y_1). Above each subgraph, the value of $(\alpha[i], \beta[i], \gamma[i])$ is given in bold.

Correction

Contrary to the statement of Sect. 2, the formulas in Definition 2 of [11] clarify that Daum's w-narrow T-function is in fact the same as an S-function. However, Daum uses w-narrow T-functions in a completely different context: to solve systems of equations, and not to calculate differential probabilities. We'd like to point out that our incorrect statement is only relevant to the literature study that we performed. It does not invalidate any of the research results in the paper.

Publication Chapter

Meet-in-the-Middle Attacks on Reduced-Round XTEA

Publication Data

Gautham Sekar, Nicky Mouha, Vesselin Velichkov, and Bart Preneel. Meet-in-the-Middle Attacks on Reduced-Round XTEA. In Aggelos Kiayias, editor, *CT-RSA*, volume 6558 of *Lecture Notes in Computer Science*, pages 250–267. Springer, 2011.

Contributions

- Main author. Devised the attack on 23 rounds, and provided complexity and success probability calculations for all attacks. The introduction and the attacks on 7 and 14 rounds are due to Gautham Sekar.

Meet-in-the-Middle Attacks on Reduced-Round XTEA*

Gautham Sekar†, Nicky Mouha‡, Vesselin Velichkov§, and Bart Preneel

[1] Department of Electrical Engineering ESAT/SCD-COSIC,
Katholieke Universiteit Leuven, Kasteelpark Arenberg 10, B-3001 Heverlee, Belgium.
[2] Interdisciplinary Institute for BroadBand Technology (IBBT), Belgium.
{Gautham.Sekar,Nicky.Mouha,Vesselin.Velichkov,Bart.Preneel}@esat.kuleuven.be

Abstract. The block cipher XTEA, designed by Wheeler and Needham, was published as a technical report in 1997. The cipher was a result of fixing some weaknesses in the cipher TEA (also designed by Wheeler and Needham), which was used in Microsoft's Xbox gaming console. XTEA is a 64-round Feistel cipher with a block size of 64 bits and a key size of 128 bits. In this paper, we present meet-in-the-middle attacks on twelve variants of the XTEA block cipher, where each variant consists of 23 rounds. Two of these require only 18 known plaintexts and a computational effort equivalent to testing about 2^{117} keys, with a success probability of $1 - 2^{-1025}$. Under the standard (single-key) setting, there is no attack reported on 23 or more rounds of XTEA, that requires less time and fewer data than the above. This paper also discusses a variant of the classical meet-in-the-middle approach. All attacks in this paper are applicable to XETA as well, a block cipher that has not undergone public analysis yet. TEA, XTEA and XETA are implemented in the Linux kernel.

Keywords: Cryptanalysis, block cipher, meet-in-the-middle attack, Feistel network, XTEA, XETA.

1 Introduction

Timeline: the TEA family of block ciphers

*This work was supported in part by the Research Council K.U.Leuven: GOA TENSE, and by the IAP Program P6/26 BCRYPT of the Belgian State (Belgian Science Policy), and in part by the European Commission through the ICT program under contract ICT-2007-216676 ECRYPT II.

†This author is supported by an FWO project.

‡This author is funded by a research grant of the Institute for the Promotion of Innovation through Science and Technology in Flanders (IWT-Vlaanderen).

§DBOF Doctoral Fellow, K.U.Leuven, Belgium.

- **1994.** The cipher TEA (Tiny Encryption Algorithm) is a 64-round Feistel cipher that operates on 64-bit blocks and uses a 128-bit key. Designed by Wheeler and Needham, it was presented at FSE 1994 [24]. Noted for its simple design, the cipher was subsequently well studied and came under a number of attacks.

- **1996.** Kelsey *et al.* established that the effective key size of TEA was 126 bits [12]. This result led to an attack on Microsoft's Xbox gaming console where TEA was used as a hash function [23].

- **1997.** Kelsey, Schneier and Wagner constructed a related-key attack on TEA with 2^{23} chosen plaintexts and 2^{32} time [13]. Following these results, TEA was redesigned by Needham and Wheeler to yield Block TEA and XTEA (eXtended TEA) [18]. While XTEA has the same block size, key size and number of rounds as TEA, Block TEA caters to variable block sizes for it applies the XTEA round function for several iterations. Both TEA and XTEA are implemented in the Linux kernel.

- **1998.** To correct weaknesses in Block TEA, Needham and Wheeler designed Corrected Block TEA or XXTEA, and published it in a technical report [19]. This cipher uses an unbalanced Feistel network and operates on variable-length messages. The number of rounds is determined by the block size, but it is at least six. An attack on the full Block TEA is presented in [20], where some weaknesses in XXTEA are also detailed.

- **2002–2010.** A number of cryptanalysis results on the TEA family were reported in this period. Table 1 lists the attacks on XTEA and their complexities. In [11], it was shown that an ultra-low power implementation of XTEA might be better suited for low resource environments than AES. Note that XTEA's smaller block size also makes it advantageous if an application requires fewer than 128 bits of data to be encrypted at a time.

The meet-in-the-middle attack. The meet-in-the-middle attack was first introduced by Diffie and Hellman in 1977 [5]. Since then, this technique and its variants have been successfully used against several block ciphers, including reduced-round DES [4,6] and the full KeeLoq [10]. Unlike Diffie and Hellman's original attack, the meet-in-the-middle attacks in this paper[3] have negligible memory requirements.

We denote the message space and the key space by $\mathcal{M}$ and $\mathcal{K}$ respectively. Now consider two block ciphers $A_K, B_K : \mathcal{M} \times \mathcal{K} \to \mathcal{M}$ and let $Y_K = B_K \circ A_K$, where

[3]The attack presented in Sect. 5 of this paper can also be seen as a meet-in-the-middle attack, however the (partial) encryptions and decryptions cannot be performed over all rounds, as the attacker only searches exhaustively over parts of the key. We therefore use a technique similar to the partial matching technique of Sasaki and Aoki. This very recent technique was successfully applied to several hash functions, including MD4 [2], MD5 [21], HAS-160 [8] and SHA-2 [1].

Table 1 – Key recovery attacks on XTEA where the time complexities are averages, if explicitly stated in the original paper, average success probabilities are given as well (KP: known plaintext, CP: chosen plaintext, RK: in a related-key setting)

Attack	Ref.	# Rnds	Time	Data	Pr[Success]
• Attacks in the standard (single-key) setting					
Meet-in-the-middle	**This paper**	**7**	$\mathbf{2^{95.00}}$	**2 KPs**	$\mathbf{1-2^{-33}}$
Impossible differential	[17]	14	2^{85}	$2^{62.5}$ CPs	Not given
Differential	[9]	15	2^{120}	2^{59} CPs	Not given
Meet-in-the-middle	**This paper**	**15**	$\mathbf{2^{95.00}}$	**3 KPs**	$\mathbf{1-2^{-65}}$
Truncated differential	[9]	23	$2^{120.65}$	$2^{20.55}$ CPs	0.969
Meet-in-the-middle	**This paper**	**23**	$\mathbf{2^{117.00}}$	**18 KPs**	$\mathbf{1-2^{-1025}}$
• Attacks in a related-key setting					
Related-key truncated differential	[14]	27	$2^{115.15}$	$2^{20.5}$ RK-CPs	0.969
Related-key rectangle (for $2^{108.21}$ weak keys)	[15]	34	$2^{31.94}$	2^{62} RK-CPs	Not given
Related-key rectangle	[16]	36	$2^{126.44}$	$2^{64.98}$ RK-CPs	0.63
Related-key rectangle (for $2^{110.67}$ weak keys)	[16]	36	$2^{104.33}$	$2^{63.83}$ RK-CPs	0.80
Related-key	[3]	37	2^{125}	2^{63} RK-CPs	Not given
Related-key (for $2^{107.5}$ weak keys)	[3]	51	2^{123}	2^{63} RK-CPs	Not given

$\circ$ denotes function composition. In a meet-in-the-middle attack, the adversary deduces K from a given plaintext-ciphertext pair (p, c), where $c = Y_K(p)$, by solving the equation

$$A_K(p) = B_K^{-1}(c) \ . \tag{1}$$

Contribution of this paper. This paper presents meet-in-the-middle attacks on block ciphers with 7, 15 and 23 rounds of XTEA. Our attacks are under the standard setting, giving the attacker less freedom than under a related-key setting. In Table 1, we see that there is no attack on 23 or more rounds of XTEA, that is better than ours given the standard setting. Furthermore, each of our attacks requires only a few *known plaintexts*, whereas every attack listed in Table 1 requires many *chosen plaintexts*.

The Linux kernel not only includes XTEA, but also a variant called XETA [7]. The cipher XETA resulted from a bug in the C implementation of XTEA, where

higher precedence was incorrectly given to exclusive-OR over addition in the round function. From this paper, it is easy to verify that all our results to XTEA directly apply to XETA as well. This is because our attacks exploit weaknesses in the key schedule, which is the same for both XTEA and XETA. To the best of our knowledge, this paper is the first to give cryptanalysis results on XETA.

Organization. This paper is organized as follows. Section 2 lists the notation and convention that we follow. The description of XTEA is provided in Sect. 3. Our main observation is presented in Sect. 4 and it is developed into an attack on 15-round XTEA in Sect. 5. Here, we also provide other sets of 15 rounds that could be similarly attacked. Section 6 describes our attack on 23 rounds on XTEA and provides other sets of 23 rounds that could be attacked in a similar way. Section 7 concludes the paper and provides an interesting open problem. In Appendix A, we show which countermeasures can be introduced to XTEA to prevent all the attacks in this paper. The 23-round attack is illustrated in Appendix B.

2 Notation and Convention

The notation used in this paper is listed in Table 2.

Table 2 – Notation

Symbol / Notation	Meaning
$\boxplus$	Addition modulo 2^{32}
$\oplus$	Exclusive-OR
$\ll$	Left shift
$\gg$	Right shift
$\|\|$	Concatenation
$\lfloor x \rfloor$	$\max_{y \in \mathbb{Z}}(y \leq x)$, $\mathbb{Z}$ is the set of integers
LSB	Least significant bit
MSB	Most significant bit
$[i]$	Select bit i, $i = 0$ is the LSB
$[j \dots i]$	Select bits k where $j \geq k \geq i$, $k = 0$ is the LSB
0^k	Concatenation of k times the string '0'

3 Description of XTEA

The block cipher XTEA has block size of 64 bits and key size of 128 bits. It uses a 64-round Feistel network (see Fig. 1). The F-function of the Feistel network (see

Fig. 2) takes a 32-bit input x and produces a 32-bit output as:

$$F(x) = ((x \ll 4) \oplus (x \gg 5)) + x \ . \tag{2}$$

The 128-bit key K of XTEA is divided into four 32-bit subkeys $K_0, \ldots, K_3$. At every round, one of the 4 subkeys is selected according to a key schedule. A constant $\delta = \lfloor (\sqrt{5} - 1) \cdot 2^{31} \rfloor$ is defined, derived from the golden ratio. Two bits from a different multiple of δ are used at every round as the index of the subkey. The 32-bit subkey α_t used in round t, where $1 \leq t \leq 64$, is chosen from the set $\{K_0, K_1, K_2, K_3\}$ according to the following rule:

$$\alpha_t \leftarrow \begin{cases} K_{\delta_t[1\ldots0]} & \text{if } t \text{ is odd} \ , \\ K_{\delta_t[12\ldots11]} & \text{if } t \text{ is even} \ , \end{cases} \tag{3}$$

where

$$\delta_t = \left\lfloor \frac{t}{2} \right\rfloor \delta, \quad 1 \leq t \leq 64 \ . \tag{4}$$

The 64-bit input to round t of XTEA consists of two 32-bit parts L_{t-1} and R_{t-1} (see Fig. 1). For round 1, the plaintext p is used as input: $(L_0 \parallel R_0) \leftarrow p$. The input for round $t+1$ is computed recursively from the input to round t as given by:

$$L_t \leftarrow R_{t-1} \ , \tag{5}$$

$$R_t \leftarrow L_{t-1} \boxplus ((\delta_t \boxplus \alpha_t) \oplus F(R_{t-1})) \ , \tag{6}$$

where α_t is selected according to (3). For reference, we also list the subkeys used in every round in Table 3.

The ciphertext c of XTEA is produced by concatenating the two parts obtained after the 64th round: $c \leftarrow L_{64} \parallel R_{64}$.

Finally, we note that in the description above by *round* we mean a *Feistel round*. This is not to be confused with the term *cycle* used in the original proposal of XTEA [18]. A cycle is equivalent to two Feistel rounds. Therefore XTEA has 64 rounds or 32 cycles.

Table 3 – Subkeys used in XTEA

Rounds	**Subkey used**
1, 8, 9, 10, 17, 18, 20, 25, 30, 33, 40, 41, 49, 50, 57, 60	K_0
3, 6, 11, 16, 19, 26, 27, 28, 35, 36, 38, 43, 46, 48, 51, 58, 59	K_1
4, 5, 13, 14, 21, 24, 29, 34, 37, 44, 45, 53, 54, 56, 61, 64	K_2
2, 7, 12, 15, 22, 23, 31, 32, 39, 42, 47, 52, 55, 62, 63	K_3

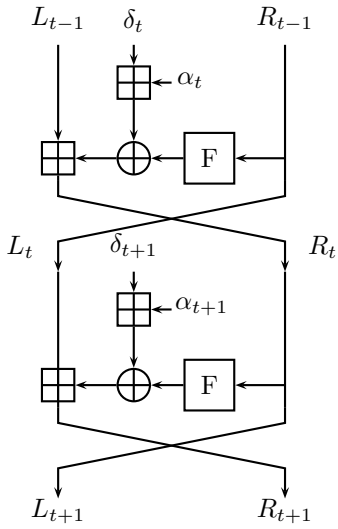


Figure 1 – The Feistel structure of XTEA showing two rounds

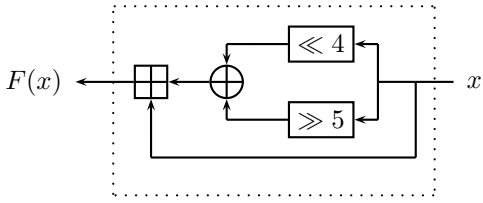


Figure 2 – The function F used in the round function of XTEA

4 Motivational Observation

We begin by observing that the subkey K_2 is not used in rounds 6–12. For the remainder of this section, let $K \leftarrow (K_0, K_1, X, K_3)$, where X can be any 32-bit value, as subkey K_2 is irrelevant in the analysis. Given one plaintext-ciphertext pair (p_0, c_0), with each key guess, the attacker checks whether

$$E_K^{(6\ldots12)}(p_0) = c_0 \ , \tag{7}$$

where $E_K^{(6\ldots12)}$ denotes the 7-round (rounds 6–12) encryption using the key K. At first glance, it may appear that 1 KP is sufficient. However, it is to be noted that the key space (2^{96} keys K) is larger than the ciphertext space (2^{64} ciphertext blocks).

We now show that obtaining a second KP (p_1, c_1) is sufficient for an attack with an average time complexity of $2^{95.00}$ 7-round encryptions and an average success probability of $1 - 2^{-33}$. The attacker iterates over the 2^k keys K, where $k = 96$. For every candidate key K, (7) is tested using the first KP. If this equality is satisfied, the second KP is used to check

$$E_K^{(6...12)}(p_1) = c_1 \ . \tag{8}$$

If either (7) or (8) is not satisfied, the candidate key K is incorrect and can be *sieved*. The approximate number of plaintext-ciphertext pairs that are needed can also be estimated from Shannon's unicity distance [22].

We make the reasonable assumption throughout this paper, that the 7-, 15- and 23-round block ciphers that we consider have perfect confusion and diffusion properties [22]. If either the plaintext or the key, or both are changed, it is assumed that the corresponding ciphertext will be generated uniformly at random, independent from previously obtained ciphertexts.

Under this assumption, each of the 64-bit conditions that result from (7) and (8) is satisfied with probability 2^{-64}. All time complexities are stated as the number of equivalent encryptions of the reduced-round block cipher.

The average success probability can be calculated as follows. The two 64-bit conditions are simultaneously satisfied with probability $2^{-2 \cdot 64} = 2^{-128}$. We can therefore eliminate a wrong key with probability $1 - 2^{-128}$. Assume that key i is the correct key, where $0 \leq i < 2^k$. It will be output by the algorithm if all previous keys are eliminated. This happens with probability $(1 - 2^{-128})^i$. The correct key can be located anywhere among the list of 2^k candidate keys with equal probability. Therefore, the average success probability is

$$2^{-k} \cdot \sum_{i=0}^{2^k-1} (1 - 2^{-128})^i = 2^{128-k} \cdot (1 - (1 - 2^{-128})^{2^k}) \approx 2^{128-k} \cdot (1 - e^{-2^{k-128}})$$
$$\approx 1 - 2^{-33} \ . \tag{9}$$

The approximations result from using the first and the second order Taylor approximations of e^x around 0. We now calculate the time complexity of the attack. For a candidate key K to be determined as wrong, the expected number of trials is $1 + 2^{-64}$. This is because for every key, (7) is always checked, and for 2^{-64} keys (8) is checked as well. If the candidate key is correct, two encryptions are always performed. As the correct key can be located anywhere in the list of 2^k candidates keys with equal probability, the average number of encryptions of the algorithm is

$$2^{-k} \cdot \sum_{i=0}^{2^k-1} \left(i \cdot (1 + 2^{-64}) + 2\right) = 2^{-1} \cdot (1 + 2^{-64}) \cdot (2^k - 1) + 2 \approx 2^{95.00} \ . \tag{10}$$

From Table 3, we obtain several other 7-round block ciphers that can be attacked in a similar way. Table 4 lists all such ciphers. Finally, we note that for

Table 4 – All 7-round attacks; each attack requires 2 KPs and on average $2^{95.00}$ 7-round encryptions for an average success probability of $1 - 2^{-33}$

Cipher consisting of XTEA rounds	Unused subkey
6–12	K_2
24–30	K_3
42–48	K_0
46–52	K_2

$n = 0$ and $n = 1$ respectively, one can replace both (7) and (8) with

$$E_K^{(6\ldots r-1)}(p_n) = D_K^{(r\ldots 12)}(c_n) \ , \tag{11}$$

where $r \in \{6, \ldots, 12\}$, $E_K^{(6\ldots 5)}(p_n) = p_n$, and $D_K^{(r\ldots 12)}$ denotes (13-r)-round (rounds r–12) decryption using the key K.[4] Therefore, what we essentially constructed above can be viewed as meet-in-the-middle attacks. In (11), the value of r determines the subkeys that are required for encryption and decryption.

5 Attacks on 15 Rounds of XTEA

The attack described in Sect. 4 on rounds 6–12, can be extended to rounds 6–20 as follows. First, the attacker performs a meet-in-the-middle attack, where (partial) encryptions and decryptions cannot be performed over all rounds, the attacker only exhaustively searches over part of the key. From the remaining rounds, however, the number of possibilities for the full key is reduced. Only three known plaintexts (p_n, c_n), $0 \leq n < 2$ are required for the attack.

Let us now split a reduced-round XTEA block cipher into *outer rounds* and *inner rounds*. In the outer rounds, one particular subkey is not used, whereas the inner rounds use only this subkey. The attack is described for rounds 6–20. As can be seen from Table 3, the outer rounds (6–12) and (15–20) do not involve K_2, whereas the two inner rounds (13–14) use only K_2.

By encrypting plaintext p_0 from round 6 to round 12 (i.e., until the beginning of round 13) and decrypting the corresponding ciphertext c_0 for 6 rounds starting backwards from round 20, we obtain the subkeys used in the inner rounds. They are denoted as $K_2^{'}$ and $K_2^{''}$ for inner rounds 13 and 14 respectively. Then, the attacker checks whether $K_2^{'} = K_2^{''}$. This can be understood from Fig. 1. Therefore, not the ciphertext values (as in Sect. 4), but the key values "meet in the middle".

[4] One may interpret (11) as encryptions and decryptions with block ciphers of fewer than seven rounds. Given the assumption that a 7-round block cipher has perfect confusion and diffusion, separate assumptions are not required for these further reduced block ciphers.

To the best of our knowledge, such an approach has not been described in previous literature.

If $K_2^{'} \neq K_2^{''}$, the candidate key of (K_0, K_1, K_3) cannot be correct, and the attacker proceeds to the next candidate key. Otherwise, the candidate key is extended to (K_0, K_1, K_2, K_3), where $K_2 = K_2^{'} = K_2^{''}$. Then, the meet-in-the-middle attack is performed as described in Sect. 4. That is, a plaintext is encrypted with candidate keys (K_0, K_1, K_2, K_3), to check which of the computed ciphertexts agrees with the actual (corresponding) ciphertext. For the 15-round attack, it is sufficient to use two additional known plaintexts (p_1, c_1) and (p_2, c_2).

The average success probability can be calculated as follows. Using (p_0, c_0) a 32-bit condition is obtained when $K_2^{'} = K_2^{''}$ is checked. Then, (p_1, c_1) and (p_2, c_2) each gives an additional 64-bit condition. A wrong key will pass these tests with probability[5] $2^{-32} \cdot \left(2^{-64}\right)^2 = 2^{-160}$. Thus, with probability $1 - 2^{-160}$, a wrong key is eliminated. Assume that i is the correct key, where $0 \leq i < 2^k$. It will be output by the algorithm if all previous keys are eliminated. This happens with probability $(1 - 2^{-160})^i$. The correct key can be located anywhere among the list of 2^k candidate keys with equal probability. The average success probability is

$$2^{-96} \cdot \sum_{i=0}^{2^{96}-1} (1 - 2^{-160})^i = 2^{160-96} \cdot (1 - (1 - 2^{-160})^{2^{96}}) \approx 2^{64} \cdot (1 - e^{-2^{64}})$$
$$\approx 1 - 2^{-65} \quad . \tag{12}$$

We now calculate the time complexity of the attack. For a candidate key (K_0, K_1, K_3) to be determined as wrong, the expected number of trials is $1 + 2^{-32} + 2^{-96}$. This is because for every candidate key (K_0, K_1, K_3), the attacker always checks whether $K_2^{'} \neq K_2^{''}$. For 2^{-32} and 2^{-96} candidate keys, the attacker encrypts using the second and third known plaintext respectively. If the candidate key is correct, the equivalent of three encryptions is always performed. As the correct key can be located anywhere in the list of 2^{96} candidates keys with equal probability, the average number of (equivalent) encryptions of the algorithm is

$$2^{-96} \cdot \sum_{i=0}^{2^{96}-1} \left(i \cdot (1 + 2^{-32} + 2^{-96}) + 3\right) = 2^{-1} \cdot (1 + 2^{-32} + 2^{-96}) \cdot (2^{96} - 1) + 3$$
$$\approx 2^{95.00} \quad . \tag{13}$$

Finally, in Table 5, we provide a list of all 15-round block ciphers that can be attacked with the same complexity.

[5]If the texts obtained by encrypting p_0 and decrypting c_0, in the 13 outer rounds, are uniformly distributed at random, then so are the subkeys $K_2^{'}$ and $K_2^{''}$. This fact, explained in Appendix C, is explicitly stated here because the assumption of perfect confusion and diffusion was made for ciphertexts, and not for subkeys.

Table 5 – All 15-round attacks; each attack requires 3 KPs and on average $2^{95.00}$ computations of the 15 rounds for an average success probability of $1 - 2^{-65}$

Cipher consisting of XTEA rounds	Inner rounds	Inner round subkey
6–20	13,14	K_2
16–30	22,23	K_3
24–38	31,32	K_3
34–48	40,41	K_0
38–52	44,45	K_2
42–56	49,50	K_0

6 Attacks on 23 Rounds of XTEA

In this section, we extend the 15-round attack of Sect. 5 to 23 rounds. This 23-round attack has an average time complexity of $2^{117.00}$ (equivalent) encryptions and an average success probability of $1 - 2^{-1025}$. It requires only 18 known (not chosen) plaintexts and corresponding ciphertexts. For the same number of rounds, both the time complexity and the data complexity of our attack are much lower than those in [9]. Our attack is therefore the best attack on 23-round XTEA so far in the standard setting, and the only attack requiring such a low number of plaintexts and corresponding ciphertexts. We note that we have optimized our attack to have the time complexity as low as possible. It is possible to reduce the number of known plaintexts even further, but not without increasing the time complexity of the attack.

The technique used is a meet-in-the-middle attack, similar to the attacks in [4]. As in Sect. 5, the reduced-round XTEA block cipher is split into *outer rounds* and *inner rounds*. In the outer rounds, one subkey is not used. The inner rounds can contain any of the subkeys. Our attack applies to rounds 16–38 of XTEA. Rounds 16–21 and 33–38 are the outer rounds, and do not involve subkey K_3. The inner rounds are rounds 22–32. The attack is a *sieving attack*, as the correct key is found by eliminating keys that lead to contradictions. The attack is given in Algorithm 1.

The k-bit key is recovered in two stages. First, the attacker exhaustively searches over k_1 bits of the key K and use m known plaintexts to check a one-bit condition that each of the m plaintexts yield. These k_1 bits consist of K_0, K_1, K_2, and the 21 least significant bits of K_3. This one-bit condition, tested in `test_keys_1(K)`, results from the following observation, also illustrated in Appendix B. We see that, without using $K_3[31 \ldots 21]$, the attacker can calculate $L_{27}[0] \leftarrow E_K^{(16 \ldots 27)}(p)[0]$, and $L'_{27}[0] \leftarrow D_K^{(28 \ldots 38)}(c)[0]$. As $L_{27}[0] = L'_{27}[0]$ always holds if the candidate key K is correct, a wrong key can be discarded if $L_{27}[0] \neq L'_{27}[0]$. Note that only k_1 bits of the candidate key K are used to test

this condition, as the remaining k_2 bits do not affect this condition.

If none of the m plaintexts cause a key to be discarded, the attacker exhaustively searches over the remaining k_2 bits of key K in `test_keys_2(K)`. These k_2 bits are the 11 most significant bits of K_3. In this stage, $\ell \leq m$ of the m plaintexts are reused. Now, $(L_{27}, R_{27}) \leftarrow E_K^{(16\ldots27)}(p)$ and $(L'_{27}, R'_{27}) \leftarrow D_K^{(28\ldots38)}(c)$ are recalculated using the full key K. For efficiency, this calculation is sped up by using stored values $p_n^\star$ and $c_n^\star$ for the outer rounds, and encrypting only the inner rounds. Equations $L_{27} = R_{27}$ and $L'_{27} = R'_{27}$ yield only 63-bit conditions, as $L_{27}[0] = L'_{27}[0]$ was already tested. If both equations are satisfied for all ℓ plaintexts, the candidate key K is output as the correct key, and the algorithm halts.

Let us now determine the average time complexity and the average success probability of Algorithm 1.

The algorithm succeeds if no wrong key K that passes all $m + \ell$ tests is encountered before the correct key. How efficiently the attacker searches through these candidate keys K, does not influence the success probability of Algorithm 1. We therefore assume that the exhaustive search is over 2^k keys, and then both `test_keys_1(K)` and `test_keys_2(K)` are performed for each of these keys.

Each of the m plaintexts yields a one-bit condition in `test_keys_1(K)`, satisfied randomly with a probability of 2^{-1}. When $\ell \leq m$ of these plaintexts are reused in `test_keys_2(K)`, there is a condition on the 63 remaining bits, satisfied randomly with a probability of 2^{-63}. A wrong key will be detected if at least one of the $m+\ell$ tests fail. This eliminates a wrong key with a probability of $1-2^{-m}\cdot 2^{-63\ell}$. Assume that i is the correct key, where $0 \leq i < 2^k$. Then, it will be output by the algorithm if all previous candidate keys lead to contradictions. This happens with probability $(1 - 2^{-m} \cdot 2^{-63\ell})^i$. As the correct key can be located anywhere in the list of 2^k candidate keys with equal probability, the average success probability of the algorithm is

$$2^{-k} \cdot \sum_{i=0}^{2^k-1} (1 - 2^{-m} \cdot 2^{-63\ell})^i = 2^{m+63\ell-k} \cdot (1 - (1 - 2^{-m-63\ell})^{2^k}) \approx 2^{m+63\ell-k} \cdot (1 - e^{-2^{k-m-63\ell}}) \ . \tag{14}$$

We now calculate the time complexity of the attack. Let i and j (where $0 \leq i < 2^{k_1}$ and $0 \leq j < 2^{k_2}$) be parts of the correct key K^c where $i = (K_0^c, K_1^c, K_2^c, K_3^c[20\ldots0])$ and $j = K_3^c[31\ldots21]$. Any 117-bit key $(K_0, K_1, K_2, K_3[20\ldots0])$, tested in `test_keys_1(K)` before the correct key is encountered, passes `test_keys_1(K)` with probability 2^{-m}. Therefore, of the i 117-bit keys tested before the correct key, $i \cdot 2^{-m}$ keys are expected to pass `test_keys_1(K)`. For each of these $i \cdot 2^{-m}$ keys, `test_keys_2()` is performed 2^{k_2} times. Summarizing,

- the attacker performs an expected $i \cdot T_1$ 23-round computations, where T_1 is the expected number of 23-round computations for a wrong key under

`test_keys_1()`;

- the attacker additionally performs an expected $i \cdot 2^{-m} \cdot 2^{k_2} \cdot T_2$ 23-round computations, where T_2 is the expected number of 23-round computations for a wrong key under `test_keys_2()`.

It is easy to see that

$$T_1 \triangleq \sum_{i=0}^{m-1} 2^{-i} \quad . \tag{15}$$

To compute T_2, note that `test_keys_2()` only encrypts the 11 inner rounds again, and uses stored values for (partial) encryptions and decryptions of the outer rounds. This is equivalent to 11/23 encryptions of the 23-round block cipher and therefore

$$T_2 \triangleq \frac{11}{23} \cdot \sum_{j=0}^{\ell-1} 2^{-63j} \quad . \tag{16}$$

For the correct (partial) key i, the number of steps under `test_keys_1()` is m. To determine the remaining part of the correct 128-bit key K^c, the attacker performs an expected $j \cdot T_2 + (11/23) \cdot \ell$ 23-round computations, where

1. $j \cdot T_2$ is the expected number of 23-round computations, under `test_keys_2()`, for all the j wrong (partial) keys preceding key j;
2. ℓ is the number of 11-round steps under `test_keys_2()` for the correct key j.

As the correct key j can take any value in the set $\{0, \ldots, 2^{k_2} - 1\}$, the average number of 23-round computations corresponding to the correct key i, is

$$2^{-k_2} \cdot \sum_{j=0}^{2^{k_2}-1} \left(j \cdot T_2 + \frac{11}{23} \cdot \ell \right) \quad . \tag{17}$$

As the correct key i can take any value in the set $\{0, \ldots, 2^{k_1} - 1\}$, the average number of 23-round computations in total is

$$2^{-k_1} \cdot \sum_{i=0}^{2^{k_1}-1} \left(i \cdot T_1 + m + i \cdot 2^{-m} \cdot 2^{k_2} \cdot T_2 + 2^{-k_2} \cdot \sum_{j=0}^{2^{k_2}-1} \left(j \cdot T_2 + \frac{11}{23} \cdot \ell \right) \right) \tag{18}$$

The derivation of (18) will be more clear from Fig. 3 in Appendix B.

Now, we choose the parameters m and ℓ for the attack on rounds 16–38. From (18), we find that we cannot lower the average time complexity below $2^{117.00}$. Therefore, we choose m and ℓ such that we have the lowest number of known plaintexts, and the highest success probability for this particular time complexity.

Setting $m = \ell = 18$, we find that 18 KPs are sufficient, and that the corresponding success probability using (14) is $1 - 2^{-1025}$. Note that the success probability of exhaustive search over the full k-bit key using 18 KPs has the same success probability. This shows that all KPs are optimally used in our attack from an information theoretic point of view [22]. Note that the number of KPs can still be lowered further, but then the time complexity must increase. This can be done by either increasing ℓ (which would make the second stage dominate in the attack), or by increasing k_1 (and thus perform the meet-in-the-middle on more than one bit).[6] We do not consider such options, as the number of KPs in our attack is already low enough for a practical attack. The time complexity, however, is still beyond reach with current hardware. Each of these attacks requires only negligible memory (about $4 \cdot 64 \cdot 18 = 2^{12.17}$ bits to store (p_n, c_n) and $(p_n^\star, c_n^\star)$ values).

As shown in Table 6, a total of 12 variants of the XTEA block cipher can be attacked, where each variant consists of 23 rounds. For rounds 34–56, the attack works in exactly the same way as for 16–38, and has the same complexities. The 10 other attacks require that $k_1 = 122$: the exhaustive search is now over all but the 6 most significant bits of one subkey in Algorithm 1, in order to obtain a condition on one bit to perform the meet-in-the-middle. The *middle bit* involved in this condition is given as well in Table 6.

Using (18), we calculate the time complexity for the 10 attacks that use 12 or 13 inner rounds. The lowest possible average time complexity for our attack strategy is $2^{122.00}$. For this time complexity, the best parameters are $m = \ell = 13$. We then obtain an average success probability of $1 - 2^{-705}$, using 13 KPs. Again, each of these attacks requires only negligible memory (about $2^{11.70}$ bits to store (p_n, c_n) and $(p_n^\star, c_n^\star)$ values).

7 Conclusions and Open Problems

This paper presented several meet-in-the-middle attacks on 7-, 15- and 23-round XTEA. The main highlight of our attacks is that they require very few known plaintexts (not more than 18) as opposed to previously reported attacks (the best of these attacks requires 2^{20} chosen plaintexts). Furthermore, our attacks use different approaches - the 7- and 23-round attacks use a straightforward meet-in-the-middle approach; in the 15-round attacks, the meet-in-the-middle corresponds to inner round subkeys rather than intermediary text values.

Each of our attacks on 23-round XTEA requires less time ($2^{117.00}$ 23-round computations) than the previously best-known attack on 23 rounds ($2^{120.65}$ 23-round computations) in the standard setting. The time complexities of the 7- and 15-round attacks are also significantly better than exhaustive key search, with each of these attacks requiring about 2^{95} time.

[6] In the attack, one bit in the middle is independent of 11 key bits. Two bits in the middle are simultaneously independent of fewer than 11 key bits, thereby corresponding to a larger k_1.

Table 6 – All 23-round attacks

Total rnds.	Inner rnds.	Middle bit	Unused key bits	# Inner rnds.
16–38	22–32	$L_{27}[0]$	$K_3[31 \dots 21]$	11 rounds
34–56	40–50	$L_{45}[0]$	$K_0[31 \dots 21]$	11 rounds
6–28	13–24	$L_{19}[0]$	$K_2[31 \dots 26]$	12 rounds
8–30	12–23	$L_{18}[0]$	$K_3[31 \dots 26]$	12 rounds
24–46	31–42	$L_{37}[0]$	$K_3[31 \dots 26]$	12 rounds
26–48	30–41	$L_{36}[0]$	$K_0[31 \dots 26]$	12 rounds
30–52	34–45	$L_{40}[0]$	$K_2[31 \dots 26]$	12 rounds
42–64	49–60	$L_{55}[0]$	$K_0[31 \dots 26]$	12 rounds
20–42	26–38	$L_{32}[0]$	$K_1[31 \dots 26]$	13 rounds
38–60	44–56	$L_{50}[0]$	$K_2[31 \dots 26]$	13 rounds
2–24	8–20	$L_{14}[0]$	$K_0[31 \dots 26]$	13 rounds
12–34	16–28	$L_{22}[0]$	$K_1[31 \dots 26]$	13 rounds

Our attacks apply to XETA as well, a close variant of XTEA that is also implemented in the Linux kernel. We are unaware of any other published cryptanalysis results on XETA.

An interesting observation from one of the anonymous reviewers, is that there is also a 15-round attack on rounds 2–16. In this case, subkey K_0 is used consecutively in the inner rounds 8, 9 and 10, but not elsewhere. By exhaustively searching over K_1, K_2, K_3 and six of the least significant bits of K_0, we can perform the same meet-in-the-middle attack that is described in Sect. 6. However, this attack has a higher time and data complexity than the other 15-round attacks of Sect. 5, for a comparable success probability.

When constructing the 23-round attack in Sect. 6, we found that for any number of *inner rounds* (where all subkeys can be used) up to 16, there is no corresponding attack on more than 23 rounds. However, if the number of inner rounds can be increased to 17, this leads to a 29-round attack. All such 29-round attacks are listed in Table 7. We present the cryptanalysis of these 29-round XTEA block ciphers as an interesting open problem.

Acknowledgments. The authors would like to thank Tor E. Bjørstad, Gaëtan Leurent, Matt Robshaw and Aleksander Wittlin for their useful comments and suggestions. Part of this work was performed at the Cryptanalysis of Light-Weight Ciphers Research Meeting, hosted by Katholieke Universiteit Leuven as an initiative of SymLab-WG2: Lightweight Cryptography of the ECRYPT II project. The authors would like to thank the anonymous reviewers for their constructive comments as well.

Table 7 – All reduced-round XTEA block ciphers for which a 29-round attack consists of 17 *inner rounds*

Total rounds	Inner rounds	Subkey containing unused key bits
11–39	27–33	K_0
15–43	21–37	K_2
29–57	35–51	K_1
33–61	40–56	K_3

References

[1] K. Aoki, J. Guo, K. Matusiewicz, Y. Sasaki, and L. Wang. Preimages for Step-Reduced SHA-2. In M. Matsui, editor, *ASIACRYPT*, volume 5912 of *Lecture Notes in Computer Science*, pages 578–597. Springer, 2009.

[2] K. Aoki and Y. Sasaki. Preimage Attacks on One-Block MD4, 63-Step MD5 and More. In R. M. Avanzi, L. Keliher, and F. Sica, editors, *Selected Areas in Cryptography*, volume 5381 of *Lecture Notes in Computer Science*, pages 103–119. Springer, 2008.

[3] C. Bouillaguet, O. Dunkelman, G. Leurent, and P.-A. Fouque. Another Look at Complementation Properties. In S. Hong and T. Iwata, editors, *FSE*, volume 6147 of *Lecture Notes in Computer Science*, pages 347–364. Springer, 2010.

[4] D. Chaum and J.-H. Evertse. Crytanalysis of DES with a Reduced Number of Rounds: Sequences of Linear Factors in Block Ciphers. In H. C. Williams, editor, *CRYPTO*, volume 218 of *Lecture Notes in Computer Science*, pages 192–211. Springer, 1985.

[5] W. Diffie and M. E. Hellman. Exhaustive Cryptanalysis of the NBS Data Encryption Standard. *Computer*, 10(6):74–84, 1977.

[6] O. Dunkelman, G. Sekar, and B. Preneel. Improved Meet-in-the-Middle Attacks on Reduced-Round DES. In K. Srinathan, C. P. Rangan, and M. Yung, editors, *INDOCRYPT*, volume 4859 of *Lecture Notes in Computer Science*, pages 86–100. Springer, 2007.

[7] A. Grothe. Kernel v2.6.14 tea.c. Linux Headquarters, 2004. `http://www.linuxhq.com/kernel/v2.6/14/crypto/tea.c`.

[8] D. Hong, B. Koo, and Y. Sasaki. Improved Preimage Attack for 68-Step HAS-160. In D. Lee and S. Hong, editors, *ICISC*, volume 5984 of *Lecture Notes in Computer Science*, pages 332–348. Springer, 2009.

[9] S. Hong, D. Hong, Y. Ko, D. Chang, W. Lee, and S. Lee. Differential Cryptanalysis of TEA and XTEA. In J. I. Lim and D. H. Lee, editors, *ICISC*, volume 2971 of *Lecture Notes in Computer Science*, pages 402–417. Springer, 2003.

[10] S. Indesteege, N. Keller, O. Dunkelman, E. Biham, and B. Preneel. A Practical Attack on KeeLoq. In N. P. Smart, editor, *EUROCRYPT*, volume 4965 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2008.

[11] J.-P. Kaps. Chai-Tea, Cryptographic Hardware Implementations of xTEA. In D. R. Chowdhury, V. Rijmen, and A. Das, editors, *INDOCRYPT*, volume 5365 of *Lecture Notes in Computer Science*, pages 363–375. Springer, 2008.

[12] J. Kelsey, B. Schneier, and D. Wagner. Key-Schedule Cryptoanalysis of IDEA, G-DES, GOST, SAFER, and Triple-DES. In N. Koblitz, editor, *CRYPTO*, volume 1109 of *Lecture Notes in Computer Science*, pages 237–251. Springer, 1996.

[13] J. Kelsey, B. Schneier, and D. Wagner. Related-key cryptanalysis of 3-WAY, Biham-DES, CAST, DES-X, NewDES, RC2, and TEA. In Y. Han, T. Okamoto, and S. Qing, editors, *ICICS*, volume 1334 of *Lecture Notes in Computer Science*, pages 233–246. Springer, 1997.

[14] Y. Ko, S. Hong, W. Lee, S. Lee, and J.-S. Kang. Related Key Differential Attacks on 27 Rounds of XTEA and Full-Round GOST. In B. K. Roy and W. Meier, editors, *FSE*, volume 3017 of *Lecture Notes in Computer Science*, pages 299–316. Springer, 2004.

[15] E. Lee, D. Hong, D. Chang, S. Hong, and J. Lim. A Weak Key Class of XTEA for a Related-Key Rectangle Attack. In P. Q. Nguyen, editor, *VIETCRYPT*, volume 4341 of *Lecture Notes in Computer Science*, pages 286–297. Springer, 2006.

[16] J. Lu. Related-key rectangle attack on 36 rounds of the XTEA block cipher. *Int. J. Inf. Sec.*, 8(1):1–11, 2009.

[17] D. Moon, K. Hwang, W. Lee, S. Lee, and J. Lim. Impossible Differential Cryptanalysis of Reduced Round XTEA and TEA. In J. Daemen and V. Rijmen, editors, *FSE*, volume 2365 of *Lecture Notes in Computer Science*, pages 49–60. Springer, 2002.

[18] R. M. Needham and D. J. Wheeler. TEA extensions. Computer Laboratory, Cambridge University, England, 1997. `http://www.movable-type.co.uk/scripts/xtea.pdf`.

[19] R. M. Needham and D. J. Wheeler. Correction to xtea. Computer Laboratory, Cambridge University, England, 1998. `http://www.movable-type.co.uk/scripts/xxtea.pdf`.

[20] M.-J. Saarinen. Cryptanalysis of Block TEA. unpublished manuscript, October 1998. http://groups.google.com/group/sci.crypt.research/msg/f52a533d1e2fa15e.

[21] Y. Sasaki and K. Aoki. Finding Preimages in Full MD5 Faster Than Exhaustive Search. In A. Joux, editor, *EUROCRYPT*, volume 5479 of *Lecture Notes in Computer Science*, pages 134–152. Springer, 2009.

[22] C. E. Shannon. Communication Theory of Secrecy Systems. *Bell System Technical Journal*, 28:656–715, 1949.

[23] M. Steil. 17 Mistakes Microsoft Made in the Xbox Security System. 22nd Chaos Communication Congress, December 2005. http://events.ccc.de/congress/2005/fahrplan/events/559.en.html.

[24] D. J. Wheeler and R. M. Needham. TEA, a Tiny Encryption Algorithm. In B. Preneel, editor, *FSE*, volume 1008 of *Lecture Notes in Computer Science*, pages 363–366. Springer, 1994.

A Countermeasures

The attacks in this paper are made possible because a particular subkey K_i is often not used for a large number of rounds. To prevent against the attacks in this paper, we propose to use each of the subkeys K_0, K_1, K_2, K_3 once every four rounds, in a random order. This countermeasure does not prevent trivial meet-in-the-middle attacks on 6 rounds. Note that the subkeys cannot repeat in a cyclic manner, as we want to avoid the possibility of slide attacks.

B Illustration of the Attack on Rounds 16–38

In Fig. 4, we illustrate the 23-round attack of Sect. 6. The attack is on rounds 16–38, and uses 11 inner rounds (22–32). Grey boxes represent bits that do not depend on the value of $K_3[31 \dots 21]$. In Fig. 3, we illustrate Algorithm 1 from the point of view of computation of its time complexity.

C Randomness of the Inner-Round Subkeys in the 15-Round Attacks

Here, we show that if the texts obtained by encrypting p_0 and decrypting c_0 in the 13 outer rounds (of a 15-round attack) are uniformly distributed at random, then so are the subkeys in the inner rounds. As there are only two inner rounds, the problem may be viewed as follows. In Fig. 1, if $L_{t-1}||R_{t-1}$ and $L_{t+1}||R_{t+1}$ are uniformly distributed at random, then we need to show that α_t and α_{t+1} are also

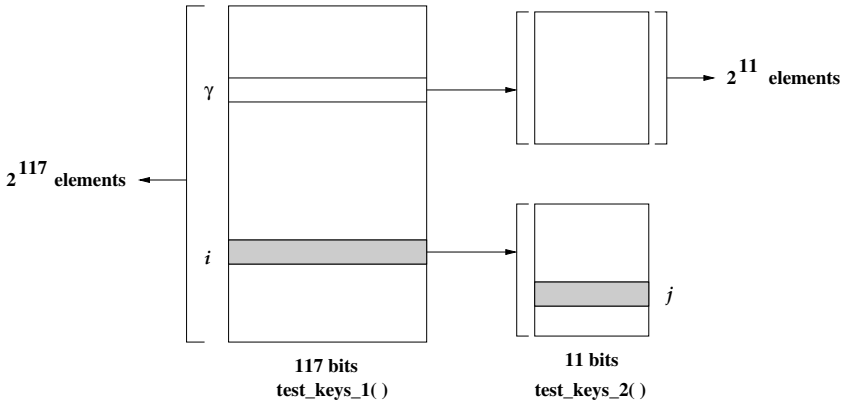


Figure 3 – Attack on rounds 16–38 using Algorithm 1: the tables (not stored in memory) denote the two stages of Algorithm 1 and the shaded 128 bits denote the correct 128-bit key; for a wrong key γ, `test_keys_2()` is performed 2^{11} times

uniformly distributed at random. Henceforth, the term *random* means *uniformly distributed at random*.

Since F is a bijection, the output of F is random given R_{t-1} is random. We know that modular addition (or subtraction) or exclusive-OR of two random values results in a random value. Given this, since $R_t = L_{t+1}$ and $L_{t+1}||L_{t-1}$ is random, from Fig. 1 we obtain that $\delta_t \boxplus \alpha_t$ is random. As δ_t is a constant, α_t is random. By similar arguments, it is easily seen that α_{t+1} is also random.

Algorithm 1 Recovering the key of the 23-round XTEA block cipher consisting of rounds 16–38; an average $2^{117.00}$ (equivalent) encryptions and 18 KPs are required for an average success probability of $1 - 2^{-1025}$

Require: m known plaintexts $p_0 \dots p_{m-1}$ and corresponding ciphertexts $c_0 \dots c_{m-1}$.

Ensure: The output key K (of length k bits) is the correct key with probability $2^{m+63\ell-k}(1 - e^{-2^{k-m-63\ell}})$, where ℓ is chosen such that $\ell \leq m$.

global $p^\star_0 \dots p^\star_{m-1}$, $c^\star_0 \dots c^\star_{m-1}$.
function test_key_1(K) **do**
  for $n \leftarrow 0 \dots m-1$ **do**
    $p^\star_n \leftarrow E_K^{(16\dots21)}(p_n)$
    $c^\star_n \leftarrow D_K^{(33\dots38)}(c_n)$
    $(L_{27}, R_{27}) \leftarrow E_K^{(22\dots27)}(p^\star_n)$
    $(L'_{27}, R'_{27}) \leftarrow D_K^{(28\dots32)}(c^\star_n)$
    if $L_{27}[0] \neq L'_{27}[0]$ **then**
      return *false*
  return *true*
function test_key_2(K) **do**
  for $n \leftarrow 0 \dots \ell-1$ **do**
    $(L_{27}, R_{27}) \leftarrow E_K^{(22\dots27)}(p^\star_n)$
    $(L'_{27}, R'_{27}) \leftarrow D_K^{(28\dots32)}(c^\star_n)$
    if $L_{27} \neq L'_{27}$ **or** $R_{27} \neq R'_{27}$ **then**
      return *false*
  return *true*
for $(K_0, K_1, K_2) \leftarrow (0 \dots 2^{32}-1, 0 \dots 2^{32}-1, 0 \dots 2^{32}-1)$ **do**
  for $K_3[20 \dots 0] \leftarrow 0 \dots 2^{21}-1$ **do**
    $K \leftarrow (K_0, K_1, K_2, 0^{11} \parallel K_3[20 \dots 0])^\dagger$
    if test_key_1(K) **then**
      for $K_3[31 \dots 21] \leftarrow 0 \dots 2^{11}-1$ **do**
        if test_key_2(K) **then**
          output K **and halt**

†Since the 11 bits $K_3[31 \dots 21]$ do not affect $L_{27}[0]$ or $L'_{27}[0]$, one can have any value β from the set $\{1, \dots, 2^{11}-1\}$ in place of 0^{11}. We have used 0^{11} for ease of understanding how the attack works.

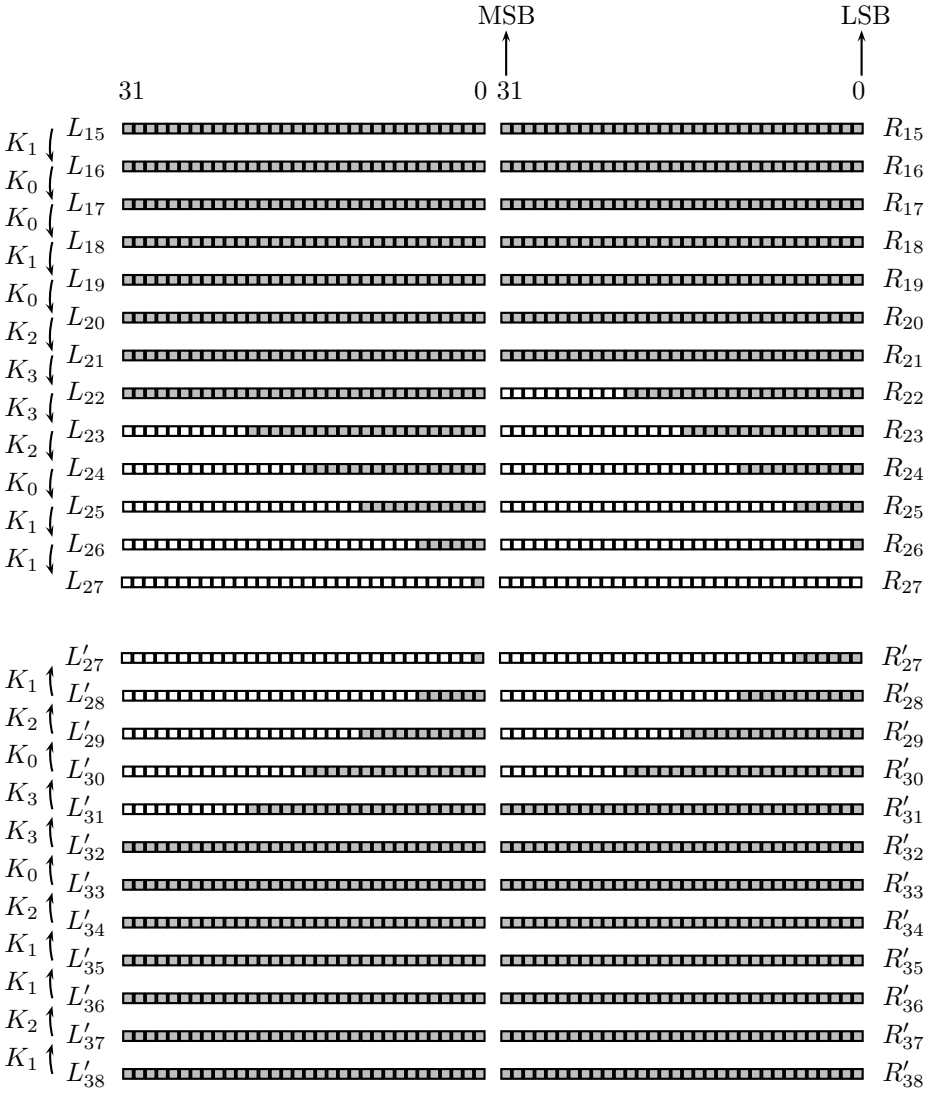


Figure 4 – 23-round attack (rounds 16–38), using 11 *inner rounds* (the grey boxes represent bits that do not depend on the value of $K_3[31 \dots 21]$)

Publication Chapter

Meet-in-the-Middle Attacks on Reduced-Round GOST

Publication Data

Gautham Sekar, Nicky Mouha, and Bart Preneel. Meet-in-the-Middle Attacks on Reduced-Round GOST. ISO/IEC JTC1/SC27 N8875, April 2010.

This paper is currently under submission to an international journal.

Contributions

- Main author together with Gautham Sekar. The paper is strongly influenced by our XTEA results [9] (see p. 147), because the ciphers can be analyzed in a similar way.

Meet-in-the-Middle Attacks on Reduced-Round GOST*

Gautham Sekar†, Nicky Mouha‡, and Bart Preneel

[1] Department of Electrical Engineering ESAT/SCD-COSIC,
Katholieke Universiteit Leuven, Kasteelpark Arenberg 10, B-3001 Heverlee, Belgium.
[2] Interdisciplinary Institute for BroadBand Technology (IBBT), Belgium.
{Gautham.Sekar,Nicky.Mouha,Bart.Preneel}@esat.kuleuven.be

Abstract. The block cipher GOST (GOST 28147-89) is a Russian standard for encryption and message authentication that is included in OpenSSL 1.0.0. In this paper, we present meet-in-the-middle attacks on several block ciphers, each consisting of 22 or fewer rounds of GOST. Our 22-round attack on rounds 10–31 requires only 5 known plaintexts and a computational effort equivalent to testing about 2^{223} keys for a success probability of $1 - 2^{-65}$. This attack is the best (going by the number of rounds) low data complexity key-recovery attack on GOST. A variant of the classical meet-in-the-middle approach is presented as well.

Keywords: Cryptanalysis, block cipher, meet-in-the-middle attack, Feistel network, GOST

1 Introduction

The GOST block cipher (GOST 28147-89) is a Russian standard for encryption and message authentication [7]. From hereon, we will refer to it as "GOST" for simplicity. It was designed in the erstwhile USSR, and declassified in 1989. This cipher is used in several applications, including OpenSSL 1.0.0, an open source toolkit for SSL/TLS [6].

Both GOST and the US standard DES [5] are Feistel networks. GOST has 32 rounds, a block size of 64 bits and a key size of 256 bits. Following its release to the public, several cryptanalysis results were published. Full-key recovery attacks on GOST are listed in Table 1. In this table, we omitted attacks that work only for classes of weak keys, as well as related-key attacks.

*This work was supported in part by the Research Council K.U.Leuven: GOA TENSE, and by the IAP Program P6/26 BCRYPT of the Belgian State (Belgian Science Policy), and in part by the European Commission through the ICT program under contract ICT-2007-216676 ECRYPT II.

†This author is supported by an FWO project.

‡This author is funded by a research grant of the Institute for the Promotion of Innovation through Science and Technology in Flanders (IWT-Vlaanderen).

Table 1 – Full-key recovery attacks on GOST; if explicitly stated in the original paper, success probabilities are given as well (KP: known plaintext, CP: chosen plaintext, MitM: meet-in-the-middle); attacks on weak key classes or using related keys are not included

Attack	Ref.	# Rounds	Time	Data	Pr[Success]
MitM	**This paper**	**8**	$\mathbf{2^{127.00}}$	**3 KPs**	$\mathbf{1-2^{-65}}$
MitM	**This paper**	**9, 10**	$\mathbf{2^{159.00}}$	**3 KPs**	$\mathbf{1-2^{-33}}$
MitM	**This paper**	**11, 12**	$\mathbf{2^{191.00}}$	**4 KPs**	$\mathbf{1-2^{-65}}$
Differential	[10]	13	Not given	2^{51} CPs	Not given
MitM	**This paper**	**13, 14**	$\mathbf{2^{223.00}}$	**4 KPs**	$\mathbf{1-2^{-33}}$
MitM	**This paper**	**16**	$\mathbf{2^{223.00}}$	**5 KPs**	$\mathbf{1-2^{-65}}$
MitM	**This paper**	**22**	$\mathbf{2^{223.00}}$	**5 KPs**	$\mathbf{1-2^{-65}}$
Slide	[1]	24	2^{64}	$\approx 2^{64}$ KPs	Not given
Slide	[1]	30	$2^{253.7}$	$\approx 2^{64}$ KPs	Not given
Reflection	[4]	30	2^{224}	2^{32} KPs	Not given

The meet-in-the-middle attack. Let $\mathcal{M}$ and $\mathcal{K}$ denote the message space and the key space, respectively. Let A_K, $B_K : \mathcal{M} \times \mathcal{K} \to \mathcal{M}$ denote two block ciphers and let $Y_K = B_K \circ A_K$, where $\circ$ denotes function composition. In a meet-in-the-middle attack, the adversary deduces K from a known plaintext-ciphertext pair (p, c), where $c = Y_K(p)$, by solving $A_K(p) = B_K^{-1}(c)$.

In this paper, we use a variant of this technique to attack 16 rounds of GOST. In this approach, the place where the meet-in-the-middle occurs is at the subkeys instead of at the intermediate texts. This technique will be explained in Sect. 4.

Contribution of this paper. In this paper, we present meet-in-the-middle attacks on block ciphers consisting of up to 22 rounds of GOST. Our aim is to find out the maximum number of rounds that could be attacked given the following criteria.

1. The key is recovered with an information theoretically optimal probability of success indicated by the unicity distance [11].
2. The attack is in a non-related-key setting.
3. The attack works for the full key space (i.e., no classes of weak keys are used).
4. Very few known plaintext-ciphertext pairs (KPs) are required.

These criteria make the scenario very difficult from the point of view of the attacker. In Table 1, the 24-round and 30-round slide attacks require almost the entire

codebook. The 30-round reflection attack also requires a large number of KPs when compared to our 22-round attack, with the time complexities of both attacks being identical. Therefore, our 22-round attack may be regarded as the best attack (going by the number of rounds) to recover the key with a low data complexity.

Biryukov and Wagner show in [2] that the reversal in the order in which the subkeys are used in the last 8 rounds, helps preclude slide attacks. We find that this reversal is responsible for many of the attacks (including the 22-round one) in this paper.

Organization. The paper is organized as follows. The specifications of GOST algorithm are given in Sect. 2. In Sect. 3, we describe our attacks on block ciphers consisting of up to 14 rounds of GOST. Sections 4 and 5 describe our attacks on 16 and 22 GOST rounds, respectively. We suggest countermeasures and conclude the paper in Sect. 6.

2 Description of GOST

First, we introduce the following notation. Addition modulo 2^{32} will be represented by $\boxplus$ and $\boxminus$ respectively. We will use $\oplus$ to denote exclusive-OR, $\lll$ for circular left shift and $\|$ for concatenation.

The block cipher GOST has a block size of 64 bits and a key size of 256 bits. It is a 32-round Feistel network in which each round uses eight 4×4 S-boxes.

The 256-bit key K of GOST is divided into eight 32-bit subkeys $K_0, \ldots, K_7$. At every round, one of the 8 subkeys is selected according to a simple key schedule. The 32-bit subkey α_i used in round i, where $1 \leq i \leq 32$, is chosen from the set $\{K_0, \ldots, K_7\}$ according to the following rule:

$$\alpha_i \leftarrow \begin{cases} K_{i-1 \bmod 8} & \text{if } i \in \{1, \ldots, 24\} \ , \\ K_{32-i \bmod 8} & \text{if } i \in \{25, \ldots, 32\} \ . \end{cases} \tag{1}$$

In this paper, we will show that the reversal of the round-key order (in the last 8 rounds), is not a good design choice with respect to meet-in-the-middle attacks.

The 64-bit input to round i of GOST consists of two 32-bit parts L_{i-1} and R_{i-1}. For round 1, the plaintext p is used as input: $(L_0 \parallel R_0) \leftarrow p$. The input for round $i+1$ is computed iteratively from the input to round i as given by $L_i \leftarrow R_{i-1}$ and $R_i \leftarrow L_{i-1} \oplus (S(R_{i-1} \boxplus \alpha_i) \lll 11)$. We select α_i according to (1). The concatenated output from the 8 S-boxes of round i is denoted by $S(x)$, where x is split into 4-bit words. The ciphertext c of GOST is produced by concatenating the two parts obtained after the 32nd round: $c \leftarrow R_{32} \parallel L_{32}$. A full description of the GOST block cipher is given in [7].

Note that [7] does not specify the S-boxes. Saarinen [8] has developed an attack with 2^{32} CPs to recover the S-boxes, assuming the attacker has black box access to the encryption device, and can specify the key used to encrypt. As his attack works for any number of rounds, it can be used to turn each of the attacks in this

paper into an attack with secret S-boxes. First, the S-boxes are recovered using Saarinen's attack, and afterwards the secret key is recovered.

3 Attacking up to 14 Rounds of GOST

In this section, we show how to construct an attack on block ciphers consisting of r rounds of GOST, where $8 \leq r \leq 14$. In each of these block ciphers, at least one subkey is not used. Therefore, exhaustive search requires less than 2^{255} encryptions on average.

From (1), we obtain ciphers with unused subkey(s). Table 2 lists all these ciphers.

Table 2 – All r-round reduced block ciphers ($8 \leq r \leq 14$) with unused subkeys

# Rounds	Rounds
8	18–25, 19–26, 20–27, 21-28, 22-29, 23-30, 24-31
9	18–26, 19–27, 20–28, 21-29, 22-30, 23-31
10	18–27, 19–28, 20–29, 21-30, 22-31
11	18–28, 19–29, 20–30, 21-31
12	18–29, 19–30, 20–31
13	18–30, 19–31
14	18–31

We now evaluate the data and time required for attacking the block ciphers listed in Table 2. Let us consider a block cipher in which s 32-bit subkeys, $1 \leq s \leq 4$, are not used.

Given one plaintext-ciphertext pair (p_0, c_0), with each key guess, the attacker checks whether

$$E_K^{(a \ldots a+r-1)}(p_0) = c_0 \ , \tag{2}$$

where $E_K^{(a \ldots a+r-1)}$ denotes the r-round (rounds a to $a+r-1$) encryption using the k-bit key K, where $k = (256 - 32 \cdot s)$. One KP is not sufficient, because the key space ($2^{256-32 \cdot s}$ keys K) is larger than the ciphertext space (2^{64} ciphertext blocks). Therefore, the attacker requires more known plaintext-ciphertext pairs to determine the key K with sufficiently high probability. The number of KPs is denoted by n.

For every candidate k-bit key K, the attacker tests (2) using the first KP. If this equality is satisfied, the attacker uses a subsequent KP to check

$$E_K^{(a \ldots a+r-1)}(p_j) = c_j \ , \tag{3}$$

where j is at most $n-1$. If one of the n equations (2), (3) is not satisfied, the candidate key K is incorrect and can be discarded.

Throughout this paper, we use the reasonable assumption that every block cipher under consideration has perfect confusion and diffusion properties as defined by Shannon [11]. If either the plaintext or the key, or both are changed, we assume that the corresponding ciphertext will be generated uniformly at random, independent from previously obtained ciphertexts.

With this assumption, each of the 64-bit conditions resulting from (2), (3) is satisfied with probability 2^{-64}. We now calculate the data and time complexities for our attacks. All time complexities are stated as the number of equivalent encryptions of the reduced-round block cipher.

The average success probability can be calculated as follows. The n 64-bit conditions are simultaneously satisfied with probability $2^{-n\cdot 64}$. The attacker can therefore eliminate a wrong key with probability $1-2^{-n\cdot 64}$. Assume that key m is the correct key, where $0 \le m < 2^k$. This key will be found by our attack if all previous keys are eliminated. This happens with probability $(1-2^{-n\cdot 64})^m$. The correct key can be located anywhere among the list of 2^k candidate keys with equal probability. Therefore, the average success probability is

$$2^{-k} \cdot \sum_{m=0}^{2^k-1} (1-2^{-n\cdot 64})^m = 2^{n\cdot 64-k} \cdot (1-(1-2^{-n\cdot 64})^{2^k}) \ , \\ \approx 2^{n\cdot 64-k} \cdot (1-e^{-2^{k-n\cdot 64}}) \approx 1-2^{k-n\cdot 64-1} \ , \qquad (4)$$

assuming $2^{k-n\cdot 64} \approx 0$. The approximations result from using the first and the second order Taylor approximations of e^x around 0. We now calculate the time complexity of the attack. For a candidate key K to be determined as wrong, the expected number of trials is $1+2^{-64}+\ldots+2^{-(n-1)\cdot 64}$. The average (equivalent) number of encryptions of the algorithm is given by:

$$2^{-k} \cdot \sum_{m=0}^{2^k-1} \Big(m \cdot (1+2^{-64}+\ldots+2^{-(n-1)\cdot 64}) + n \Big) = \frac{1}{2} \cdot \frac{1-2^{-n\cdot 64}}{1-2^{-64}} \cdot (2^k-1) + n \ . \qquad (5)$$

Table 3 gives the average time complexities and the average success probabilities for various values of s $(= (256-k)/32)$ and n. The approximate number of plaintext-ciphertext pairs that are needed can also be calculated from Shannon's unicity distance [11] as $k/64$.

We note that (2), (3) can be replaced with $E_K^{(a\ldots t-1)}(p_j) = D_K^{(t\ldots a+r-1)}(c_j)$, where $0 \le j < n$, $t \in \{a, \ldots, a+r-1\}$, $E_K^{(a\ldots a-1)}(p_j) = p_j$, and $D_K^{(t\ldots a+r-1)}$ denotes $(a+r-t)$-round (rounds t to $a+r-1$) decryption using the key K. Therefore, the attacks in this section can also be seen as meet-in-the-middle attacks.

Table 3 – Time complexities and success probabilities of attacks of Sect. 3 for several values of s and n

s	n	k	**Average time complexity**	**Average success probability**
1	4	224	2^{223}	$1-2^{-33}$
2	4	192	2^{191}	$1-2^{-65}$
3	3	160	2^{159}	$1-2^{-33}$
4	3	128	2^{127}	$1-2^{-65}$

4 Attack on 16-Round GOST

In this section, we analyze the block cipher consisting of rounds 17–32 of GOST. We begin with the observation that K_7 is used consecutively in rounds 24 and 25.

Our attack assumes that the S-boxes are bijective. Note, however, that a similar attack works for non-bijective S-boxes, but then the computations of the time complexity and success probability become more involved.

Let $K = (K_0, K_1, K_2, K_3, K_4, K_5, K_6, X)$, where X is not relevant to the analysis because the attacker exhaustively searches over all subkeys except K_7. For every candidate key K, the attacker computes $E_K^{(17\ldots23)}(p_0)$, given a plaintext-ciphertext pair (p_0, c_0), and gets L_{23} and R_{23}. Similarly, the attacker computes $D_K^{(26\ldots32)}(c_0)$ and gets L_{25} and R_{25}. Using $\alpha_{24} = S^{-1}((L_{25} \oplus L_{23}) \ggg 11) \boxminus R_{23}$ and $\alpha_{25} = S^{-1}((R_{25} \oplus R_{23}) \ggg 11) \boxminus L_{25}$, the subkeys used in rounds 24 and 25 are obtained. If they are equal (for a wrong candidate key K, this happens with probability 2^{-32}),[3] the attacker sets $K_7 \leftarrow \alpha_{24} = \alpha_{25}$.

Then, using $n-1$ other plaintext-ciphertext pairs (p_j, c_j), $1 \le j \le n-1$, the attacker tests if $E_K^{(17\ldots32)}(p_j) = c_j$ with the value found for K_7. A wrong key will pass these tests with probability $2^{-32} \cdot \left(2^{-64}\right)^{n-1} = 2^{-32-(n-1)\cdot 64}$. Thus, with probability $1 - 2^{-32-(n-1)\cdot 64}$, a wrong key is eliminated. Using a similar reasoning as in Sect. 3, we obtain the average success probability:

$$
\begin{aligned}
2^{-224} \cdot \sum_{m=0}^{2^{224}-1} (1 - 2^{-32-(n-1)\cdot 64})^m &= 2^{32+(n-1)\cdot 64-224} \cdot (1 - (1 - 2^{-32-(n-1)\cdot 64})^{2^{224}}) \\
&\approx 2^{32+(n-1)\cdot 64-224} \cdot (1 - e^{-2^{224-32-(n-1)\cdot 64}}) \\
&\approx 1 - 2^{224-32-(n-1)\cdot 64-1} \quad , \qquad (6)
\end{aligned}
$$

where the approximations hold when $n \ge 5$. We now calculate the time complexity of the attack. For a candidate key K to be determined as wrong, the expected

[3] If the texts obtained by encrypting p_0 and decrypting c_0, in the 13 outer rounds, are distributed uniformly at random, then so are the subkeys in rounds 24 and 25.

number of trials is $1 + 2^{-32} + 2^{-32-64} + \ldots + 2^{-32-(n-2)\cdot 64}$. This is because for every candidate key K, the attacker always checks whether the subkeys used in rounds 24 and 25 agree. For 2^{-32} candidate keys, the attacker uses the second known plaintext, for 2^{-96} the attacker uses the third known plaintext, and so on. If the candidate key is correct, the attacker always performs n encryptions. As the correct key can be located anywhere in the list of 2^{224} candidates keys with equal probability, the average number of 16-round computations is

$$2^{-224} \cdot \sum_{m=0}^{2^{224}-1} \left(m \cdot (1 + 2^{-32} + 2^{-32-64} + \ldots + 2^{-32-(n-2)\cdot 64}) + n\right)$$
$$= \frac{1}{2} \cdot (1 + 2^{-32} + 2^{-32-64} + \ldots + 2^{-32-(n-2)\cdot 64}) \cdot (2^{224} - 1) + n \ . \qquad (7)$$

Substituting $n = 5$ in (6) and (7), the average success probability is $1 - 2^{-65}$ and the average number of 16-round computations is $2^{223.00}$.

5 Attack on 22-Round GOST

From (1), we observe that the subkey K_0 is used only once in the block cipher consisting of rounds 10–31 of GOST. Therefore, here the attacker first checks for the equality of R_{16} and R'_{16}. These are obtained by respectively computing $E_K^{(10\ldots16)}(p_0)$ and $D_K^{(18\ldots31)}(c_0)$, where $K = (X, K_1, K_2, K_3, K_4, K_5, K_6, K_7)$. As subkey K_0 is not necessary to perform these partial encryptions and decryptions, X can be any 32-bit value.

If $R_{16} = R'_{16}$ (this happens with probability 2^{-32}), the corresponding value of K_0 $(= \alpha_{17})$ is obtained using:

$$\alpha_{17} = S^{-1}((R_{17} \oplus L_{16}) \ggg 11) \boxminus R_{16} \ . \qquad (8)$$

The attacker uses $n - 1$ KPs (p_j, c_j) subsequently to check $E_K^{(10\ldots31)}(p_j) = c_j$ with the value obtained for K_0. For every j, where j is at most $n-1$, this equation is satisfied with probability 2^{-64}.

Using the same formulas as in Sect. 4, we find an average time complexity of $2^{223.00}$ for a success probability of $1 - 2^{-65}$. A similar attack can be mounted on other reduced-round block ciphers, each with less than 22 GOST rounds (e.g., rounds 11–31), where a particular subkey is used only once. Again, attacks similar to those in this section can be applied to the respective block ciphers even if the S-boxes are not bijective.

6 Conclusions and Open Problems

This paper presented several meet-in-the-middle attacks on GOST reduced to up to 22 rounds. To the best of our knowledge, the 22-round attack is the best attack (going by the number of rounds) to recover the key with very few known plaintexts.

Our attacks use different approaches – attacks on 14 or fewer rounds use a straightforward meet-in-the-middle approach and so does the 22-round attack; in the 16-round attacks, the meet-in-the-middle corresponds to inner round subkeys rather than intermediary text values. Our attacks work in a non-related-key setting.

The time complexity of both the 16-round and 22-round attacks is $2^{223.00}$. It is required in these attacks that the S-boxes are bijective, but similar attacks can be constructed as well if this is not the case.

An interesting open problem would be extending our attacks to more rounds using other approaches to the meet-in-the-middle technique; for example, similar to those of [3].

References

[1] E. Biham, O. Dunkelman, and N. Keller. Improved Slide Attacks. In A. Biryukov, editor, *FSE*, volume 4593 of *Lecture Notes in Computer Science*, pages 153–166. Springer, 2007.

[2] A. Biryukov and D. Wagner. Advanced Slide Attacks. In B. Preneel, editor, *EUROCRYPT*, volume 1807 of *Lecture Notes in Computer Science*, pages 589–606. Springer, 2000.

[3] O. Dunkelman, G. Sekar, and B. Preneel. Improved Meet-in-the-Middle Attacks on Reduced-Round DES. In K. Srinathan, C. P. Rangan, and M. Yung, editors, *INDOCRYPT*, volume 4859 of *Lecture Notes in Computer Science*, pages 86–100. Springer, 2007.

[4] O. Kara. Reflection Cryptanalysis of Some Ciphers. In D. R. Chowdhury, V. Rijmen, and A. Das, editors, *INDOCRYPT*, volume 5365 of *Lecture Notes in Computer Science*, pages 294–307. Springer, 2008.

[5] National Institute of Standards and Technology. FIPS PUB 46-3: Data Encryption Standard (DES), October 1999. `http://www.itl.nist.gov/fipspubs/fip186-2.pdf`.

[6] OpenSSL version 1.0.0, March 2010. `http://www.openssl.org/`.

[7] J. Pieprzyk and L. Tombak. Soviet Encryption Algorithm, June 1994. `http://freeworld.thc.org/root/phun/stego-challenge/gost-spec.pdf`.

[8] M.-J. Saarinen. A chosen key attack against the secret S-boxes of GOST, 1998. `http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.41.5532`.

[9] G. Sekar, N. Mouha, V. Velichkov, and B. Preneel. Meet-in-the-Middle Attacks on Reduced-Round XTEA. In A. Kiayias, editor, *CT-RSA*, volume 6558 of *Lecture Notes in Computer Science*, pages 250–267. Springer, 2011.

[10] H. Seki and T. Kaneko. Differential Cryptanalysis of Reduced Rounds of GOST. In D. R. Stinson and S. E. Tavares, editors, *Selected Areas in Cryptography*, volume 2012 of *Lecture Notes in Computer Science*, pages 315–323. Springer, 2000.

[11] C. E. Shannon. Communication Theory of Secrecy Systems. *Bell System Technical Journal*, 28:656–715, 1949.

Publication Chapter

Challenging the Increased Resistance of Regular Hash Functions Against Birthday Attacks

Publication Data

Nicky Mouha, Gautham Sekar, and Bart Preneel. Challenging the Increased Resistance of Regular Hash Functions Against Birthday Attacks. ECRYPT II Hash Workshop, May 2011. `http://www.ecrypt.eu.org/hash2011/`.

This paper is currently under submission to an international journal.

Contributions

- Main author together with Gautham Sekar.

Challenging the Increased Resistance of Regular Hash Functions Against Birthday Attacks*

Nicky Mouha[1,2,†], Gautham Sekar[3,‡], and Bart Preneel[1,2]

[1] Department of Electrical Engineering ESAT/SCD-COSIC,
Katholieke Universiteit Leuven, Kasteelpark Arenberg 10, B-3001 Heverlee, Belgium.
[2] Interdisciplinary Institute for BroadBand Technology (IBBT), Belgium.
[3] Temasek Laboratories, National University of Singapore,
5A, Engineering Drive 1, #09-02, Singapore 117411.
{Nicky.Mouha,Bart.Preneel}@esat.kuleuven.be, tslgs@nus.edu.sg

Abstract. At EUROCRYPT 2004, Bellare and Kohno presented the concept of a regular hash function. For a hash function to be regular, every hash value must have the same number of preimages in the domain. The findings of their paper remained unchallenged for over six years, and made their way into several research papers and textbooks. In their paper, Bellare and Kohno claim that regular hash functions are more resistant against the birthday attack than random hash functions. We counter their arguments, by showing that the success probability of the birthday attack against a regular hash function can be made arbitrarily close to that of a random hash function (for the same number of trials). Our analysis uses the fact that the choices of the attacker can be limited to any subset of the domain. Furthermore, we prove that it is not possible to construct a hash function that is regular for only a small fraction of subsets of the domain. In order to avoid these problems, we propose to model hash functions as random functions. Compared to regular functions, we argue that the statistics of random functions are more similar to hash functions used in practice, regardless of how the attacker chooses the domain points.

Keywords: Hash function, balance, regularity, birthday attack, (linear) subset regularity, random function.

*This work was supported in part by the Research Council K.U.Leuven: GOA TENSE, and by the IAP Program P6/26 BCRYPT of the Belgian State (Belgian Science Policy), and in part by the European Commission through the ICT program under contract ICT-2007-216676 ECRYPT II.

†This author is funded by a research grant of the Institute for the Promotion of Innovation through Science and Technology in Flanders (IWT-Vlaanderen).

‡Part of this work was performed during the author's PhD research at COSIC.

1 Introduction

Let $h: D \to R$ be a function, where both the domain D and the range R are finite sets. Denote $|D|$ by d and $|R|$ by r. If $d > r$, h is referred to as a hash function. Although strictly not required, D is finite for commonly used hash functions; e.g. for SHA-1, D consists of all strings of length at most $2^{64} - 1$ bits [8].

Any pair (x, y) where $x, y \in D$ for which $x \neq y$ and $h(x) = h(y)$, is denoted as a *collision* for hash function h. A *possibly trivial collision* is a pair (x, y) where $x, y \in D$ for which $h(x) = h(y)$. That is, unlike for a collision, it is allowed that $x = y$. Among other requirements [14], a cryptographic hash function should be collision resistant, i.e. it should be computationally infeasible to find a collision. In this paper, a "hash function" does not necessarily refer to a "cryptographic hash function." For any choice of h, a generic birthday attack can be used to find a collision.

In a birthday attack, points $x_1, \ldots, x_q$ are picked from D. For $i = 1, \ldots, q$, we compute $y_i = h(x_i)$. We say that the birthday attack is successful, if we find $h(x_i) = h(x_j)$, where $1 \leq i < j \leq q$. We refer to q as the number of trials of the birthday attack.

There are several variants of the birthday attack, that differ in the way that the points $x_1, \ldots, x_q$ are chosen. In their analysis [1, 2], Bellare and Kohno only consider the case where the domain points are chosen independently and uniformly at random from all m-bit strings (therefore $|D| = 2^m$). Yuval [26] instead suggests using q minor modifications of a message, in such a way that all messages are meaningful. Using distinguished points, Quisquater and Delescaille [18] showed that collisions for meaningful messages can also be found with negligible memory requirements, i.e. without storing all $(x_i, h(x_i))$ for $i = 1, \ldots, q$. An efficient parallel implementation of their algorithm was proposed by Van Oorschot and Wiener [22].

For most applications, only a small subset of all m-bit strings are meaningful. If, for example, the messages consist of only ASCII characters, a necessary (but not sufficient) requirement is that the most significant bit of every character is zero.

Let $C_h(q)$ be the probability that the birthday attack finds a possibly trivial collision for h after q trials. If for every domain point, the corresponding range point of h is chosen uniformly and independently at random from all r range points, we refer to h as a random function. The success probability of the birthday attack for a random function is denoted as $C_h^{\$}(q)$.

Bellare and Kohno [2] point out that if h is a random function, this does not necessarily mean that $h(x)$ is uniformly distributed in R. In order to have such a uniform distribution in R, every range point must have the same fraction of preimages under the hash function. We refer to such a hash function as a regular function. This can be defined more formally as follows.

Definition 1 (Balance and Regularity)**.** Let $h : D \to R$ be a hash function with

domain D of size d and range $R = \{R_1, R_2, \ldots, R_r\}$ of size r. For h to be a hash function, we must have $d > r$. For $1 \leq i \leq r$, $d_i = |h^{-1}(R_i)|$ denotes the size of the preimage of R_i under h. The *balance* of h is then defined as

$$\mu(h) = \log_r \left(\frac{d^2}{d_1^2 + d_2^2 + \ldots + d_r^2} \right) \quad . \tag{1}$$

A hash function is regular iff $\mu(h) = 1$ (that is, iff $\forall\ 1 \leq i \leq r : d_i = d/r$) [2, §5].

If h is a regular function, the success probability of the birthday attack is denoted by $C_h^{\text{reg}}(q)$. Bellare and Kohno calculate[4] that $C_h^{\$}(q) > (8/5) \cdot C_h^{\text{reg}}(q)$, if $d = 2r \geq 10$. Therefore, they conclude that "regular functions fare better than random functions [against the birthday attack]."

We recall that their reasoning assumes that the attacker chooses the messages uniformly at random from D. In the following sections, we investigate the case where the attacker limits the choice of the domain points to subsets of D. We prove that it is not possible to construct a hash function that is regular with respect to only a small fraction of subsets of the domain. For this, we introduce the concepts of subset regularity and linear subset regularity.

Bellare and Kohno pointed out in their analysis that there is only a small difference between regular and random functions in their resistance against the birthday attack. For random functions, the success probability of the birthday attack does not depend on how the attacker chooses the domain points.

NIST is currently holding a competition in search for a new hash function standard [15]. Our result may be relevant to the analysis of statistical properties of the hash functions in this competition.

Organization. This paper is organized as follows. In Sect. 2, we describe the birthday problem and its relation to the birthday attack. Section 3 provides a brief overview of some works that employ the notion of regularity. In Sect. 4, we compute the ratio of regular functions to all functions with the same domain and range. Our notions of subset regularity and linear subset regularity are introduced in Sects. 5 and 6, respectively. The impact of our observations on the birthday attack is discussed in Sect. 7, where we show that the success probability of the birthday attack against a regular hash function can be made arbitrarily close to that of a random hash function (for the same number of trials). In Sect. 8, we describe the relation of regularity to the dictionary problem in computer science.

We propose in Sect. 9 that, to analyze the complexity of the birthday attack for commonly used hash functions, we model hash functions as random functions instead of as regular functions. A proof that random functions do not suffer from any of the problems in this paper is given as well. We conclude in Sect. 10.

We show in Appendix A how the construction of a 3-to-1 bit linear subset regular hash function fails. In Appendix B, we calculate the inverse of some

[4] In this proof, the values from the domain are randomly chosen, but with replacement. The replacement can result in a possibly trivial collision with a collision in the domain. The authors show in [2, §7.2] that the higher success probability is not due to the possibility of such collisions.

matrices that we use in Sect. 6.

2 The Birthday Problem

Assume that there are N people in one room. How large must N be, in order to have a probability of at least $1/2$ that two people share the same birthday? It is assumed that birthdays are independently and uniformly distributed over the 365 days of the year (leap years are ignored). This is the birthday problem (see Feller [9, §2.3]), which dates back to von Mises [23]. The answer to the problem is $N \geq 23$.

Bloom showed that the probability that two people share the same birthday, is the lowest when birthdays are uniformly distributed [3]. Nunnikhoven [16] analyzed the birthday problem for nonuniform birth frequencies.

Based on the mathematics of the birthday problem, Yuval proposed the birthday attack for hash functions [26]. In the attack, a large number of messages are generated, until two messages are found that result in the same hash value. The attack complexity depends on the distribution of the hash values. If the hash values are uniformly distributed, the analysis of the original birthday problem applies. In case of a nonuniform distribution, collision probabilities were calculated by Cachin [5, §3.2.5], as well as Bellare and Kohno [2].

In this paper, we point out that the distribution of the hash values not only depends on the hash function, but also on how the attacker chooses the input messages. This is different from the birthday problem, where the probability distribution of the birthdays is fixed in advance (to have a uniform distribution). In the following sections, we investigate the impact of the attacker's choice of the messages.

3 Balance and Regularity in Existing Literature

The results of [2] not only remained unchallenged for over six years, but were also often cited in papers on cryptographic theory, in cryptanalysis papers and in textbooks. In this section, we give a brief overview of some of the most notable results.

Since Bellare and Kohno introduced their balance measure $\mu(h)$ in $[1, 2]$ (defined in (1)), this measure has been applied to several hash functions. Already in their original paper, the balance measures of truncated variants of SHA-1 were analyzed. Later, Yoshida *et al.* calculated the balance of a reduced version of MAME [25]. Ødegård and Gligoroski recently computed the balance measures of reduced versions of EDON-$\mathcal{R}$ [17].

In each of these papers, hash function balances are calculated. However, the results show that not a single one of the hash functions variants under consideration is regular, and the balance measure $\mu(h)$ seems to decrease if the number of output

bits of h is increased. The balance of the actual (untruncated) hash functions is never calculated, because this would be computationally infeasible. Because of this difficulty, we question the applicability of the balance measure to analyze practical hash functions.

The notions of balance and regularity also appear in several textbooks. In [11], Goldwasser and Bellare state that "If h is not regular, it turns out the [birthday] attack succeeds even faster, telling us that we ought to design hash functions to be as "close" to regular as possible." In this paper, we explain why we counter this design criterion.

Buchmann's book [4] states: "We assume that strings from [the domain] can be chosen such that the distribution on the corresponding hash values is the uniform distribution." However, it is the attacker who can freely determine how strings are chosen from the domain. In this paper, we show that there always exists a way for the attacker to restrict the domain so that the resulting function is not regular.

In the first edition of his book [19], Stinson describes the birthday attack under the assumption that the hash function is regular. This assumption is dropped in the second edition [20], in favor of random oracles [10].

In [13], Joux refers to [1] for a more precise analysis of collisions in hash functions for the unbalanced case. Bellare and Kohno provide bounds for this unbalanced case [2], which they refer to as "the generalized birthday problem". The reader should not confuse this with the generalized birthday problem that Wagner studied earlier [24].

4 Fraction of Regular Functions

We begin with the following lemmata.

Lemma 1. *The total number of hash functions $|h|$ is given by r^d.*

Proof. Each of the d elements of the domain, can have r possible range points. This results in a total of r^d combinations. □

Lemma 2. *The total number of functions $|h^{\mathrm{reg}}|$ that are regular is given by*

$$|h^{\mathrm{reg}}| = \begin{cases} {}^{d!}\!/_{((d/r)!)^r} & \text{if } r \mid d \ , \\ 0 & \text{if } r \nmid d \ . \end{cases} \tag{2}$$

Proof. For a function to be regular, each range point must have the same number of preimages under the function. This is achieved if and only if $r \mid d$. Given that the function is regular, the first range point that we consider has one of $C(d, d/r)$ possible sets of d/r preimages mapping to it. Here, $C(u, v)$ denotes the quantity $u!/(v! \cdot (u - v)!)$. Any domain point in the set that maps to this range point cannot map to any other range point; otherwise the mappings do not constitute a function. Therefore, the second range point that we consider will have one of only

$C(d - d/r, d/r)$ possible sets of d/r preimages mapping to it. Similarly, the i-th range point will have one of $C(d-(i-1)\cdot d/r, d/r)$ sets of domain points mapping to it. In total, therefore, we have

$$\prod_{i=1}^{r} C(d - (i-1)\cdot d/r, d/r) = \frac{d!}{((d/r)!)^r} \tag{3}$$

functions that are regular. Figure 1 illustrates the above arguments with an example. □

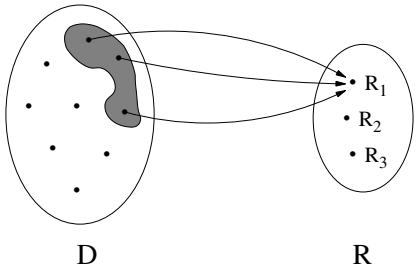


Figure 1 – In this example, $d = 9$ and $r = 3$; the shaded area represents one of the $C(9,3)$ possible sets of 3 domain points that can map to the range point R_1 given that the function is regular; for R_2 there are only $C(6,3)$ sets

Theorem 1. *Assume $r \mid d$. The probability that a random function is also a regular function, is given by*

$$\frac{|h^{\mathrm{reg}}|}{|h|} \approx 2^{-(r/2)\cdot\log_2(2\pi d/r)} \quad . \tag{4}$$

Proof. Stirling's approximation:

$$\log_2(z!) \approx \frac{1}{2}\log_2(2\pi z) + z\log_2\left(\frac{z}{e}\right) \quad . \tag{5}$$

Using Lemma 1 and Lemma 2, we obtain:

$$\begin{aligned}
\log_2\left(\frac{|h^{\mathrm{reg}}|}{|h|}\right) &= \log_2(|h^{\mathrm{reg}}|) - \log_2(|h|) \\
&= \log_2\left(d!/(\tfrac{d}{r}!)^r\right) - \log_2\left(r^d\right) \\
&= \log_2(d!) - r\log_2\left(\frac{d}{r}!\right) - d\log_2(r)
\end{aligned}$$

$$
\begin{aligned}
&\approx \frac{1}{2}\log_2(2\pi d) + d\log_2\left(\frac{d}{e}\right) \\
&\quad - \frac{r}{2}\log_2(2\pi d/r) - d\log_2\left(\frac{d}{re}\right) \\
&\quad - d\log_2(r) \\
&= \frac{1}{2}\log_2(2\pi d) - \frac{r}{2}\log_2(2\pi d/r) \\
&\approx -\frac{r}{2}\log_2\left(\frac{2\pi d}{r}\right) \quad . \qquad (6)
\end{aligned}
$$

□

Let us consider a random hash function with $d = 2^{161}$ and $r = 2^{160}$. According to Theorem 1, the probability[5] that this function is a regular function, is $2^{-2^{160.9}}$. We note that it is therefore extremely unlikely that a hash function chosen uniformly at random from the set of r^d hash functions is regular. This relates to the observations made in the literature study of Sect. 3, where we discuss papers that analyze the balance of several hash function variants.

5 Subset Regularity

First, we recall a rather obvious point from [2]. Assume that for an n-bit hash function h, we restrict the input of h to messages of at most m bits. Let g be a hash function, such that the domain is restricted to at most m' bits, where $m' \geq m$. Suppose $g(x) = h(x),\ \forall\ x : |x| \leq m$. Then, a collision for h will also be a collision for g. If g is SHA-1, then m' can be at most $2^{64} - 1$. A collision for, say, $h : \{0,1\}^{161} \rightarrow \{0,1\}^{160}$ is a collision for SHA-1 for any $m' > 161$. In other words, as separately stated by Bellare and Kohno [2, §7.2],

"[A]n adversary attacking a hash function with a very large domain D might restrict its choices of domain elements to some smaller subset of D."

One possibility is to restrict the domain elements to sets of size 2^a, where $a \in \mathbb{N}$ and $2^a > r$. In this paper, we assume that the attacker chooses to make such a restriction. We also assume that $|D|$ is even, and that the size of the restricted domain is always half the size of D.

For certain applications, the domain D must be restricted to a smaller subset. For example, if a message consists of ASCII characters, the most significant bit of every character must be zero.

[5] Although Stirling's approximation (5) is used for a small value of z, namely $d/r = 2$, all digits of the calculated probability using the approximation are correct.

Definition 2 (Subset regularity)**.** Let $h : D \to R$ be a hash function with domain D and range $R = \{R_1, R_2, \ldots, R_r\}$ of size d and r respectively. Assuming $|D|$ is even, the attacker can restrict the elements of D to a subset S such that $|S| = |D|/2$. For $1 \leq i \leq r$, $s_i = |h^{-1}(R_i) \in S|$ denotes the size of the preimage of R_i under h, when the domain is restricted to S. We say that a hash function is *subset regular* with respect to S, if it is not only regular, but also $\forall\ 1 \leq i \leq r : s_i = d/(2r)$. That is, it must also be regular when the domain is restricted to subset S. We impose the condition $d > 2r$, to ensure that $|S| > |R|$.

We now introduce the following lemma.

Lemma 3. *The total number of hash functions $|h^{\mathrm{sreg}}|$ that are subset regular with respect to S, is given by*

$$|h^{\mathrm{sreg}}| = \begin{cases} \left({}^{(d/2)!}/_{((d/2r)!)^r}\right)^2 & \text{if } 2r \mid d \ , \\ 0 & \text{if } 2r \nmid d \ . \end{cases} \tag{7}$$

Proof. Suppose that $|D|$ is even. Let the domain D be partitioned into two equally-sized sets D_1 and D_2, and consider only domain elements in one of these sets (D_1 or D_2). Then every range point can have the same number of preimages, if and only if $2r \mid d$. This also implies $r \mid d$, which is required for the regularity criterion on the entire domain. The reasoning now is exactly the same as for Lemma 2, but with d replaced by $d/2$ as the regularity criterion holds on the smaller domain as well. Because the subset regularity criterion has to hold on the other subset of the domain, we square the entire expression. If $|D|$ is not even, it is not possible that h is subset regular with respect to S. □

Theorem 2. *If $2r \mid d$, the probability that a regular function chosen uniformly at random is also subset regular with respect to S, is given by*

$$\frac{|h^{\mathrm{sreg}}|}{|h^{\mathrm{reg}}|} \approx 2^{(-r/2)\cdot \log_2(\pi d/2r)} \ . \tag{8}$$

Proof. Using Lemma 2 and Lemma 3, we obtain:

$$\begin{aligned} \log_2\left(\frac{|h^{\mathrm{sreg}}|}{|h^{\mathrm{reg}}|}\right) &= \log_2\left(|h^{\mathrm{sreg}}|\right) - \log_2\left(|h^{\mathrm{reg}}|\right) \\ &= \log_2\left(\left({}^{\frac{d}{2}!}/_{(\frac{d}{2r}!)^r}\right)^2\right) - \log_2\left({}^{d!}/_{(\frac{d}{r}!)^r}\right) \\ &= 2\log_2\left(\frac{d}{2}!\right) - 2r\log_2\left(\frac{d}{2r}!\right) - \log_2\left(d!\right) + r\log_2\left(\frac{d}{r}!\right) \\ &\approx \log_2(\pi d) + d\log_2\left(\frac{d}{2e}\right) - r\log_2(\pi d/r) - d\log_2\left(\frac{d}{2re}\right) \\ &\quad - \frac{1}{2}\log_2(2\pi d) - d\log_2\left(\frac{d}{e}\right) + \frac{r}{2}\log_2(2\pi d/r) \end{aligned}$$

$$+ d \log_2 \left(\frac{d}{re} \right)$$
$$= \frac{1}{2} \log_2 \left(\frac{\pi d}{2} \right) + \frac{r}{2} \log_2 \left(\frac{2r}{\pi d} \right) \approx -\frac{r}{2} \log_2 \left(\frac{\pi d}{2r} \right) \quad . \tag{9}$$

□

Assume that for a regular hash function, $d = 2^{162}$ and $r = 2^{160}$. The attacker decides to restrict the choice of the domain points to a smaller subset, consisting of 2^{161} elements. According to Theorem 2, the probability[6] that a randomly chosen regular function is also subset regular with respect to S, is $2^{-2^{160.5}}$.

This leads us to conclude that if h is a regular function chosen uniformly at random (from all regular functions with the same domain and range), the probability that h is also a regular function for a particular subset is negligible.

6 Linear Subset Regularity

In Sect. 5, we showed that a randomly chosen regular hash function is also subset regular with respect to S with a probability of almost zero. Our calculations assumed that r was at least reasonably large, otherwise finding collisions using the birthday attack becomes feasible in practice.

One might therefore propose the design of a hash function h that is not only regular, but also subset regular with respect to arbitrary subsets. We now prove that no such h exists, by showing that a hash function can be subset regular with respect to only a negligible fraction of all $C(d, d/2)$ possible subsets. In order to do this, we first introduce the definition of linear subset regularity.

Definition 3 (Linear subset regularity)**.** Let $h : D \to R$ be a hash function with $d = |D| = 2^m$ and $r = |R|$. Every element of D consists of m bits, which we label from x_0 to x_{m-1}, where x_0 represents the least significant bit. The attacker can restrict the elements of D to a smaller subset, including only domain points that satisfy $a_{m-1}x_{m-1} \oplus a_{m-2}x_{m-2} \oplus \ldots \oplus a_0 x_0 = 0$, where $a_i \in \{0, 1\}$. We can therefore construct $2^m - 1$ subsets of D, for all choices of a_i, $0 \leq i < m$, except the all-zero case. We impose $d > 2r$, to ensure that each of these subdomains is larger than the range R. We say that a hash function is *linear subset regular*, if it is not only regular for the domain D, but also for each of the $2^m - 1$ subsets of the domain that we defined.

We first prove that there are no m-to-1 bit hash functions that are linear subset regular. Using this, we prove that there are also no m-to-n bit hash functions that are linear subset regular.

[6]The approximation given by (8) results in $2^{-2^{160.4}}$, but we have calculated this value more accurately by including additional terms in Stirling's approximation (5).

Theorem 3. *There does not exist an m-to-1 bit hash function that is linear subset regular.*

Proof. A necessary condition for a 3-to-1 bit hash function to be linear subset regular, is that exactly four hash values are 0, and that for every linear subset exactly two hash values are 0. This condition can be described by the following system of linear equations:

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \end{bmatrix} \begin{bmatrix} h(000) \\ h(001) \\ h(010) \\ h(011) \\ h(100) \\ h(101) \\ h(110) \\ h(111) \end{bmatrix} = \begin{bmatrix} 4 \\ 2 \\ 2 \\ 2 \\ 2 \\ 2 \\ 2 \\ 2 \end{bmatrix} . \tag{10}$$

We find that there is only one solution, namely $h(000) = h(001) = \ldots = h(111) = 1/2$. As none of these range points are in the set $\{0, 1\}$, we conclude that there does not exist a 3-to-1 bit hash function that is linear subset regular. In Appendix A, we show how the explicit construction of a 3-to-1 bit linear subset regular hash function fails.

Let the 8×8 matrix in (10) be denoted as A_8. By $\bar{A}_d$ we denote the matrix that results when the logical negation operator is applied to every element of A_d. Matrices A_d can then be constructed as follows:

$$A_1 = [1] , \tag{11}$$

$$A_d = \begin{bmatrix} A_{d/2} & A_{d/2} \\ A_{d/2} & \bar{A}_{d/2} \end{bmatrix}, \quad \text{for } 1 \leq \log_2(d) \in \mathbb{N} . \tag{12}$$

Every row of A_d corresponds to a subset of the domain defined by the linear expression $a_{m-1}x_{m-1} \oplus a_{m-2}x_{m-2} \oplus \ldots \oplus a_0 x_0$, where $a_i \in \{0, 1\}$, $0 \leq i < m$ indicates if a linear term is included or not, and x_0 refers to the least significant bit. By definition, we assume that a linear expression containing zero terms corresponds to the regularity condition. The values of a_i are different for every row, and A_d is constructed such that the top $d/2$ rows have $a_{m-1} = 0$ and the bottom $d/2$ rows have $a_{m-1} = 1$.

In order to extend our result from 3-to-1 bit hash functions to m-to-1 bit hash functions, we must prove that the following system of equations has no solutions that consist of only elements in $\{0, 1\}$:

$$A_d X = \frac{d}{4} \begin{bmatrix} 2 & 1 & 1 & \ldots & 1 \end{bmatrix}^T . \tag{13}$$

By counting the number of ones in every row of A_d, we make the observation that $X = \begin{bmatrix} 1 & 1 & \cdots & 1 \end{bmatrix}^T /2$ is always a valid solution. This is the only solution

if A_d is invertible. In Appendix B, we prove that matrices A_d are invertible, by showing their relation to Hadamard matrices. As none of the elements of X are in the set $\{0, 1\}$, there are no m-to-1 bit hash functions that are linear subset regular. □

We now show that if no m-to-1 bit linear subset regular hash functions exist, there exist no m-to-n bit linear subset regular hash functions.

Theorem 4. *There exists no m-to-n bit hash function that is linear subset regular.*

Proof. We show by induction on n. Let $P(n)$ denote the proposition:

There exists no m-to-n bit hash function that is linear subset regular.

The base case $P(1)$ is true by Theorem 3. Let $P(i)$ be true for some $i < n$. Then, we derive the truth table of Table 1.

Table 1 – Truth table for an m-to-i bit hash function h; $\alpha_{j,\ell} \in \{0, 1\}$ $\forall\ j \in \{0, \ldots, 2^m - 1\}$ and $\ell \in \{0, \ldots, i - 1\}$

x				$h(x)$			
x_{m-1}	x_{m-2}	$\cdots$	x_0	α_{i-1}	α_{i-2}	$\cdots$	α_0
0	0	$\cdots$	0	$\alpha_{0,i-1}$	$\alpha_{0,i-2}$	$\cdots$	$\alpha_{0,0}$
0	0	$\cdots$	1	$\alpha_{1,i-1}$	$\alpha_{1,i-2}$	$\cdots$	$\alpha_{1,0}$
$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
0	1	$\cdots$	1	$\alpha_{2^{m-1}-1,i-1}$	$\alpha_{2^{m-1}-1,i-2}$	$\cdots$	$\alpha_{2^{m-1}-1,0}$
1	0	$\cdots$	0	$\alpha_{2^{m-1},i-1}$	$\alpha_{2^{m-1},i-2}$	$\cdots$	$\alpha_{2^{m-1},0}$
1	0	$\cdots$	1	$\alpha_{2^{m-1}+1,i-1}$	$\alpha_{2^{m-1}+1,i-2}$	$\cdots$	$\alpha_{2^{m-1}+1,0}$
$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
1	1	$\cdots$	1	$\alpha_{2^{m-1},i-1}$	$\alpha_{2^{m-1},i-2}$	$\cdots$	$\alpha_{2^{m-1},0}$

Now, if a hash function is linear subset regular then it is

- regular, and
- subset regular under all linear conditions, each of which partitions the domain into two equally-sized sets.

Therefore, if a hash function is not linear subset regular then it is either (*i*) not regular or (*ii*) not subset regular with respect to at least one linear condition. Given case (*ii*), without loss of generality, let us assume that subset regularity does not hold for $x_{m-1} = 0$. Then, in the truth table in Table 1, at least one of the 2^{m-1} i-bit outputs corresponding to $x_{m-1} = 0$ appears more than the expected $2^{m-1}/2^i = 2^{m-i-1}$ times. Again, without loss of generality, let the output $\{0\}^i$

appear $t > 2^{m-i-1}$ times (i.e., say, $\alpha_{j,\ell} = 0 \,\forall\, j \in \{0, \ldots, t-1\}$ and $\ell \in \{0, \ldots, i-1\}$ in Table 1). Then, if we append one bit to each of the t output strings $\{0\}^i$, one of the strings $\{0\}^i||0$ and $\{0\}^i||1$ appears *strictly* more than $2^{m-1}/2^{i+1} = 2^{m-i-2}$ times. Since $\{0\}^i||0$ and $\{0\}^i||1$ should each appear exactly 2^{m-i-2} times when the m-to-$(i+1)$ bit hash function is subset regular under the linear condition $x_{m-1} = 0$, $P(i+1)$ is true when $P(i)$ is true. Given case (i), following a similar line of reasoning, replacing 2^{m-1} with 2^m and recalculating the formulas, we obtain that $P(i) \Rightarrow P(i+1)$. Therefore, by the principle of mathematical induction, $P(n)$ is true. □

Let us again consider a regular hash function with $d = 2^{162}$ and $r = 2^{160}$. If we require this function to be linear subset regular as well, the function must be subset regular for a fraction of $d - 1$ out of all $C(d, d/2)$ possible subsets consisting of half of the domain elements. For $d = 2^{162}$, this fraction evaluates to about $2^{-2^{162}}$. Therefore, by imposing subset regularity for only an extremely small fraction of the possible subsets that we consider, we prove that no linear subset regular functions exist.

In the previous section, we showed that the fraction of subset regular hash functions was negligible. In this section, we obtained an even stronger result: there does not exist a hash function that is regular for more than a negligibly small fraction of subsets of the domain.

Therefore, in the birthday attack, the attacker can always restrict the domain in such a way that the resulting hash function is not regular. This counters Bellare and Kohno's interpretation of why regular functions fare better than random functions against the birthday attack. However, we do not dispute the mathematics of their analysis.

7 Impact on the Birthday Attack

In the previous sections, we showed how unlikely it is that a hash function is regular, if the attacker restricts the inputs to a particular subset. We now use this observation to increase the success probability of the birthday attack against a regular hash function (for the same number of trials), compared to Bellare and Kohno's analysis.

Bellare and Kohno [2, §7.2] see a possibility for the attacker to restrict the domain to a smaller subset of $d = 2r > 10$ elements, and calculate that in this case, $C_h^{\$}(q) > (8/5) \cdot C_h^{\text{reg}}(q)$. From this, they conclude that regular hash functions fare better than random hash functions against the birthday attack. However, Bellare and Kohno's analysis assumes that the attacker restricts the domain in such a way, that D consists of all strings of length $\log_2(r) + 1$ bits.

As Bellare and Kohno already pointed out, $C_h^{\text{reg}}(q)$ approaches $C_h^{\$}(q)$ if the length of the input strings increases. Therefore, to increase the success probability of the birthday attack against a regular hash function with $d = 2r$ (for the same

number of trials q), the attacker can consider long input messages. The attacker will then restrict these long input messages to a set of $d = 2r$ elements, and perform the birthday attack. Therefore, by increasing the length of the input messages (but still restricting the domain points in the birthday attack to $d = 2r$ elements), the success probability of the birthday attack against a regular hash function can be made arbitrarily close to that of a random hash function, for the same number of trials q. This contradicts Proposition 7.4 of [2], which states that if $|D| = 2|R| \geq 10$, then $C_h^{\$}(q) > (8/5) \cdot C_h^{\text{reg}}(q)$ for all q satisfying $2 \leq q \leq 0.1 \cdot r^{1/2}$.

8 Related Work

In 1956, Dumey introduced the concept of (non-cryptographic) hashing [7]. It was proposed as a solution to the dictionary problem. In the dictionary problem, a sequence of operations $\textsc{Insert}(k, x)$, $\textsc{Delete}(k)$ and $\textsc{Lookup}(k)$ are given. They are used to respectively insert, delete and look up key-value pairs (k, x), and are performed on an initially empty table of key-value pairs. The goal is to minimize the time and memory used by these operations.

Let $h' : D' \to R'$ be a hash function, where both the domain $|D'| = d'$ and the range $|R'| = r'$ are finite, and $d' > r'$. The construction known as *chained hashing* is then described as follows. We initialize an array $A[1 \ldots r']$, and let $A[i]$ contain a linked list of all key-value pairs (k, x) for which $h'(k) = i$.

Assume that $r' \mid d'$. For chained hashing, h' is ideally chosen such that every $A[i]$ contains the same number of key-value pairs. This is related to the notion of a regular hash function by Bellare and Kohno, where every hash value has the same number of preimages in the domain D'. If D' is the set of all keys that are added to the table, then the number of key-value pairs that have to be read when either of the three operations are performed, is at most d'/r'. If there exists an $A[i]$ with fewer than d'/r' elements, then there also exists an $A[j]$ where $i \neq j$ with more than d'/r' elements. Therefore, regular hash functions obtain the best performance in the worst-case scenario.

Doing a rigorous analysis of chained hashing is difficult, because the calculations strongly depend on sequence of keys k. For example, by the pigeonhole principle there always exists a sequence of keys k that all map to the same hash value $h'(k)$. Sometimes assumptions are placed on the sequence of keys k, but these may be very difficult (or even impossible) to guarantee in practice. This is also evident from the analysis in our paper.

As a novel solution to the dictionary problem, we mention the universal classes of hash functions proposed by Carter and Wegman [6]. In their paper, it is proposed that h' is chosen uniformly at random, but not from the set of all possible functions. The class of hash functions H' is chosen in such a way, that the average performance (for all $h' \in H'$) for the worst case input is bounded.

More formally, let $h' : D' \to R'$ be a hash function, where both the domain $|D'| = d'$ and the range $|R'| = r'$ are finite, and $d' \geq r'$. A universal class of hash

functions is then defined such that for a randomly chosen $h' \in H'$, the probability that any $h'(x) = h'(y)$ is at most $1/r'$ for any two distinct x and y.

However, not all protocols allow a hash function to be selected uniformly at random from a class of hash functions. In that case, the notion of universal classes of hash functions is not meaningful.

9 Random Functions

Bellare and Kohno showed [2] that several reduced versions of SHA-1 do not behave as regular functions. This indicates that regular functions may not be a suitable theoretical model to analyze the collision resistance of commonly used hash functions. In previous sections, we also made an observation on Bellare and Kohno's claim that regular hash functions fare better than random hash functions against the birthday attack. Based on this, we suggest not to model hash functions as regular functions.

Instead, we propose to model hash functions as random functions when analyzing the complexity of the birthday attack. We agree with Bellare and Kohno that "the design principle of attempting to make hash functions have random behavior [...] is sound and central to security" [2]. We now explain why random functions do not suffer from any of the problems described in this paper.

A random function can be defined as follows:

Definition 4 (Random Function)**.** Let $F : \{0,1\}^* \rightarrow \{0,1\}^n$ be a random function. If $x_i \in \{0,1\}^n$ has not been queried before, the random function chooses y_i uniformly at random from all 2^n range points, and outputs $y_i = F(x_i)$. Otherwise, if $x_i = x_j$ where $j < i$, the random function outputs $y_j = F(x_j) = F(x_i)$.

Unlike for a regular hash function, it is not necessary for a random function to require that the domain consists of a finite number of elements. Also, it is clear from the random function definition, that for any subset of the domain, the range points y_i are chosen randomly and independently from a uniform distribution as well. The statistics of a random function are the same, no matter how the domain points are chosen. Therefore, for a random function, the success probability of the birthday attack does not depend on how the domain points are chosen.

10 Conclusions

The notion of a regular hash function was introduced by Bellare and Kohno at EUROCRYPT 2004, and has subsequently appeared in several research papers. It is defined as a hash function that has the same number of preimages in the domain for every hash value. In their original paper, Bellare and Kohno state that "regular functions fare better than random functions [against the birthday attack]".

We explain that this statement, which until now remained unchallenged, is based on the assumption that the attacker chooses the domain points uniformly at

random. However, Bellare and Kohno note that "there are several variants of [the birthday attack] which differ in the way the [domain] points $x_1, \ldots, x_q$ are chosen." One possible restriction is that domain points correspond to meaningful messages. For example, if messages consist of only ASCII characters, the most significant bit of every character must be zero.

For simplicity, we assumed that the choices of the attacker are restricted to half of the domain points. In that case, we calculate that the probability that the resulting function is still regular under this restriction is very close to zero.

We then attempt to extend the concept of regularity, and require that a hash function is also regular under subsets of the domain. We prove that no such hash function exists, even if we consider only a very small fraction of all possible ways to divide the domain into subsets.

Thus, the attacker can restrict the domain points in the birthday attack in such a way that the resulting hash function is not regular. This is our point of disagreement with Bellare and Kohno's analysis of why regular functions perform better than random functions against the birthday attack.

We show how the success probability of the birthday attack against a regular hash function can be increased (for the same number of trials), compared to Bellare and Kohno's analysis. Our results hold even for a highly restricted domain.

If hash functions are modeled as random functions, the choice of the domain points does not change the success probability of the birthday attack.

Acknowledgments. The authors would like thank the anonymous reviewers for their detailed comments and suggestions.

References

[1] M. Bellare and T. Kohno. Hash Function Balance and Its Impact on Birthday Attacks. In C. Cachin and J. Camenisch, editors, *EUROCRYPT*, volume 3027 of *Lecture Notes in Computer Science*, pages 401–418. Springer, 2004.

[2] M. Bellare and T. Kohno. Hash Function Balance and Its Impact on Birthday Attacks, 2004. `http://cseweb.ucsd.edu/~mihir/papers/balance.html`.

[3] D. M. Bloom. A birthday problem. *American Mathematical Monthly*, 80:1141–1142, 1973.

[4] J. A. Buchmann. *Introduction to Cryptography, Second Edition.* Springer, 2004.

[5] C. Cachin. *Entropy Measures and Unconditional Security in Cryptography.* PhD thesis, ETH Zurich, 1997. Reprint as vol. 1 of *ETH Series in Information Security and Cryptography*, ISBN 3-89649-185-7, Hartung-Gorre Verlag, Konstanz, 1997.

[6] J. L. Carter and M. N. Wegman. Universal Classes of Hash Functions. *Journal of Computer and System Sciences*, 18(2):143–154, April 1979.

[7] A. I. Dumey. Indexing for Rapid Random Access Memory Systems. *Computers and Automation*, 5(12):6–9, December 1956.

[8] Federal Information Processing Standards. FIPS 180-1: Secure Hash Standard. `http://www.itl.nist.gov/fipspubs/fip180-1.htm`, 1995.

[9] W. Feller. *An Introduction to Probability Theory and its Applications.* John Wiley, New York, NY, USA, 1950.

[10] A. Fiat and A. Shamir. How to Prove Yourself: Practical Solutions to Identification and Signature Problems. In A. M. Odlyzko, editor, *CRYPTO*, volume 263 of *Lecture Notes in Computer Science*, pages 186–194. Springer, 1986.

[11] S. Goldwasser and M. Bellare. Lecture Notes on Cryptography. `http://cseweb.ucsd.edu/~mihir/papers/gb.pdf`, 2008.

[12] J. Hadamard. Résolution d'une question relative aux déterminants. *Bulletin des Sciences Mathématiques*, 17:240–246, 1893.

[13] A. Joux. *Algorithmic Cryptanalysis.* Chapman & Hall / CRC, 2009.

[14] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone. *Handbook of Applied Cryptography.* CRC Press, 1996.

[15] National Institute of Standards and Technology. Announcing Request for Candidate Algorithm Nominations for a New Cryptographic Hash Algorithm (SHA-3) Family. *Federal Register*, 27(212):62212–62220, November 2007. `http://csrc.nist.gov/groups/ST/hash/documents/FR_Notice_Nov07.pdf`.

[16] T. S. Nunnikhoven. A birthday problem solution for nonuniform birth frequencies. *The American Statistician*, 46(4):270–274, Nov. 1992.

[17] R. S. Ødegård and D. Gligoroski. On the Randomness and Regularity of Reduced EDON-$\mathcal{R}$ Compression Function. Cryptology ePrint Archive, Report 2009/234, 2009. `http://eprint.iacr.org/`.

[18] J.-J. Quisquater and J.-P. Delescaille. How Easy is Collision Search? Application to DES (Extended Summary). In *EUROCRYPT*, pages 429–434, 1989.

[19] D. R. Stinson. *Introduction to Cryptography, First Edition.* CRC-Press, 1995.

[20] D. R. Stinson. *Introduction to Cryptography, Second Edition.* Chapman & Hall / CRC, 2002.

[21] J. J. Sylvester. Thoughts on inverse orthogonal matrices, simultaneous sign-successions, and tessellated pavements in two or more colours, with applications to Newton's rule, ornamental tile-work, and the theory of numbers. *Philosophical Magazine*, 34:461–475, 1867.

[22] P. C. van Oorschot and M. J. Wiener. Parallel Collision Search with Cryptanalytic Applications. *J. Cryptology*, 12(1):1–28, 1999.

[23] R. von Mises. Über Aufteilungs- und Besetzungswahrscheinlichkeiten. (On Partitioning and Occupation Probabilities). *İstanbul Üniversitesi Fen Fakültesi Mecmuasi*, 4:145–163, 1939.

[24] D. Wagner. A Generalized Birthday Problem. In M. Yung, editor, *CRYPTO*, volume 2442 of *Lecture Notes in Computer Science*, pages 288–303. Springer, 2002.

[25] H. Yoshida, D. Watanabe, K. Okeya, J. Kitahara, H. Wu, Ö. Küçük, and B. Preneel. MAME: A Compression Function with Reduced Hardware Requirements. In P. Paillier and I. Verbauwhede, editors, *CHES*, volume 4727 of *Lecture Notes in Computer Science*, pages 148–165. Springer, 2007.

[26] G. Yuval. How to Swindle Rabin. *Cryptologia*, 3:187–189, 1979.

A Linear Subset Regularity for 3-to-1 Bit Hash Functions

Here, we will attempt to construct a 3-to-1 bit linear subset regular hash function $h(x)$. Let the input x be a binary string, resulting from the concatenation of the three input bits, such that $x \leftarrow x_2 \parallel x_1 \parallel x_0$. We set $h(000) = A$, where A can be either 0 or 1. The other output symbol will then be denoted by B. We now consider three cases, as shown in Table 2.

- **Case 1:** Assume $h(001) = A$. Subset regularity with respect to x_2 then leads to $h(010) = h(011) = B$. Furthermore, subset regularity with respect to x_1 results in $h(100) = h(101) = B$. For h to be regular, we must have $h(110) = h(111) = A$. However, we now find that restricting the inputs to $x_2 \oplus x_1 = 0$ results in a constant function.

- **Case 2:** Assume $h(001) = B$ and $h(010) = A$. Subset regularity with respect to x_2 then leads to $h(011) = B$. Furthermore, subset regularity with respect to x_0 results in $h(100) = h(110) = B$. For h to be regular, we must have $h(101) = h(111) = A$. However, we now find that restricting the inputs to $x_2 \oplus x_0 = 0$ results in a constant function.

Table 2 – Constructing a 3-to-1 bit linear subset regular hash function $h(x)$, where $x \leftarrow x_2 \parallel x_1 \parallel x_0$; the values in bold were set initially, the others are derived from the linear subset regular conditions

Case 1

x_2	x_1	x_0	$h(x)$
0	0	0	***A***
0	0	1	***A***
0	1	0	B
0	1	1	B
1	0	0	B
1	0	1	B
1	1	0	A
1	1	1	A

Case 2

x_2	x_1	x_0	$h(x)$
0	0	0	***A***
0	0	1	***B***
0	1	0	***A***
0	1	1	B
1	0	0	B
1	0	1	A
1	1	0	B
1	1	1	A

Case 3

x_2	x_1	x_0	$h(x)$
0	0	0	***A***
0	0	1	***B***
0	1	0	***B***
0	1	1	A
1	0	0	B
1	0	1	A
1	1	0	A
1	1	1	B

- **Case 3:** Assume $h(001) = B$ and $h(010) = B$. Subset regularity with respect to x_2 then leads to $h(011) = A$. Furthermore, subset regularity with respect to $x_1 \oplus x_0$ results in $h(100) = h(111) = B$. For h to be regular, we must have $h(101) = h(110) = A$. However, we now find that restricting the inputs to $x_2 \oplus x_1 \oplus x_0 = 0$ results in a constant function.

Consequently, there are no 3-to-1 bit hash functions that are linear subset regular. Also note that imposing all but one linear subset regular condition in Table 2 leads to an affine hash function. We found by exhaustive search that all 3-to-1 bit hash functions where all but one linear subset regular conditions are imposed, result in affine hash functions.

B Calculating the Inverses of Matrices A_d

In this section, we prove that the matrices A_d of Theorem 3 are invertible, by showing their relation to Hadamard matrices. We give an explicit formula for their inverses.

Hadamard matrices are square matrices of which all elements are either 1 or -1. They were initially proposed by Sylvester [21]. Hadamard [12] later showed that these matrices are the solution to his maximum determinant problem. An $d \times d$ Hadamard matrix H_d can be defined a matrix satisfying

$$H_d H_d^T = dI_d \ , \tag{14}$$

where I_d denotes the $d \times d$ identity matrix.

If d is a power of two, Sylvester [21] proposed the following construction for H_d:

$$H_1 = [1]\,, \tag{15}$$

$$H_d = \begin{bmatrix} H_{d/2} & H_{d/2} \\ H_{d/2} & -H_{d/2} \end{bmatrix}, \text{ for } 1 \leq \log_2(d) \in \mathbb{N}\ . \tag{16}$$

Let J_d be the $d \times d$ matrix where every element is equal to one. Matrix K_d is the $d \times d$ matrix where every element of the first column is 1, and all other elements are zero. Note that $K_d K_d^T = J_d$. Matrices A_d of Theorem 3 satisfy the equation $H_d = 2A_d - J_d$. We now show that matrices A_d are invertible, and calculate their inverse. Using (14), we obtain

$$\begin{aligned} (2A_d - J_d)(2A_d - J_d)^T &= dI_d \qquad (17)\\ \Leftrightarrow (2A_d - J_d)(2A_d^T - J_d^T) &= dI_d \\ \Leftrightarrow 4A_dA_d^T - 2A_dJ_d^T - 2J_dA_d^T + J_dJ_d^T &= dI_d \\ \Leftrightarrow 4A_dA_d^T - d(K_d^T + J_d) - d(K_d + J_d^T) + dJ_d &= dI_d \\ \Leftrightarrow 4A_dA_d^T - dK_d^T - dK_d - dJ_d^T &= dI_d \\ \Leftrightarrow 4A_dA_d^T - dK_d^T - dK_d - dK_dK_d^T &= dI_d \\ \Leftrightarrow 4A_dA_d^T &= d(K_d + I_d)(K_d + I_d)^T \qquad (18) \end{aligned}$$

As $K_dK_d = K_d$, we have

$$(K_d + I_d)(I_d - K_d/2) = K_d - K_dK_d/2 + I_d - K_d/2 = I_d\ . \tag{19}$$

Therefore, $(K_d + I_d)^{-1} = I_d - K_d/2$. From (18), we then obtain

$$A_dA_d^T(2I_d - K_d)^T(2I_d - K_d) = dI_d\ . \tag{20}$$

This equation shows that A_d is invertible, and that its inverse is given by

$$A_d^{-1} = \frac{1}{d}A_d^T(2I_d - K_d)^T(2I_d - K_d)\ . \tag{21}$$

Publication Chapter

Algebraic Techniques in Differential Cryptanalysis Revisited

Publication Data

Meiqin Wang, Yue Sun, Nicky Mouha, and Bart Preneel. Algebraic Techniques in Differential Cryptanalysis Revisited. In Udaya Parampalli and Philip Hawkes, editors, *ACISP*, volume 6812 of *Lecture Notes in Computer Science*, pages 120–141. Springer, 2011.

Contributions

- Did a complete rewrite of an earlier draft. All results were extensively verified, and corrections were made where necessary. Most computer experiments were performed by Yue Sun.

Algebraic Techniques in Differential Cryptanalysis Revisited*

Meiqin Wang[1,2,3,†], Yue Sun[1], Nicky Mouha[2,3,‡], and Bart Preneel[2,3]

[1] School of Mathematics, Shandong University, Jinan 250100, China
[2] Department of Electrical Engineering ESAT/SCD-COSIC,
Katholieke Universiteit Leuven, Kasteelpark Arenberg 10, B-3001 Heverlee, Belgium.
[3] Interdisciplinary Institute for BroadBand Technology (IBBT), Belgium.
mqwang@sdu.edu.cn

Abstract. At FSE 2009, Albrecht *et al.* proposed a new cryptanalytic method that combines algebraic and differential cryptanalysis. They introduced three new attacks, namely Attack A, Attack B and Attack C. For Attack A, they explain that the time complexity is difficult to determine. The goal of Attacks B and C is to filter out wrong pairs and then recover the key. In this paper, we show that Attack C does not provide an advantage over differential cryptanalysis for typical block ciphers, because it cannot be used to filter out any wrong pairs that satisfy the ciphertext differences. Furthermore, we explain why Attack B provides no advantage over differential cryptanalysis for PRESENT. We verify our results for PRESENT experimentally, using both PolyBoRi and MiniSat. Our work helps to understand which equations are important in the differential-algebraic attack. Based on our findings, we present two new differential-algebraic attacks. Using the first method, our attack on 15-round PRESENT-80 requires 2^{59} chosen plaintexts and has a worst-case time complexity of $2^{73.79}$ equivalent encryptions. Our new attack on 14-round PRESENT-128 requires 2^{55} chosen plaintexts and has a worst-case time complexity of $2^{112.83}$ equivalent encryptions. Although these attacks have a higher time complexity than the differential attacks, their data complexity is lower.

Keywords: Differential-Algebraic Attack, Block Cipher, PRESENT

*This work was supported in part by the Research Council K.U.Leuven: GOA TENSE, the IAP Program P6/26 BCRYPT of the Belgian State (Belgian Science Policy), and in part by the European Commission through the ICT program under contract ICT-2007-216676 ECRYPT II.

†This author is supported by 973 Project (No.2007CB807902), NSFC Projects (No.61070244 and No.60931160442), Outstanding Young Scientists Foundation Grant of Shandong Province (No.BS2009DX030), IIFSDU Project (No. 2009TS087).

‡This author is funded by a research grant of the Institute for the Promotion of Innovation through Science and Technology in Flanders (IWT-Vlaanderen).

1 Introduction

Differential cryptanalysis [6, 7] is one of classic cryptanalytic methods for block ciphers. Resistance against differential cryptanalysis is a typical design criterion for new block ciphers. Algebraic cryptanalysis is a general method to attack ciphers. It has been widely used to cryptanalyze many primitives such as stream ciphers [12, 15], multivariate cryptosystems [18] and in particular block ciphers [13, 14, 16, 21]. The basic idea of algebraic cryptanalysis is to express the block cipher as a large multivariate polynomial system of equations. The secret key of the cipher is the solution of this system of equations. If the system is very sparse, overdefined or structured, it may be solved faster than a generic non-linear system of equations. By solving the system of equations for the block cipher, the key can be recovered with only a few plaintext-ciphertext pairs.

There are several methods to solve these systems of equations, such as computing a Gröbner basis or using a SAT solver. To compute a Gröbner basis, PolyBoRi [10] can be used. MiniSat [17] is a fast SAT solver. The advantage of computing a Gröbner basis is that useful equations can be generated, but this computation is typically slower than using a SAT solver and can more easily run out of memory.

However, the feasibility of algebraic cryptanalysis against block ciphers still remains a source of speculation. The main problem is that the size of the corresponding algebraic system is so large (thousands of variables and equations) that it seems infeasible to correctly predict the complexity of solving such polynomial systems. Therefore, algebraic cryptanalysis has so far had limited success in targeting modern block ciphers.

Recently, some works combining statistical cryptanalysis and algebraic cryptanalysis were presented [1, 3, 4, 19, 25]. Specifically, the combination of differential cryptanalysis and algebraic cryptanalysis appears to offer an advantage in reducing the data complexity. In [1,3], Albrecht *et al.* propose new differential-algebraic cryptanalytic methods, which they refer to as Attack A, Attack B and Attack C. In order to describe them, let p denote the probability of the r-round differential characteristic for an N-round block cipher.

In Attack A, the system of equations consists of the equations of the plaintext bits, ciphertext bits, and subkey bits, the equations of the key schedule, and the linear equations resulting from the differential characteristic and the filter equations of the last $(N - r)$ rounds (i.e. the equations that must hold if the output difference after round r holds). Attack A recovers the key by solving this system of equations for each of the about $1/p$ plaintext-ciphertext pairs.

In Attack B, the same system of equations is used. The longest time to find that the system of equations is inconsistent, is measured. If this time is exceeded, a right pair is found with a high probability.

In Attack C, the system of equations only consists of the filter equations after r rounds for an r-round differential and the key schedule algorithm after r rounds. The conditions resulting from the differential characteristic and the conditions

from the plaintext to the corresponding ciphertext are omitted in Attack C. The goal of Attack C in [3] is to filter out wrong pairs by solving the system of equations using tools such as PolyBoRi or MiniSat, and to use the remaining right pair to recover the subkey bits.

In differential cryptanalysis, the filtering process can only filter out the wrong pairs according to the difference values of the ciphertext pairs. That is, after the filtering process, a lot of wrong pairs may still remain, which may increase the time complexity to recover the key in the differential attack. However, in Attack B and Attack C, Albrecht *et al.* claim that the right pairs can be identified with a good probability if the equations after the r-th round of the differential characteristic are inconsistent. They claim that with their technique, the time complexity will be lower than in the standard differential attack. Their work received a lot of attention in the cryptographic community [5, 8, 11, 20, 22], because it gives hope for the combination of a statistical attack and an algebraic attack.

In this paper, we will revisit the differential-algebraic attack given by Albrecht *et al.*, which they applied to PRESENT [9]. We find that Albrecht's method cannot filter out most of the wrong pairs satisfying the ciphertexts differences. However, we will show that wrong pairs that do not satisfy the ciphertext differences, can easily be filtered out without the algebraic method. Using [3, 4], it is not possible to filter out more wrong pairs than using differential cryptanalysis.

Firstly, we show that Attack C typically cannot be used to filter out wrong pairs that do not satisfy the difference values of the ciphertexts to improve the differential cryptanalysis. Secondly, we verify using PolyBoRi and MiniSat2 that Attack B does not improve the current differential results for the PRESENT block cipher. The reason is that there are too few usable equations in the system of equations to derive an inconsistency for the wrong pairs or to find a solution for the right pairs. Based on our findings, we introduce two new methods that can more reliably use the right pairs to solve the right key within an acceptable time. For wrong pairs, no solution will be produced. One method is to fix certain key bits in the system of equations. This will allow an inconsistency to be derived faster. Another method is to use more than one plaintext-ciphertext pair to construct the system of equations.

We apply our attack methods to a reduced-round PRESENT block cipher. With the first method, we attack 15-round PRESENT-80 with 2^{59} chosen plaintexts and $2^{73.79}$ equivalent encryptions in the worst case. The 2R-differential attack on 15-round PRESENT-80 has a data complexity of more than 2^{59} and a time complexity of less than 2^{62} memory accesses. Therefore, the time complexity of the differential-algebraic attack for PRESENT-80 is much larger than that of the differential attack, but the data complexity is lower and the key does not have to be the same for every pair. If the number of chosen plaintext pairs that the attacker can obtain is limited, the algebraic-differential attack might be the only feasible attack. Note, however, that more rounds can be attacked in the case of PRESENT-80 using differential cryptanalysis (16 rounds instead of 15 rounds). We also provide a new attack on 14-round PRESENT-128 with a data complexity

of 2^{55} chosen plaintexts and a worst-case time complexity of $2^{112.83}$ equivalent encryptions.

With our second method, the time complexity will be larger than with the first method for 15-round PRESENT-80. It is an open question whether the second method can offer an improvement for other block ciphers.

Our work also points out which equations are important in the differential-algebraic attack. With pure algebraic cryptanalysis, a 5-round PRESENT block cipher [14, 21] can be attacked. Compared to this result, our differential-algebraic attack can attack more rounds, but the data complexity will be higher than that for the pure algebraic attack.

This paper is organized as follows. Section 2 describes Albrecht's differential-algebraic attack. In Sect. 3, we show why Attack C cannot filter out more wrong pairs than differential cryptanalysis for most block ciphers. We verify using PolyBoRi and MiniSat2 that Attack B cannot improve the differential cryptanalysis of the PRESENT block cipher. In Sect. 4, we present two methods that can be used to successfully solve the right key with the right pairs. Our attack methods are then applied to a reduced-round PRESENT block cipher. We conclude the paper in Sect. 5.

2 Description of Albrecht's Differential-Algebraic Attack

In [1, 3], Albrecht *et al.* proposed three types of attacks that combine algebraic techniques with differential cryptanalysis. They are referred to as Attack A, Attack B and Attack C. We now describe these three types of attacks.

Attack A.

For an r-round differential characteristic $\Delta = (\delta_0, \delta_1, \ldots, \delta_r)$, the probability of the differential characteristic is denoted by p. For a pair of plaintexts (P', P''), where $P' \oplus P'' = \delta_0$, and the corresponding ciphertexts (C', C''), two systems of equations F' and F'' are constructed under the same encryption key K. With the differential characteristic, the following linear equations are constructed:

$$X'_{i,j} \oplus X''_{i,j} = \Delta X_{i,j} \rightarrow \Delta Y_{i,j} = Y'_{i,j} \oplus Y''_{i,j} \ ,$$

where $X'_{i,j}$ and $X''_{i,j}$ are the j-th bit of the input to the S-box layer in round i for the systems F' and F'' respectively. The corresponding output bits are $Y'_{i,j}$ and $Y''_{i,j}$. The values resulting from the differential characteristic are $\Delta X_{i,j}$ and $\Delta Y_{i,j}$. The linear expressions corresponding to bits of active S-boxes hold with some non-negligible probability. For the non-active S-boxes, the following linear relations also hold with non-negligible probability:

$$X'_{i,j} \oplus X''_{i,j} = 0 = Y'_{i,j} \oplus Y''_{i,j} \ .$$

If the r-round differential characteristic is used to recover the key for N rounds, the differences from the $(r+1)$-th round to the N-th round can be derived from the output difference of the r-th round. Theses differences after the r-th round are described by equations. Attack A combines the two systems of equations F' and F'', the above linear relations resulting from the differential characteristic and the equations from the difference values after round r to produce the system of equations $\overline{F}$ that holds with probability p. If about $1/p$ systems corresponding to $1/p$ pairs of plaintext-ciphertext can be solved, a right pair is expected to be found which can then be used to obtain the right key. However, the time complexity to solve the system about $1/p$ times may be very high.

Attack B.

Attack B uses the same system equations as Attack A to filter out the wrong pairs. In a differential attack, the ciphertext difference values are commonly used to filter out wrong pairs. However, in Attack B, by measuring the time t it maximally takes to find that the system is inconsistent, it is assumed that a right pair has been identified with high probability if a time t has elapsed without finding an inconsistency. More specifically, Attack B assumes that $\Delta Y_{1,j}$ holds with a high probability after time t has elapsed. With the remaining pairs, the subkey bits involved in the active S-boxes in the first round can be recovered. An alternative form of Attack B is to recover key bits from the last round. It is assumed that if time t passes for a given plaintext-ciphertext pair, a right pair has been found. In this case, some subkey bits in the last rounds will be fixed, and then it is checked whether time t still passes without contradiction. The time to find an inconsistency or a reduced-round PRESENT block cipher was measured in Appendix C of [3].

Attack C.

In Attack C, the differential is used instead of the differential characteristic as in Attack B. If the r-round differential $\delta_0 \to \delta_r$ is used to recover the key for N rounds, the system of equations only consists of the equations resulting from the round functions from round $(r+1)$ to round N, the relations for the difference values from the $(r+1)$-th round to the N-th round, and the equations of key schedule from the $(r+1)$-th round to the N-th round. In this system of equations, there are no equations to restrict the relations between the plaintext and the corresponding ciphertext, and there are no equations for the difference values from the first round to the r-th round. By solving the system of equations and waiting for a fixed time t, a contradiction can be found in the system of equations. If one tested pair did not produce a contradiction after a fixed time, it is assumed to be a right pair satisfying the differential. Then with the right pair, the partial information for the subkey bits can be recovered. Appendix D in [3] measured the time to find an inconsistency for a reduced-round PRESENT block cipher. Based on this measured time, attacks on 16-round PRESENT-80, 17, 18 and 19 rounds of

PRESENT-128 block cipher were given in [1, 3].

3 Inapplicability of Albrecht *et al.*'s Attacks

3.1 Inapplicability of Attack C

In this section, we will show that Attack C typically cannot be used to filter out the wrong pairs satisfying the difference values of the ciphertexts. Therefore, the right pairs cannot be identified and the key cannot be recovered. Moreover, Attack C can not filter out more wrong pairs than differential cryptanalysis to improve the differential cryptanalysis. As in the previous description, the system of equations in Attack C consists of the equations resulting from the round functions from round $(r+1)$ to round N, the relations resulting from the difference values from the $(r+1)$-th round to the N-th round, and the equations of key schedule from the $(r+1)$-th round to the N-th round. Let C'_i and C''_i be the i-th bit of ciphertext pair C' and C'' respectively, and ΔC_i is the i-th bit of the difference value of ciphertext pair C' and C''. We then classify these equations into three groups, Group A, Group B and Group C.

Group A.

The linear equations resulting from the difference values of ciphertexts corresponding to the non-active S-boxes in the last round are

$$\Delta C_i = C'_i \oplus C''_i = 0 \ ,$$

where the i-th bit position corresponds to an output bit of any non-active S-box.

Group B.

The equations resulting from the difference values of ciphertexts corresponding to the active S-boxes in the last round are

$$\begin{aligned}(\Delta C_{i_1} \parallel \Delta C_{i_2} \parallel \cdots \parallel \Delta C_{i_a}) &= \left(C'_{i_1} \parallel C'_{i_2} \parallel \cdots \parallel C'_{i_a}\right) \oplus \left(C''_{i_1} \parallel C''_{i_2} \parallel \cdots \parallel C''_{i_a}\right) \\ &= \delta_N, \delta_N \in \Gamma_N \ ,\end{aligned}$$

where $i_1, i_2, \ldots, i_a$ correspond to output bits of the active S-boxes, and Γ_N is the set of the ciphertext difference values.

Group C.

The remaining equations are the equations resulting from the round functions from round $(r+1)$ to round N, the relations resulting from the difference values from the $(r+1)$-th round to the $(N-1)$-th round, and the equations of key schedule

from the $(r+1)$-th round to the N-th round.

If a plaintext-ciphertext pair satisfies all the equations in Group A, Group B and Group C, it must be a right pair for the given differential. In the differential attack, the wrong pairs that do not satisfy the equations in Group A and Group B are easy to filter out using a look-up table combined with a time-memory trade-off. Because the equations in Group C involve unknown subkey bits, they cannot easily be used to filter out the remaining wrong ciphertext pairs after the filtering process with the ciphertext differences. In Attack C, Albrecht *et al.* wish to measure the maximum time t to identify a pair as a wrong pair with all the equations in Group A, B and C. In fact, the equations in Group A and Group B can easily be used to find a contradiction because they are only related to the ciphertext difference values. For a typical block cipher, it is impossible to find contradictions for the equations in Group C. To understand why this is the case, we claim the following.

Claim 1. *If there is a wrong ciphertext pair that satisfies all the equations in Group A and Group B but does not satisfy the equations in Group C, it is impossible for a typical block cipher to find a contradiction for the equations in Group C.*

Proof. We consider a block cipher based on a substitution-permutation network (SPN). For other structures (Feistel, Generalized Feistel,...), a similar proof can be given. We assume that the difference value of the ciphertext pair satisfies the equations in Group A and Group B, but does not satisfy the equations in Group C. First, we will prove Claim 1 for a 1R-attack and extend the proof to an sR-attack[4] $(s = 1, 2, 3, \ldots)$.

In a 1R-attack, the wrong ciphertext pair satisfies the output difference values of all non-active and active S-boxes in the last round, but does not satisfy the input difference of some active S-boxes in the last round. In most SPN block ciphers, after the S-box layer in the last round, the whitening subkeys will be XORed.

Let us introduce the shortened notation

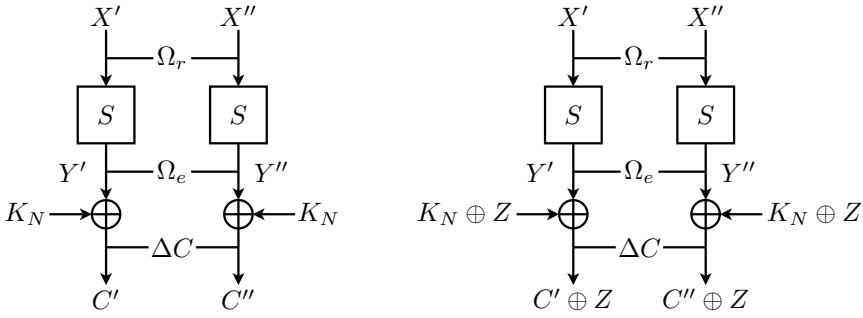
$$X'_i \leftarrow X'_{i,j_1} || X'_{i,j_2} || \ldots || X'_{i,j_m} \ ,$$

where $X'_{i,j}$ is the j-th bit of the input to the S-box layer in round i. We can then describe the round function for the last round as follows:

$$\begin{aligned} Y'_N &= S[X'_N] \ , \ C'_N = Y'_N \oplus K_N \ , \\ Y''_N &= S[X''_N] \ , \ C''_N = Y''_N \oplus K_N \ , \end{aligned}$$

where X'_N and X''_N are the inputs of the S-box layer S in the last round for the system F' and F'' respectively, and Y'_N and Y''_N are the corresponding outputs. The values C'_N and C''_N are the ciphertext bits, and K'_N is the whitening subkey in the last round.

[4] An sR-attack means that the r-round differential is used to recover the key for $(r+s)$ rounds of the block cipher. We require in this paper that $s \ll N$, which is the case for typical differential attacks.



(a) Right pair, right key (b) Wrong pair, wrong key

Figure 1 – It is not possible to detect that $(C' \oplus Z, C'' \oplus Z)$ is a wrong pair (see Claim 1).

We now consider Fig. 1. Under the right key, the wrong ciphertext pair $(C' \oplus Z, C'' \oplus Z)$ will result in the output difference of the S-box Ω_e and the input difference of the S-box Ω_w, however, the right pair (C', C'') will result in the output difference and the input difference for the S-box as Ω_e and Ω_r respectively. As the subkey bits in the above equations are unknown variables, we will solve the following system of equations,

$$X'_N \oplus X''_N = \Omega_r.$$

We can obtain

$$S^{-1}[Y'_N] \oplus S^{-1}[Y''_N] = \Omega_r,$$

where S^{-1} denotes the inverse S-boxes Layer. Then we have

$$S^{-1}[C'_N \oplus K_N] \oplus S^{-1}[C''_N \oplus K_N] = \Omega_r \ .$$

Because the right pair always can produce the difference from $\Omega_r \mapsto \Omega_e$ for the active S-boxes, there is at least one pair of input values (X'_r, X''_r) and the corresponding output values (Y'_r, Y''_r) satisfying the following equations:

$$X'_r \oplus X''_r = \Omega_r, \ \ Y' \oplus Y'' = \Omega_e \ .$$

We have

$$S^{-1}[Y'_r] \oplus S^{-1}[Y''_r] = X'_r \oplus X''_r = \Omega_r.$$

For the wrong pair $(C' \oplus Z, C'' \oplus Z)$, let the whitening subkey in the last round satisfy the following equations:

$$C'_N \oplus Z \oplus K_N = Y'_r \ , \ \ C''_N \oplus Z \oplus K_N = Y''_r \ .$$

The resulting wrong whitening subkey $K_N \oplus Z$ in the last round can make the wrong pair $(C' \oplus Z, C'' \oplus Z)$ produce the right input difference Ω_r, so the wrong

pair $(C' \oplus Z, C'' \oplus Z)$ cannot be filtered out with the system of equations in the last round.

The proof for 1R-attack is helpful to understand the idea. The analysis of the sR-attack works in a similar way. As stated by Biham and Shamir [7] (and similarly by Selçuk [23]):

"Each surviving pair suggests several possible values for [the subkey] bits. Right pairs always suggest the correct value for [the subkey] bits (along with several wrong values), while wrong pairs suggest random values [for the subkey bits]."

This statement is true for typical block ciphers. Therefore, any remaining wrong pair must produce some solutions for the subkey satisfying the difference values in the last s-round. The solution may be the right subkey or the wrong subkey. Thus, it is impossible for most block ciphers to produce a contradiction for the sR-attack in the above s-round equations.

The equations for the key schedule may lead to a contradiction in Group C for the derived subkey value for the last s rounds, but the number of the subkey bits involved in the last s rounds is usually not large enough to produce a contradiction, assuming the key schedule is random. However, assume that the equations for the key schedule result in a contradiction for the subkey values of the last s rounds. Then, this contradiction holds for all values of the subkeys. That is, the contradiction is independent of the subkey values. The contradiction must be a contradiction on the difference of the ciphertext pair: a contradiction on the values of the ciphertext pair cannot appear because the ciphertext is calculated as $C = Y_N \oplus K_N$. Therefore, this contradiction can be included into Group A or Group B. Because the differential cryptanalysis attack uses the equations of Group A and Group B to filter the ciphertext values, an inconsistency in the key schedule does not improve the differential attack. □

In order to verify Claim 1, we tested the filtering time for different values of N and r of the PRESENT block cipher. In our tests, we constructed wrong ciphertext pairs that only satisfy the equations in Group A and Group B, but do not satisfy the equations in Group C when evaluated on the correct key. We used the source code provided by Albrecht [2] to apply Attack C with PolyBoRi-0.6 and MiniSat2. We performed a Gröbner basis computation to generate the filtering equations from the $(r+1)$-th round to the $(r+4)$-th round for the differential characteristic $(2 \leq r \leq 14)$ for PRESENT-80. These filtering equations can speed up the procedure of producing the contradiction.

However, there is no contradiction for any ciphertext pair with PolyBoRi-0.6 after six hours of computation. MiniSat2 always obtained the wrong solution for the key. In Table 1, we list these test results. For the wrong pairs under the right key, the wrong solution can be obtained within t seconds. We tested 20 wrong pairs for different values of r and N, and list one example of a wrong pair (P', P'') and the corresponding right key K. Due to space limitations, we only present the

difference values for the wrong pair in the last row of Table 1 and the differential characteristic for the right pair in Table 2. In Table 2, the output difference for the wrong pair of the r-th ($r = 12$) round is not equal to the output difference of the characteristic, but the output difference of the 13-th round is equal to the output difference of the characteristic. Therefore, this is a wrong pair.

At the same time, we construct the wrong ciphertext pairs for PRESENT-80 which do not satisfy the equations in any Group, the contradiction can be produced quickly and the filtering time is listed in Table 3. In addition, we construct some wrong ciphertext pairs that only satisfy the equation in Group A, the time to produce the contradiction is listed in Table 4. Moreover, we use a look-up table combined with a time-memory trade-off in differential cryptanalysis to filter out these pairs. As a result, our filter is more efficient than Attack C.

The computer we used is an IBM X3950 M2 with a CPU clock frequency of 2.4 GHz and 64GB RAM. From Tables 3 and 4, our test time with PolyBoRi approaches the corresponding time in Appendix D of [3], but our tested time with MiniSat2 is greater. The main reason is that our CPU is not same as Albrecht's. However, we can deduce that the wrong pairs Albrecht *et al.* used are wrong pairs that do not satisfy the equations in Group A or Group B, so they did not filter out wrong pairs that do satisfy the equations in Group A and Group B. Furthermore, even if Attack C is used as a filter for wrong pairs that do not satisfy the equations in Group A and Group B, its efficiency is much lower than the filter used in differential cryptanalysis. This shows that Attack C does not provide an advantage over differential cryptanalysis for most block ciphers.

Using Group A and Group B in a Differential Attack.

We now clarify in more detail how the equations of Group A and Group B can be used in a differential attack. We consider two types of differential attacks:

(a) By generating a table of all possible ciphertext differences (corresponding to all solutions to the equations of Group A and Group B), wrong pairs can easily be filtered out. Because key counters will be used for the subkey bits corresponding to the active S-boxes, the number of output differences is less than the number of key counters required. Therefore, the table of all possible ciphertext differences provides only a relatively small overhead.

(b) In the filtering process, for each pair of ciphertexts (C', C''), a table is made of all possible input differences for the last round. This table does not depend on the value of the subkey bits in the last round. If we do not find a valid input difference for a particular pair of ciphertexts, this pair is identified as a wrong pair (i.e. it does not satisfy the equations of Group A and Group B). In this way, it is only necessary to make table of all input differences, and not all ciphertext differences. Typically, the table of all input differences should be small. For the remaining pairs, subkey bits in the last round will be guessed

(instead of using key counters), to filter out pairs. For a wrong key, no pairs will remain, but the right pair will remain for the right key.

Note that (b) is in fact a time-memory trade-off applied to (a). In both (a) and (b), if output differences are invalid for some active S-boxes, they can be filtered using smaller tables. Then, the table that is described in (a) and (b) will be used to filter out the remaining pairs. In the next paragraph, we describe in detail how (a) can be used for a 2R attack on PRESENT. To construct a filter for a 3R and 4R attack on PRESENT, (b) can be used.

Relation to the Work of [4].

The equation system that Albrecht *et al.* set up in [4], is similar to the system of [3], except that the ciphertext bits (C_i' and C_i'') are variables instead of fixed values. This equation system is used to compute a Gröbner basis for PRESENT up to degree $D = 3$ using PolyBoRi. Polynomials that contain non-ciphertext variables are removed.

The resulting equations are used as a first filter for the ciphertext pairs. Albrecht *et al.* estimate the probability p_1 that a random ciphertext pair passes the first filter as $p_1 \approx 2^{-50.669}$ for a 2R-attack on PRESENT-80 and PRESENT-128. Afterwards, [4] uses Attack C to filter out the remaining pairs. They estimate the total filtering probability $p_2 \approx 2^{-51.669}$ for PRESENT-80 and $p_2 \approx 2^{-51.361}$ for PRESENT-128.

For a 2R-attack on PRESENT, it is straightforward to write a fast program to compute the total number of ciphertext differences. We find that $11664 \approx 2^{13.51}$ ciphertext differences are possible, and store them in a small table. This results in the accurate filtering probability of $p_a = 2^{13.51}/2^{64} = 2^{-50.49}$ for both PRESENT-80 and PRESENT-128. When we derive the probability of p_1 ourselves, using the equations in [4, Fig. 2], we find that $p_1 = p_2 = p_a = 2^{-50.49}$. This confirms our result, and shows that the calculation of p_1 and p_2 in [4] is not correct. The accurate filtering probability p_a is slightly lower than the probability of the rough filter used by Wang [24].

By storing the output differences in a small table, we can easily filter out the wrong ciphertext pairs without using the algebraic method. Furthermore, we calculate that the reinterpretation of Attack C in [4] as a technique to filter ciphertext differences, does not result in a better filter. Therefore, Attack C does not provide an advantage over differential cryptanalysis in the case of a 2R-attack on PRESENT.

For a 3R-attack and a 4R-attack on PRESENT, we used a look-up table combined with a time-memory trade-off to filter out 1000 randomly generated wrong pairs. We note that although the filtering probability of our filter and Attack C is same, our filter is much faster than Attack C.

3.2 Inapplicability of Attack B to PRESENT

Attack B involves two other types of equations, besides the equations in Group A, Group B and Group C in Attack C. The first type of equations is the linear equations derived from the difference values from round 1 to round r, and the second type of equations is the round functions and the key schedule algorithm from round 1 to round r. In this way, the restriction from the plaintext to the corresponding ciphertext was added. Although we cannot show that Attack B does not provide an advantage over differential cryptanalysis for any block cipher, we make the following two observations for Attack B:

Observation 1.

If N approaches the maximum number of rounds that can be attacked with a pure algebraic attack, the linear equations for the inner rounds and the round functions restricting the relation between the plaintext and the ciphertext are all usable to solve the system of equations. There are three possible subcases:

1. If the key size is much larger than the block size, for a wrong pair, the probability that a solution can be found for the key in the system of equations is non-negligible. In this way, there is a non-negligible probability that a contradiction for the wrong pairs cannot be produced. Attack B will likely fail.

2. If the key size is smaller than the block size, for a wrong pair, the probability that no solution can be found for the key in the system of equations is high. In this way, the contradiction for the wrong pairs can be produced and the right solution for the right pair can be found with a high probability. Attack B is likely to succeed.

3. If the key size approaches the block size, Attack B can either succeed or fail.

Observation 2.

If N is much larger than the maximum number of rounds that can be attacked with a pure algebraic attack, the linear equations for the inner rounds and the round functions and the key schedule algorithm for the inner rounds are not crucial to solve the system of equations. Only the equations for the outer rounds are relevant. We consider two subcases.

1. If there are few active S-boxes in the outer rounds, the restriction conditions are so few that a contradiction will be produced with low probability. Attack B will likely fail.

2. If there are many active S-boxes in the outer rounds, there are enough restriction conditions to derive a contradiction with high probability. Attack B is then likely to succeed.

In order to verify our observations for a small number of rounds, we apply Attack B to PRESENT-80 with for $N = 4$, $r = 3$. The block size and the key size for PRESENT-80 are 64 and 80, respectively. We have tested 10 wrong pairs satisfying the filter conditions in Group A and Group B, but not satisfying the conditions in Group C. We found that among 10 wrong pairs, only one wrong pair was filtered out within 1500 seconds. The reason is that the key size is larger than the block size.

As N and r increase, we ran several tests and list the results in Table 5. We identify different differential characteristics for the PRESENT-80 block cipher. For any value of r we tested, the characteristics have two active S-boxes from round 1 to round r. There will be two active S-boxes in round $(r + 1)$ and 6, 7 or 8 active S-boxes in round $(r + 2)$. Round $r + 3$ has at least 12 active S-boxes and round $(r + 4)$ has 16 active S-boxes. We use MiniSat2 to filter out the wrong pairs. For $N = r$, $N = r + 1$ or $N = r + 2$, no wrong pairs were filtered out. For $N = r + 3$, very few wrong pairs were filtered out. Although for $N = r + 4$, more wrong pairs were filtered out compared to $N = r + 3$, lots of wrong pairs still remain. The reason is that there are more active S-boxes in round $(r + 4)$ than in round $(r + 3)$. This result is consistent with Table 10.8 of [1], where $N = r + 4$ is used as well.

Further experiments are listed in Table 5. In Table 5, the plaintext pairs are all wrong pairs and we cannot filter them out within 1500 seconds. Even if wrong pairs can be filtered out after 1500 seconds, the time complexity of Attack B would become much higher than differential cryptanalysis. Due to space limitations, we only present the difference values for the pair in the last row of Table 5 and the characteristics for the right pair in Table 6. For the pair in Table 6, the output difference of the r-th ($r = 14$) round is same as that of the characteristics, but the difference values from round 2 to round 10 are different from that of the characteristic. Therefore, this pair is a wrong pair. We also confirmed experimentally that Attack B cannot filter out wrong pairs that do not satisfy the output difference for the first round.

Observation 2 can be derived from the following statements:

1. SAT solvers use a tree-structured search algorithm, where branching is performed by heuristic guesses based on non-algebraic criteria. In order to reduce the search time, we must minimize both the average search depth and the dependencies of the unknown variables. In this way, those equations should be identified that tend to result in an inconsistency sooner.

2. In the system of equations in Attack B, the equations that lead to inconsistencies the soonest, are the equations related to the difference values, the round functions in the outer rounds such as the previous few rounds and the later few rounds. In contrast, the equations related to the difference values and the round functions in the inner rounds do not easily lead to inconsistencies. Therefore, the equations in the inner rounds can be removed in order to reduce the solving time.

3. Since the equations for the difference value in the outer rounds are very important for the solving process, we must obtain enough such equations to ensure there are enough restrictions for the dependent unknown subkey bits. If there are fewer active S-boxes in the outer rounds, there are not enough restrictions on the involved unknown subkey bits to obtain the right solution or filter out the wrong solutions. In other words, if there are more active S-boxes in the outer rounds, the solving process or the filtering process will be more efficient.

It is noted that if there are more active S-boxes in the outer rounds, the filtering process will be efficient, but it is not favorable to filter out the wrong ciphertext pairs directly according to the difference value of the ciphertexts. This will further increase the time complexity.

To overcome these problems, we propose the following two methods for the differential-algebraic attack. The first method is to fix certain key bits to ensure with a high probability that the right key can be recovered from the right pair. The second method has the same goal, but adds some extra equations. We will describe these two attacks in Sect. 4.

4 New Differential-Algebraic Attacks

In Sect. 3, we showed that neither Attack C nor Attack B can improve the differential cryptanalysis of the PRESENT block cipher. We also explained why Attack C does not provide an improvement for most block ciphers. The reason is that the attacks cannot filter out the wrong pairs satisfying the ciphertext difference values to identify the right pair. We present two methods that can find the right solution in acceptable time t, based on the system of equations constructed in Attack B. For the right pair, we can solve the right key within time t. If a pair cannot be filtered within time t, we discard it and consider another pair.

Attack 1 Based on Fixing Certain Key Bits.

According to the key schedule algorithm and the outer rounds of the characteristic, fix the key bits related to the active S-boxes in the top rounds or the bottom rounds. In this way, inconsistencies can be found sooner. As we showed in Sect. 3.2, Attack B cannot be used to filter out most wrong pairs. Therefore, our attack fixes key bits in all tested pairs. The idea of fixing key bits was already proposed in [3]. The difference with Attack 1 is that we recover the entire key, and not only subkey bits from the last rounds.

Attack 2 Based on Multiple Pairs.

Because the equations for the difference values in the outer rounds lead to inconsistencies sooner, appending more such equations will be helpful to find the

inconsistency. Using multiple plaintext-ciphertext pairs to construct more equations of outer rounds will make the solving process or the filtering process more efficient. For example, if two plaintext-ciphertext pairs are used to perform the attack, the number of such equations will double. This means that if we use two right pairs to solve the system of equations, the right key can be found. However, if there is at least one wrong pair involved in the two pairs, the key cannot be found. In addition, if we use three plaintext-ciphertext pairs, the efficiency can be improved further. However, as the number of pairs increase, the number of combinations of pairs grows exponentially and the time complexity increases. So the number of pairs to construct the system of equations should not be too high.

Our experiments show that some wrong pairs can be filtered out quickly, but others cannot. However, if most of the wrong pairs cannot be filtered out, the attack becomes infeasible. So we attack the PRESENT block cipher with the above approaches and try to solve the right key with the right pairs.

4.1 Attack 1 for the PRESENT Block Cipher

We now apply Attack 1 to the PRESENT block cipher. The results are listed in Table 7. If we use $r = 13$ to attack $N = 15$ rounds of PRESENT-80, the probability of the characteristic is 2^{-58} (using the last 13 rounds of the 14-round characteristic of [24]). The filtering probability according to the difference value for the ciphertext pair is $2^{-50.49}$ (as calculated at the end of Sect. 3.1). The CPU clock frequency is 2.4 GHz. From Table 7, we find that it takes at most 523.16 s to find an inconsistency. The table also shows that we should guess at least 34 key bits, so the time complexity will be $2^{34} \cdot 2^{58-50.49} \cdot 2.4 \cdot 10^9 \cdot 523.16 = 2^{34} \cdot 2^{7.51} \cdot 2^{31.16} \cdot 2^{9.03} = 2^{81.70}$ CPU cycles. We assume that a single encryption costs at least 16 CPU cycles per round.[5] Therefore, the time complexity for our attack ($2^{73.79}$ equivalent encryptions) is better than exhaustive search (2^{80}).[6] The data complexity is 2^{59} chosen plaintexts. For the 2R-differential attack, the data complexity must be higher than 2^{59} chosen plaintexts, because then one right plaintext-ciphertext pair is not sufficient to recover the key with a high success probability. However, the time complexity of the 15-round 2R-differential attack must be lower than 2^{62} memory accesses (the time complexity given for the 16-round differential attack in [24]). Depending on the processor, one memory access requires about 2 to 10 CPU cycles. This means the complexity of the differential-algebraic attack for PRESENT-80 is much higher than that of the differential attack, but the data complexity is lower. Depending on how many chosen plaintext-ciphertext pairs the attacker can obtain, the algebraic-differential attack might however be the only feasible attack.

For PRESENT-128, we could not identify the right pairs for $r > 12$ using the method from [1]. If we use the 12-round differential characteristic with the

[5] The bitsliced implementation of PRESENT by Albrecht achieves 16.5 cycles per round [1].

[6] We used 20 trials to obtain time t. Although more trials may result in a longer time t, we expect that our attack will still be much faster than exhaustive search.

probability 2^{-54} to attack 14-round PRESENT-128, the time complexity will be about $2^{78+54-50.49+31.16+7.97} = 2^{120.64}$ CPU cycles, or about $2^{112.83}$ equivalent encryptions. The data complexity is 2^{55} chosen plaintexts.

4.2 Attack 2 for the PRESENT Block Cipher

We respectively use two pairs and three pairs to attack PRESENT. The test results are listed in Tables 8 and 9. For the right pairs, the right key can be solved within t seconds. We ran 10 trails for different values of r and N, and one example of right pairs $\{(P'_0, P''_0), (P'_1, P''_1)\}$ or $\{(P'_0, P''_0), (P'_1, P''_1), (P'_2, P''_2)\}$ and list the corresponding right key K. As in Attack 1, we can solve the right key from the right pairs, but the wrong pairs cannot always be filtered out. So we perform the test with the right pairs to recover the right key. We obtained the following results:

1. For $N = r + 3$ or $N = r + 4$ rounds of PRESENT-80 with the r-round differential characteristic, the right key can be solved with the two right pairs. Some test results are listed in Table 8. However, because we use two right pairs, this means that if m pairs of ciphertexts remain after filtering according to the ciphertext difference, we must consider $\binom{m}{2}$ combinations of two pairs. However, the solving time for $\binom{m}{2}$ combinations of two pairs becomes unacceptable. If we attack 16-round PRESENT-80 with a 13-round differential characteristic with the probability 2^{-58}, we choose 2^{59} pairs of plaintexts and the filtering probability with the ciphertext difference is about $2^{-25.711}$, so the number of the remaining ciphertext pairs is about $2^{33.289}$ which will be combined to produce $2^{65.578}$ combinations of two pairs. The time complexity will be $2^{65.578} \cdot 2^{31.16} \cdot t > 2^{88}$. We have not identified the right pairs for $r = 13$, so we cannot test the time for t and it should be more than 100 seconds according to the test time for $r < 13$. Therefore, Attack 2 is slower than exhaustive search.

2. For $N = r + 2$ rounds of PRESENT-80, only few combinations of two right pairs can be used to solve the right key, so the success rate is too low.

3. For $N = r + 4$ rounds of PRESENT-128 with the r-round differential, only few combinations of two right pairs can be used to recover the right key and the success rate is also very low.

4. For $N = r+3$ rounds of PRESENT-80 and $N = r+4$ rounds of PRESENT-128 with the r-round differential, the right key can be solved with the three right pairs. The test results are listed in Table 9. However, because we use three pairs, this means that if m pairs of ciphertexts remain, there are $\binom{m}{3}$ combinations of three pairs. However, the solving time for $\binom{m}{3}$ combinations of three pairs becomes unacceptable.

From the above results, Attack 2 (using two pairs or three pairs for PRESENT) has no advantage over Attack 1 (fixing certain key bits). Maybe these attacks have some advantage for other ciphers. For example, if there would be more active S-boxes involved in the outer rounds in PRESENT, maybe we could obtain the right key using two right pairs with a high success probability.

5 Conclusion

The cryptanalytic method combining differential cryptanalysis and algebraic cryptanalysis has been a focus topic in the field of the cryptanalysis of symmetric ciphers. At FSE 2009, Albrecht *et al.* propose new differential-algebraic attacks, which they claim improves the results of the differential cryptanalysis. In this paper, we revisited Albrecht's cryptanalytic method and identified that the time complexity to identify the right pairs is not correct. Firstly, we showed that Attack C cannot be used to filter out the wrong pairs satisfying the difference value of the ciphertexts for most block ciphers to improve the differential cryptanalysis. We identified some important properties for Attack B and showed that Attack B does not provide an advantage over differential cryptanalysis for PRESENT. Faugère *et al.* presented a similar attack for DES, however, they could only attack 8-round DES with a 5-round differential characteristic. Their attack for DES is accordant with our Observation 1 in Sect. 3.2 because the key size for DES is smaller than the block size.

In this paper, we introduce two new methods to perform a differential-algebraic attack. The first method is to fix certain key bits to solve the system of equations and the second method is to use multiple pairs to construct the system of equations. This method is more efficient for the PRESENT block cipher and its data complexity is better than that of the differential attack, but the time complexity is worse. Although we did not significantly improve the results of the differential cryptanalysis for PRESENT, our work indicates which equations are important in the differential-algebraic attack. For the differential-algebraic attack, we obtain the following three conclusions:

1. Compared with the differential cryptanalysis, the differential-algebraic attack can reduce the data complexity, but the time complexity increases. Compared with the algebraic cryptanalysis, the differential-algebraic attack can attack more rounds because the relations resulting from the differential characteristic are very important for the solving process.

2. In order to make the solving process in the differential-algebraic attack more efficient, more active S-boxes should be involved in the outer rounds. However, more active S-boxes will reduce the filtering probability with the ciphertext difference and it will increase the time complexity. The lower bound for the number of the active S-boxes should be used to ensure the system of

equations can be solved reliably. The detailed analysis of this case can be seen as future work.

3. If the methods to solve systems of equations can be improved, and if the computational power available increases, we expect that differential-algebraic attacks will gain in importance.

Acknowledgments. The authors would like thank the anonymous reviewers for their detailed comments and suggestions.

References

[1] M. Albrecht. *Algorithmic Algebraic Techniques and their Application to Block Cipher Cryptanalysis*. PhD thesis, Royal Holloway, University of London, 2010.

[2] M. Albrecht. Tools for the algebraic cryptanalysis of cryptographic primitives, 2010. `http://www.ecrypt.eu.org/tools/tools-for-algebraic-cryptanalysis`.

[3] M. Albrecht and C. Cid. Algebraic Techniques in Differential Cryptanalysis. In O. Dunkelman, editor, *FSE*, volume 5665 of *Lecture Notes in Computer Science*, pages 193–208. Springer, 2009.

[4] M. Albrecht, C. Cid, T. Dullien, J.-C. Faugère, and L. Perret. Algebraic Precomputations in Differential and Integral Cryptanalysis. In X. Lai, M. Yung, and D. Lin, editors, *Inscrypt*, volume 6584 of *Lecture Notes in Computer Science*, pages 387–403. Springer, 2010.

[5] G. V. Bard. *Algebraic Cryptanalysis*, volume XXXIV of *Security and Cryptology*. Springer, 2009.

[6] E. Biham and A. Shamir. Differential Cryptanalysis of DES-like Cryptosystems. *J. Cryptology*, 4(1):3–72, 1991.

[7] E. Biham and A. Shamir. Differential Cryptanalysis of the Full 16-Round DES. In E. F. Brickell, editor, *CRYPTO*, volume 740 of *Lecture Notes in Computer Science*, pages 487–496. Springer, 1992.

[8] C. Blondeau and B. Gérard. Multiple Differential Cryptanalysis: Theory and Practice. In A. Joux, editor, *FSE*, volume 6733 of *Lecture Notes in Computer Science*, pages 35–54. Springer, 2011.

[9] A. Bogdanov, L. R. Knudsen, G. Leander, C. Paar, A. Poschmann, M. J. B. Robshaw, Y. Seurin, and C. Vikkelsoe. PRESENT: An Ultra-Lightweight Block Cipher. In P. Paillier and I. Verbauwhede, editors, *CHES*, volume 4727 of *Lecture Notes in Computer Science*, pages 450–466. Springer, 2007.

[10] M. Brickenstein and A. Dreyer. PolyBoRi: A framework for Gröbner-basis computations with Boolean polynomials. *J. Symb. Comput.*, 44(9):1326–1345, 2009.

[11] J. Y. Cho. Linear Cryptanalysis of Reduced-Round PRESENT. In J. Pieprzyk, editor, *CT-RSA*, volume 5985 of *Lecture Notes in Computer Science*, pages 302–317. Springer, 2010.

[12] N. Courtois. Fast Algebraic Attacks on Stream Ciphers with Linear Feedback. In D. Boneh, editor, *CRYPTO*, volume 2729 of *Lecture Notes in Computer Science*, pages 176–194. Springer, 2003.

[13] N. Courtois and G. V. Bard. Algebraic Cryptanalysis of the Data Encryption Standard. In S. D. Galbraith, editor, *IMA Int. Conf.*, volume 4887 of *Lecture Notes in Computer Science*, pages 152–169. Springer, 2007.

[14] N. Courtois and B. Debraize. Specific S-Box Criteria in Algebraic Attacks on Block Ciphers with Several Known Plaintexts. In S. Lucks, A.-R. Sadeghi, and C. Wolf, editors, *WEWoRC*, volume 4945 of *Lecture Notes in Computer Science*, pages 100–113. Springer, 2007.

[15] N. Courtois and W. Meier. Algebraic Attacks on Stream Ciphers with Linear Feedback. In E. Biham, editor, *EUROCRYPT*, volume 2656 of *Lecture Notes in Computer Science*, pages 345–359. Springer, 2003.

[16] N. Courtois and J. Pieprzyk. Cryptanalysis of Block Ciphers with Overdefined Systems of Equations. In Y. Zheng, editor, *ASIACRYPT*, volume 2501 of *Lecture Notes in Computer Science*, pages 267–287. Springer, 2002.

[17] N. Eén and N. Sörensson. An Extensible SAT-solver. In E. Giunchiglia and A. Tacchella, editors, *SAT*, volume 2919 of *Lecture Notes in Computer Science*, pages 502–518. Springer, 2003.

[18] J.-C. Faugère and A. Joux. Algebraic Cryptanalysis of Hidden Field Equation (HFE) Cryptosystems Using Gröbner Bases. In D. Boneh, editor, *CRYPTO*, volume 2729 of *Lecture Notes in Computer Science*, pages 44–60. Springer, 2003.

[19] J.-C. Faugère, L. Perret, and P.-J. Spaenlehauer. Algebraic-Differential Cryptanalysis of DES. In *Western European Workshop on Research in Cryptology - WEWoRC 2009*, pages 1–5, 2009.

[20] Z. Gong, P. H. Hartel, S. Nikova, and B. Zhu. Towards Secure and Practical MACs for Body Sensor Networks. In B. K. Roy and N. Sendrier, editors, *INDOCRYPT*, volume 5922 of *Lecture Notes in Computer Science*, pages 182–198. Springer, 2009.

[21] J. Nakahara, P. Sepehrdad, B. Zhang, and M. Wang. Linear (Hull) and Algebraic Cryptanalysis of the Block Cipher PRESENT. In J. A. Garay, A. Miyaji, and A. Otsuka, editors, *CANS*, volume 5888 of *Lecture Notes in Computer Science*, pages 58–75. Springer, 2009.

[22] O. Özen, K. Varıcı, C. Tezcan, and Çelebi Kocair. Lightweight Block Ciphers Revisited: Cryptanalysis of Reduced Round PRESENT and HIGHT. In C. Boyd and J. M. G. Nieto, editors, *ACISP*, volume 5594 of *Lecture Notes in Computer Science*, pages 90–107. Springer, 2009.

[23] A. A. Selçuk. On Probability of Success in Linear and Differential Cryptanalysis. *J. Cryptology*, 21(1):131–147, 2008.

[24] M. Wang. Differential Cryptanalysis of Reduced-Round PRESENT. In S. Vaudenay, editor, *AFRICACRYPT*, volume 5023 of *Lecture Notes in Computer Science*, pages 40–49. Springer, 2008.

[25] M. Wang, X. Wang, and L. C. Hui. Differential-algebraic cryptanalysis of reduced-round of Serpent-256. *SCIENCE CHINA Information Sciences*, 53(3):546–556, 2010.

Table 1 – Attack C's Filtering Test for Wrong Pairs with MiniSat2

N	r	P'	P''	K	$t(s)$
8-10	7	8b29917c174f21b7	8c29917c174f26b7	2b8bc6ad5d4b869101c2	12.20-12.77
9-11	8	d549bf122a09$edfa$	d249bf122a09$eafa$	5d05c98dce5da5894fc5	12.26-12.92
10-12	9	f5fc5a0d3979d9d3	f2fc5a0d3979ded3	f53e4$ecaf$9ce361ee6d7	12.11-13.03
11-13	10	50d752ee7f6017d7	57d752ee7f6010d7	afc238c99ce160d8254b	12.22-12.73
12-14	11	155$fdec$5b70e8b3a	125$fdec$5b70e8c3a	b544c98fce9474d53925	12.33-12.92
13-15	12	504ad07e763a8289	574ad07e763a8589	a7ece17b6ab73269d7e9	12.01-12.71

N: the round number we attack; r: the round number of the differential; K: right key; (P', P''): one example of wrong pairs; t: the wrong solution obtained within t seconds.

Table 2 – Difference Values for Wrong Pair and Right Pair in Attack C

R		Δ_{wrong}	Δ_{right}	R		Δ_{wrong}	Δ_{right}
I		$x_2 = 7, x_{14} = 7$	$x_2 = 1, x_{14} = 1$				
R1	S	$x_2 = 1, x_{14} = 1$	$x_2 = 1, x_{14} = 1$	R8	S	$x_0 = 9, x_2 = 9$	$x_8 = 9, x_{10} = 9$
R1	P	$x_0 = 4, x_3 = 4$	$x_0 = 4, x_3 = 4$	R8	P	$x_0 = 5, x_{12} = 5$	$x_2 = 5, x_{14} = 5$
R2	S	$x_0 = 5, x_3 = 5$	$x_0 = 5, x_3 = 5$	R9	S	$x_0 = 1, x_{12} = 1$	$x_2 = 1, x_{14} = 1$
R2	P	$x_0 = 9, x_8 = 9$	$x_0 = 9, x_8 = 9$	R9	P	$x_0 = 1, x_3 = 1$	$x_0 = 4, x_3 = 4$
R3	S	$x_0 = 4, x_8 = 4$	$x_0 = 4, x_8 = 4$	R10	S	$x_0 = 3, x_3 = 3$	$x_0 = 5, x_3 = 5$
R3	P	$x_8 = 1, x_{10} = 1$	$x_8 = 1, x_{10} = 1$	R10	P	$x_0 = 9, x_4 = 9$	$x_0 = 9, x_8 = 9$
R4	S	$x_8 = 3, x_{10} = 3$	$x_8 = 9, x_{11} = 9$	R11	S	$x_0 = 4, x_4 = 4$	$x_0 = 4, x_8 = 4$
R4	P	$x_2 = 5, x_6 = 5$	$x_2 = 5, x_{14} = 5$	R11	P	$x_8 = 1, x_9 = 1$	$x_8 = 1, x_{10} = 1$
R5	S	$x_2 = 1, x_6 = 1$	$x_2 = 1, x_{14} = 1$	R12	S	$x_8 = 9, x_9 = 9$	$x_8 = 9, x_{10} = 9$
R5	P	$x_0 = 4, x_1 = 4$	$x_0 = 4, x_3 = 4$	R12	P	$\mathbf{x_2 = 3, x_{14} = 3}$	$\mathbf{x_2 = 5, x_{14} = 5}$
R6	S	$x_0 = 5, x_1 = 5$	$x_0 = 5, x_3 = 5$	R13	S	$\mathbf{x_2 = 1, x_{14} = 1}$	$\mathbf{x_2 = 1, x_{14} = 1}$
R6	P	$x_0 = 3, x_8 = 3$	$x_0 = 9, x_8 = 9$	R13	P	$\mathbf{x_0 = 4, x_3 = 4}$	$\mathbf{x_0 = 4, x_3 = 4}$
R7	S	$x_0 = 1, x_8 = 1$	$x_0 = 4, x_8 = 4$				
R7	P	$x_0 = 1, x_2 = 1$	$x_8 = 1, x_{10} = 1$				

Rj: output difference after round j (S: after S-box layer, P: after permutation layer); Δ_{wrong}: differential value for wrong pair; Δ_{right}: differential value for right pair.

Table 3 – Filter Time for Wrong Pairs Not Satisfying Equations in any Group

N	r	♯trails	PolyBoRi	MiniSat2	N	r	♯trails	PolyBoRi	MiniSat2
9	8	20	3.51-3.85	4.06-4.64	13	12	20	4.99-5.34	4.96-5.25
10	8	20	4.89-5.23	7.57-8.44	14	12	20	6.67-6.83	8.86-9.26
11	8	20	7.89-8.41	11.29-12.34	15	12	20	9.69-10.20	12.80-13.15
10	9	20	3.92-4.27	4.55-4.79	14	13	20	5.66-5.78	5.07-5.37
11	9	20	5.32-5.66	8.40-8.66	15	13	20	7.02-7.50	9.08-9.38
12	9	20	6.24-6.59	12.19-12.45	16	13	20	7.99-8.51	12.91-13.58
11	10	20	4.28-4.67	4.73-4.99	15	14	20	6.06-6.18	5.24-5.52
12	10	20	4.75-5.09	8.35-8.59	16	14	20	6.50-6.95	9.04-9.47
13	10	20	6.93-7.05	12.32-12.59	17	14	20	8.48-8.88	13.17-13.77
12	11	20	4.66-5.02	4.87-5.12					
13	11	20	6.09-6.42	8.69-8.97					
14	11	20	7.41-10.17	12.42-12.75					

♯trails: the number of wrong pairs we test;
PolyBoRi: the filtering time in seconds with PolyBori;
MiniSat2: the filtering time in seconds with Minisat2.

Table 4 – Filter Time for Wrong Pairs Only Satisfying Equations in Group A

N	r	♯trails	PolyBoRi	MiniSat2
10	8	20	5.07-5.55	8.09-8.53
11	9	20	6.33-6.68	7.34-7.81
12	10	20	6.02-6.45	7.53-8.12

Table 5 – Attack B's Filtering Test for Wrong Pairs Satisfying Ciphertext Difference Values with MiniSat2 (Timeout $t = 1500$ s)

N	r	P'	P''	K
5-7	4	67279b1$efdb$93674	60279b1$efdb$93174	9ad864e12a6ecc872280
6-8	5	cdc43299824183d4	cac43299824184d4	70be32f5dd35396$cdbfd$
7-9	6	bc887a5de0597dd6	bb887a5de0597ad6	716d9698292707b0b6da
8-10	7	c53f11ab7329e7cf	c23f11ab7329e0cf	78bf3977$acaffded$898a
9-11	8	6d736a36a28d4f93	6a736a36a28d4893	5e7f5234d2063c5dd11d
10-12	9	94bd4ffd6585072e	93bd4ffd6585002e	1e00538c107f7abc4a73
11,12,13	10	f02f740d8d4b6d37	f72f740d8d4b6a37	df76f9$fdaf$4ead07d9a2
12,13,14	11	85f4ab19cf1dd9ac	82f4ab19cf1$ddeac$	5d0de0769a874e36d362
13,14,15	12	ca8b8755e65217af	cd8b8755e65210af	2d0d71c7a40d3084ac3a
15,16,17	14	934c64486fa9ed41	944c64486fa9ea41	8b1c1828ec601df09214

Table 6 – Difference Values for Wrong Pair and Right Pair in Attack B

R		Δ_{wrong}	Δ_{right}	R		Δ_{wrong}	Δ_{right}
I		$x_2 = 7, x_{14} = 7$	$x_2 = 7, x_{14} = 7$				
R1	S	$x_2 = 1, x_{14} = 1$	$x_2 = 1, x_{14} = 1$	R8	S	$x_8 = 5, x_{10} = 5$	$x_8 = 9, x_{10} = 9$
R1	P	$x_0 = 4, x_3 = 4$	$x_0 = 4, x_3 = 4$	R8	P	$x_2 = 5, x_{10} = 5$	$x_2 = 5, x_{14} = 5$
R2	S	$x_0 = 9, x_3 = 9$	$x_0 = 5, x_3 = 5$	R9	S	$x_2 = 1, x_{10} = 1$	$x_2 = 1, x_{14} = 1$
R2	P	$x_0 = 9, x_{12} = 9$	$x_0 = 9, x_8 = 9$	R9	P	$x_0 = 4, x_2 = 4$	$x_0 = 4, x_3 = 4$
R3	S	$x_0 = 4, x_{12} = 4$	$x_0 = 4, x_8 = 4$	R10	S	$x_0 = 5, x_2 = 5$	$x_0 = 5, x_3 = 5$
R3	P	$x_8 = 1, x_{11} = 1$	$x_8 = 1, x_{10} = 1$	R10	P	$x_0 = 5, x_8 = 5$	$x_0 = 9, x_8 = 9$
R4	S	$x_8 = 9, x_{11} = 9$	$x_8 = 9, x_{10} = 9$	R11	S	$x_0 = 4, x_8 = 4$	$x_0 = 4, x_8 = 4$
R4	P	$x_2 = 9, x_{14} = 9$	$x_2 = 5, x_{14} = 5$	R11	P	$x_8 = 1, x_{10} = 1$	$x_8 = 1, x_{10} = 1$
R5	S	$x_2 = 4, x_{14} = 4$	$x_2 = 1, x_{14} = 1$	R12	S	$x_8 = 9, x_{10} = 9$	$x_8 = 9, x_{10} = 9$
R5	P	$x_8 = 4, x_{11} = 4$	$x_0 = 4, x_3 = 4$	R12	P	$x_2 = 5, x_{14} = 5$	$x_2 = 5, x_{14} = 5$
R6	S	$x_8 = 5, x_{11} = 5$	$x_0 = 5, x_3 = 5$	R13	S	$x_2 = 1, x_{14} = 1$	$x_2 = 1, x_{14} = 1$
R6	P	$x_2 = 9, x_{10} = 9$	$x_0 = 9, x_8 = 9$	R13	P	$x_0 = 4, x_3 = 4$	$x_0 = 4, x_3 = 4$
R7	S	$x_2 = 4, x_{10} = 4$	$x_0 = 4, x_8 = 4$	R14	S	$x_2 = 4, x_{10} = 4$	$x_0 = 4, x_8 = 4$
R7	P	$x_8 = 4, x_{10} = 4$	$x_8 = 1, x_{10} = 1$	R14	P	$x_0 = 9, x_8 = 9$	$x_0 = 9, x_8 = 9$

Rj: output difference after round j (S: after S-box layer, P: after permutation layer); Δ_{wrong}: differential value for wrong pair; Δ_{right}: differential value for right pair.

Table 7 – Time to Solve Right Key under Some Fixed Key Bits with MiniSat2

K_s	N	r	♯trails	N_k	$t(s)$	K_s	N	r	♯trails	N_k	$t(s)$
80	10	10	20	32	45.18-285.20	80	14-17	14	20	36	63.47-120.08
80	11	10	20	32	64.45-564.87	128	10	10	20	79	43.75-288.63
80	12	10	20	32	61.88-591.56	128	11	10	20	78	63.38-821.45
80	13	10	20	32	53.49-497.96	128	12	10	20	75	79.83-966.38
80	11	11	20	33	60.19-151.28	128	13	10	20	72	89.15-751.30
80	12	11	20	33	53.01-316.94	128	11	11	20	79	98.35-662.19
80	13	11	20	33	56.64-528.03	128	12	11	20	79	58.73-483.92
80	14	11	20	33	56.25-104.26	128	13	11	20	79	69.41-805.18
80	12	12	20	34	97.19-487.77	128	14	11	20	71	78.20-891.08
80	13	12	20	34	69.24-680.41	128	12	12	20	82	57.35-115.11
80	14	12	20	34	61.09-110.02	128	13	12	20	82	118.08-668.53
80	15	12	20	34	59.25-77.82	128	14	12	20	78	61.84-251.14
80	13-16	13	20	34	85.54-523.16	128	15	12	20	66	64.86-309.90

N_k: the number of fixed key bits.

Table 8 – Time to Solve Right Key using Two Right Pairs with MiniSat2

K_s	N	r	P'_0, P'_1	P''_0, P''_1	K	$t(s)$
80	12	9	39121b2$bffad$3bbc,	3e121b2$bffad$3cbc,	4634342e33 ‖	132.88-377.13
			91f1a75a4f4d33e0	96f1a75a4f4d34e0	0d53e8cd71	
80	13	10	67bb6$eecd$081767c,	60bb6$eecd$081717c,	6$fcaf$3033d ‖	122.00-849.89
			6f62c9bd561f718e	6862c9bd561f768e	39296c0f66	
80	14	11	c2b3135aa3b8f3b4,	c5b3135aa3b8f4b4,	22c587b7b2 ‖	129.01-213.98
			8a43480c3122ab14	8d43480c3122ac14	607$cddab$90	
80	15	12	c2b3135aa3b8f3b4,	125fcb08$afed$6df3,	155fcb08af ‖	133.64-141.75
			85c6576306a6a545	82c6576306a6a245	ed6af317f1	
80	13	9	0c03406225bf97cd,	0b03406225bf90cd,	cca9$deeb$2c ‖	115.61-133.35
			0bbd25aea7c5b0c9	0cbd25aea7c5b7c9	0d98071ca6	
80	14	10	9434381cb8083429,	9334381cb8083329,	ab7b47fdf8 ‖	124.22-132.99
			0b40a64e215244c6	0c40a64e215243c6	93fb87c9cd	
80	15	11	8814d6bea07fd660,	8f14d6bea07fd160,	a7d16cda8d ‖	130.48-144.89
			f02e367f419a412e	f72e367f419a462e	b76ec42756	
80	16	12	$cbaef$2f923614742,	$ccaef$2f923614042,	6b9b4087a6 ‖	189.26-280.49
			b37ee1f334c4207b	b47ee1f334c4277b	254f2$bbef$2	

Table 9 – Time to Solve Right Key using Three Right Pairs with MiniSat2

K_s	N	r	P'_0,P'_1,P'_2	P''_0,P''_1,P''_2	K	$t(s)$
80	11	9	$d9591ff50fc1df6d$, $f9866c0009f3bf44$, $0e768137f568779d$	$de591ff50fc1d86d$, $fe866c0009f3b844$, $09768137f568709d$	$66efab8af3 \parallel 74afe67553$	177.77-1402.2
80	12	10	$3a659aa3dc72107c$, $62129df1a637b88f$, $c566bb319010f0df$	$3d659aa3dc72177c$, $65129df1a637bf8f$, $c266bb319010f7df$	$2dc9fceff3 \parallel 174f9919c4$	240.70-578.68
80	13	11	$383663a9bc01cec5$, $88042f67e3b59e95$, $c842b19a415d9105$	$3f3663a9bc01c9c5$, $8f042f67e3b59995$, $cf42b19a415d9605$	$a0f5a7209b \parallel b95180a21c$	247.53-t ($t > 2500$)
80	14	12	$2ddbc9427defb9ee$, $2aa2624e2cb1dede$, $4d19fefd126a29ee$	$2adbc9427defbeee$, $2da2624e2cb1d9de$, $4a19fefd126a2eee$	$3200679dd6 \parallel 3d29ae18bc$	293.21-408.40
80	12	9	$3d84126858c7435e$, $32a6811bd0c6a32e$, $cd66cbdb18c23c55$	$3a84126858c7445e$, $35a6811bd0c6a42e$, $ca66cbdb18c23b55$	$5da70ed0b5 \parallel 13fb14435c$	216.35-239.90
80	13	10	$e519cccfa40ce691$, $e5aa80afcfc216a3$, $8a179faf87127908$	$e219cccfa40ce191$, $e2aa80afcfc211a3$, $8d179faf87127e08$	$72ada6021d \parallel d2667ab4e5$	238.47-258.13
80	14	11	$f5a33b54749b6624$, $b2f64b6c661d6101$, $2d106b5e6d2b4e24$	$f2a33b54749b6124$, $b5f64b6c661d6601$, $2a106b5e6d2b4924$	$8ab6e28d86 \parallel 9ef6858a87$	292.15-319.56
80	15	12	$e6005b48d2abd194$, $41909dfa1ac196d9$, $0e43381eb485d900$	$e1005b48d2abd694$, $46909dfa1ac191d9$, $0943381eb485de00$	$393d660706 \parallel 1dbe32c806$	271.31-340.26
128	13	9	$9d6902f268514522$, $95d585a882e6e250$, $2da0d2114f1805c2$	$9a6902f268514222$, $92d585a882e6e550$, $2aa0d2114f1802c2$	$0578224d0c9eba10 \parallel bb0fd3b56d8b4834$	235.64-265.20
128	14	10	$972331fa763f86bd$, $50d342a2a6dce17a$, $efdfd44485f1ee81$	$902331fa763f81bd$, $57d342a2a6dce67a$, $e8dfd44485f1e981$	$d8ca446899016e69 \parallel 17641f71e11d09f5$	235.16-291.02
128	15	11	$76971713b1f0d438$, $aed2ee07ad11dc6d$, $e609bfed79d4143b$	$71971713b1f0d338$, $a9d2ee07ad11db6d$, $e109bfed79d4133b$	$9e3328405c865b25 \parallel 2201229c273fd1dd$	285.00-303.82
128	16	12	$eb449a907d31f33e$, $84363465aaddb304$, $e3a2e5866f5814a9$	$ec449a907d31f43e$, $83363465aaddb404$, $e4a2e5866f5813a9$	$73fdf364db99c472 \parallel bb7a8e563b20a1f2$	316.21-414.30

Publication Chapter

Differential and Linear Cryptanalysis using Mixed-Integer Linear Programming

Publication Data

Nicky Mouha, Qingju Wang, Dawu Gu, and Bart Preneel. Differential and Linear Cryptanalysis using Mixed-Integer Linear Programming. In Moti Yung and Chuan-Kun Wu, editors, *Inscrypt*, Lecture Notes in Computer Science. Springer, 2011.

Contributions

- Main author.

Differential and Linear Cryptanalysis using Mixed-Integer Linear Programming*

Nicky Mouha[1,3,†], Qingju Wang[1,2,3], Dawu Gu[2], and Bart Preneel[1,3]

[1] Department of Electrical Engineering ESAT/SCD-COSIC,
Katholieke Universiteit Leuven. Kasteelpark Arenberg 10, B-3001 Heverlee, Belgium.
[2] Department of Computer Science and Engineering, Shanghai Jiao Tong University,
Shanghai, China.
[3] Interdisciplinary Institute for BroadBand Technology (IBBT), Belgium.
{Nicky.Mouha,Qingju.Wang}@esat.kuleuven.be

Abstract. Differential and linear cryptanalysis are two of the most powerful techniques to analyze symmetric-key primitives. For modern ciphers, resistance against these attacks is therefore a mandatory design criterion. In this paper, we propose a novel technique to prove security bounds against both differential and linear cryptanalysis. We use mixed-integer linear programming (MILP), a method that is frequently used in business and economics to solve optimization problems. Our technique significantly reduces the workload of designers and cryptanalysts, because it only involves writing out simple linear inequality constraints that are input into an MILP solver. As very little programming is required, both the time spent on cryptanalysis and the possibility of human errors are greatly reduced. Our method is used to analyze Enocoro-128v2, a stream cipher that consists of 96 rounds. We prove that 38 rounds are sufficient for security against differential cryptanalysis, and 61 rounds for security against linear cryptanalysis. We also illustrate our technique by calculating the number of active S-boxes for AES.

Keywords: Differential cryptanalysis, Linear Cryptanalysis, Mixed-Integer Linear Programming, MILP, Enocoro, AES, CPLEX

1 Introduction

Differential cryptanalysis [1] and linear cryptanalysis [18] have shown to be two of the most important techniques in the analysis of symmetric-key cryptographic

*This work was supported in part by the Research Council K.U.Leuven: GOA TENSE, the IAP Program P6/26 BCRYPT of the Belgian State (Belgian Science Policy), and in part by the European Commission through the ICT program under contract ICT-2007-216676 ECRYPT II, and is funded by the National Natural Science Foundation of China (No. 61073150).

†This author is funded by a research grant of the Institute for the Promotion of Innovation through Science and Technology in Flanders (IWT-Vlaanderen).

primitives. For block ciphers, differential cryptanalysis analyzes how input differences in the plaintext lead to output differences in the ciphertext. Linear cryptanalysis studies probabilistic linear relations between plaintext, ciphertext and key. If a cipher behaves differently from a random cipher for differential or linear cryptanalysis, this can be used to build a distinguisher or even a key-recovery attack.

For stream ciphers, differential cryptanalysis can be used in the context of a resynchronization attack [10]. In one possible setting, the same data is encrypted several times with the same key, but using a different initial value (IV). This is referred to as the standard (non-related-key) model, where the IV value is assumed to be under control of the attacker. An even stronger attack model is a related-key setting, where the same data is encrypted with different IVs *and* different keys. Not only the IV values, but also the differences between the keys are assumed to be under control of the attacker. Similar to differential cryptanalysis, linear cryptanalysis can also be used to attack stream ciphers in both standard and related-key models. In the case of stream ciphers, linear cryptanalysis amounts to a known-IV attack instead of a chosen-IV attack.

Resistance against linear and differential cryptanalysis is a standard design criterion for new ciphers. For the block cipher AES [12], provable security against linear and differential cryptanalysis follows from the wide trail design strategy [11]. In this work, we apply a similar strategy. After proving a lower bound on the number of active S-boxes for both differential and linear cryptanalysis, we use the maximum differential probability (MDP) of the S-boxes to derive an upper bound for the probability of the best characteristic. We assume (as is commonly done) that the probability of the differential can accurately be estimated by the probability of the best characteristic. Several works focus on calculating the minimum number of active S-boxes for both Substitution-Permutation Networks (SPNs) [11] and (Generalized) Feistel Structures (GFSs) [4, 5, 15, 23]. Unfortunately, it seems that a lot of time and effort in programming is required to apply those techniques. This may explain why many related constructions have not yet been thoroughly analyzed. In this paper, we introduce a novel technique using mixed-integer linear programming in order to overcome these problems.

Linear programming (LP) is the study of optimizing (minimizing or maximizing) a linear objective function $f(x_1, x_2, \ldots, x_n)$, subject to linear inequalities involving decision variables x_i, $1 \leq i \leq n$. For many such optimization problems, it is necessary to restrict certain decision variables to integer values, i.e. for some values of i, we require $x_i \in \mathbb{Z}$. Methods to formulate and solve such programs are called mixed-integer linear programming (MILP). If all decision variables x_i must be integer, the term (pure) integer linear programming (ILP) is used. MILP techniques have found many practical applications in the fields of economy and business, but their application in cryptography has so far been limited. For a good introductory level text on LP and (M)ILP, we refer to Schrage [22].

In [6], Borghoff *et al.* transformed the quadratic equations describing the

stream cipher Bivium into a MILP problem. The IBM ILOG CPLEX Optimizer[4] was then used to solve the resulting MILP problem, which corresponds to recovering the internal state of Bivium. In the case of Bivium A, solving this MILP problem takes less than 4.5 hours, which is faster than Raddum's approach (about a day) [21], but much slower than using MiniSAT (21 seconds) [8].

For the hash function SIMD, Bouillaguet *et al.* [7] used an ILP solver to find a differential characteristic based on local collisions. Using the SYMPHONY solver,[5] they could not find the optimal solution, but found lower bounds for both SIMD-256 and SIMD-512. The computation for SIMD-512 took one month on a dual quad-core computer.

In [4,5], Bogdanov calculated the minimum number of linearly and differentially active S-boxes of unbalanced Feistel networks with contracting MDS diffusion. He proved that some truncated difference weight distributions are impossible or equivalent to others. For the remaining truncated difference weight distributions, he constructed an ILP program which he then solved using the MAGMA[6] Computational Algebra System [3]. Compared to Bogdanov's technique, the fully automated method in this paper is much simpler to apply: Bogdanov's approach requires a significant amount of manual work, and the construction of not one but several ILP programs. We will show how this can be avoided by introducing extra dummy variables into the MILP program.

While this paper was under submission, Wu and Wang released a paper on ePrint [27] that also uses integer linear programming to count the number of active S-boxes for both linear and differential cryptanalysis. Just as in Bogdanov's approach, their algorithms require a large number of ILP programs to be solved, instead of only one as in the technique of this paper.

We apply our technique to the stream cipher Enocoro-128v2 [25, 26], in order to obtain bounds against differential and linear cryptanalysis. We consider both the standard and related-key model. All MILP programs are solved using CPLEX. There are 96 initialization rounds in Enocoro-128v2. We prove that 38 rounds are sufficient for security against differential cryptanalysis, and 61 rounds against linear cryptanalysis. These security bounds are obtained after 52.68 and 228.94 seconds respectively. We also calculate the minimum number of active S-boxes for up to 14 rounds of AES, which takes at most 0.40 seconds for each optimization program. Our experiments are performed on a 24-core Intel Xeon X5670 Processor, with 16 GB of RAM.

This paper is organized as follows. Sect. 2 explains how to find the minimum number of active S-boxes for a cryptographic primitive by solving an MILP program. A brief description of Enocoro-128v2 is given in Sect. 3. In Sect. 4 and Sect. 5, we construct an MILP program to prove that Enocoro-128v2 is secure against differential cryptanalysis and linear cryptanalysis respectively. We provide some ideas for future work in Sect. 6, and conclude the paper in Sect. 7. In

[4] `http://www.ibm.com/software/integration/optimization/cplex-optimizer/`
[5] `http://projects.coin-or.org/SYMPHONY`
[6] `http://magma.maths.usyd.edu.au/`

App. A, we calculate the minimum number of active S-boxes for AES using our technique, and provide the full source code of our program.

2 Constructing an MILP Program to Calculate the Minimum Number of Active S-boxes

We now explain a technique to easily prove the security of many ciphers against differential and linear cryptanalysis. Our method is based on counting the minimum number of active S-boxes. To illustrate our technique, we use Enocoro-128v2 and AES as test cases in this paper. The constraints we describe are not specific to these ciphers, but can easily be applied to any cipher constructed using S-box operations, linear permutation layers, three-forked branches and/or XOR operations.

2.1 Differential Cryptanalysis

We consider truncated differences, that is, every byte in our analysis can have either a zero or a non-zero difference. More formally, we define the following difference vector:

Definition 1. Consider a string Δ consisting of n bytes $\Delta = (\Delta_0, \Delta_1, \ldots, \Delta_{n-1})$. Then, the difference vector $x = (x_0, x_1, \ldots, x_{n-1})$ corresponding to Δ is defined as

$$x_i = \begin{cases} 0 & \text{if } \Delta_i = 0 \ , \\ 1 & \text{otherwise} \ . \end{cases}$$

Constraints Describing the XOR Operation.

Let the input difference vector for the XOR operation be $(x^{\oplus}_{in_1}, x^{\oplus}_{in_2})$ and the corresponding output difference vector be $x^{\oplus}_{out}$. The differential branch number is defined as the minimum number of input and output bytes that contain differences, excluding the case where there are no differences in inputs nor outputs. For XOR, the differential branch number is 2. In order to express this branch number in linear inequality constraints, we need to introduce a new binary dummy variable $d^{\oplus}$.[7] If and only if all of the three variables $x^{\oplus}_{in_1}$,$x^{\oplus}_{in_2}$ and $x^{\oplus}_{out}$ are zero, $d^{\oplus}$ is zero, otherwise it should be one. Therefore we obtain the following linear constraints (in binary variables) to describe the relation between the input and output difference

[7] Note that this extra variable was not added in [4, 5], which is why Bogdanov had to solve several ILP programs instead of only one.

vectors:

$$\begin{aligned} x_{in_1}^{\oplus} + x_{in_2}^{\oplus} + x_{out}^{\oplus} &\geq 2d^{\oplus} \ , \\ d^{\oplus} &\geq x_{in_1}^{\oplus} \ , \\ d^{\oplus} &\geq x_{in_2}^{\oplus} \ , \\ d^{\oplus} &\geq x_{out}^{\oplus} \ . \end{aligned}$$

Constraints Describing the Linear Transformation.

The constraints for a linear transformation L can be described as follows. Assume L transforms the input difference vector $(x_{in_1}^L, x_{in_2}^L, \cdots, x_{in_{\mathcal{M}}}^L)$ to the output difference vector $(x_{out_1}^L, x_{out_2}^L, \cdots, x_{out_{\mathcal{M}}}^L)$. Given the differential branch number $\mathcal{B}_D$, a binary dummy variable d^L is again needed to describe the relation between the input and output difference vectors. The variable d^L is equal to 0 if all variables $x_{in_1}^L, x_{in_2}^L, \cdots, x_{in_{\mathcal{M}}}^L, x_{out_1}^L, x_{out_2}^L, \cdots, x_{out_{\mathcal{M}}}^L$ are 0, and 1 otherwise. Therefore the linear transformation L can be constrained by the following linear inequalities:

$$\begin{aligned} x_{in_1}^L + x_{in_2}^L + \cdots + x_{in_{\mathcal{M}}}^L + x_{out_1}^L + x_{out_2}^L + \cdots + x_{out_{\mathcal{M}}}^L &\geq \mathcal{B}_D d^L \ , \\ d^L &\geq x_{in_1}^L \ , \\ d^L &\geq x_{in_2}^L \ , \\ &\cdots\cdots \\ d^L &\geq x_{in_{\mathcal{M}}}^L \ , \\ d^L &\geq x_{out_1}^L \ , \\ d^L &\geq x_{out_2}^L \ , \\ &\cdots\cdots \\ d^L &\geq x_{out_{\mathcal{M}}}^L \ . \end{aligned}$$

The Objective Function.

The objective function that has to be minimized, is the number of active S-boxes. This function is equal to the sum of all variables that correspond to the S-box inputs.

Additional Constraints.

An extra linear constraint is added to ensure that at least one S-box is active: this avoids the trivial solution where the minimum active S-boxes is zero. If all d-variables and all x-variables are restricted to be binary, the resulting program is a pure ILP (Integer Linear Programming) problem. If all d-variables are restricted to be binary, but only the x-variables corresponding to the input (plaintext), the linear inequality constraints ensure that the optimal solution for all other x-variables

will be binary as well. This is similar to Borghoff's suggestion in [6], and results in an MILP (Mixed-Integer Linear Programming) problem that may be solved faster.

2.2 Linear Cryptanalysis

For linear cryptanalysis, we define a linear mask vector as follows:

Definition 2. Given a set of linear masks $\Gamma = (\Gamma_0, \Gamma_1, \ldots, \Gamma_{n-1})$, the linear mask vector $y = (y_0, y_1, \ldots, y_{n-1})$ corresponding to Γ is defined as

$$y_i = \begin{cases} 0 & \text{if } \Gamma_i = 0 \ , \\ 1 & \text{otherwise} \ . \end{cases}$$

The duality between differential and linear cryptanalysis was already pointed out by Matsui [19]. The constraints describing a linear function are the same as in the case for differential cryptanalysis, however the differential branch number $\mathcal{B}_D$ is replaced by the linear branch number $\mathcal{B}_L$. The linear branch number is the minimum number of non-zero linear masks for the input and output of a function, excluding the all-zero case. No extra constraints are introduced for the XOR operations, because the input and output linear masks are the same.

For a three-forked branch, we proceed as follows. Let the input linear mask vector for the three-forked branch be $y^{\vdash}_{in}$, and the corresponding output linear mask vector be $(y^{\vdash}_{out_1}, y^{\vdash}_{out_2})$. We introduce a binary dummy variable $l^{\vdash}$ to generate the following linear constraints for the three-forked branch:

$$\begin{aligned} y^{\vdash}_{in} + y^{\vdash}_{out_1} + y^{\vdash}_{out_2} &\geq 2l^{\vdash} \ , \\ l^{\vdash} &\geq y^{\vdash}_{in} \ , \\ l^{\vdash} &\geq y^{\vdash}_{out_1} \ , \\ l^{\vdash} &\geq y^{\vdash}_{out_2} \ . \end{aligned}$$

3 Description of Enocoro-128v2

The first Enocoro specification was given in [24]. Enocoro is a stream cipher, inspired by the Panama construction [9]. Two versions of Enocoro were specified: Enocoro-80v1 with a key size of 80 bits, and Enocoro-128v1 with a key size of 128 bits. Later, a new version for the 128-bit key size appeared in [14]. It is referred to as Enocoro-128v1.1. We now give a short description of Enocoro-128v2. For more details, we refer to the design document [25, 26].

Internal State.

The internal state of Enocoro-128v2 is composed of a buffer b consisting of 32 bytes $(b_0, b_1, \ldots, b_{31})$ and a state a consisting of two bytes (a_0, a_1). The initial state is

loaded with a 128-bit key K and a 64-bit IV I as follows:

$$b_i^{(-96)} = K_i, \qquad 0 \le i < 16 \ ,$$
$$b_{i+16}^{(-96)} = I_i, \qquad 0 \le i < 8 \ .$$

All other internal state bytes are loaded with predefined constants.

Update Function.

The update function $Next$ uses functions ρ and λ to update the internal state as follows:

$$(a^{(t+1)}, b^{(t+1)}) = Next(S^{(t)}) = (\rho(a^{(t)}, b^{(t)}), \lambda(a^{(t)}, b^{(t)})) \ .$$

An schematic overview of this function is given in Fig. 1.

Function ρ.

The function ρ updates the state a. It consists of an 8-bit S-box operation, a linear transformation L and XORs. The transformation L is defined as a linear transformation with a 2-by-2 matrix over GF(2^8):

$$\begin{pmatrix} v_0 \\ v_1 \end{pmatrix} = L(u_0, u_1) = \begin{pmatrix} 1 & 1 \\ 1 & d \end{pmatrix} \begin{pmatrix} u_0 \\ u_1 \end{pmatrix} \ , \qquad d \in \mathrm{GF}(2^8) \ ,$$

where $d =$ `0x02`, $u_0 = a_0^{(t)} \oplus S[b_2^{(t)}]$ and $u_1 = a_1^{(t)} \oplus S[b_7^{(t)}]$. The updated state $(a_0^{(t+1)}, a_1^{(t+1)})$ is then calculated as follows:

$$a_0^{(t+1)} = v_0 \oplus S[b_{16}^{(t)}] \ ,$$
$$a_1^{(t+1)} = v_1 \oplus S[b_{29}^{(t)}] \ .$$

Function λ.

The λ function of Enocoro-128v2 consists of XOR operations and a byte-wise rotation of the buffer b. It is defined as follows:

$$b_i^{(t+1)} = \begin{cases} b_{31}^{(t)} \oplus a_0^{(t)}, & \text{if } i = 0 \ , \\ b_2^{(t)} \oplus b_6^{(t)}, & \text{if } i = 3 \ , \\ b_7^{(t)} \oplus b_{15}^{(t)}, & \text{if } i = 8 \ , \\ b_{16}^{(t)} \oplus b_{28}^{(t)}, & \text{if } i = 17 \ , \\ b_{i-1}^{(t)} & \text{otherwise} \ . \end{cases}$$

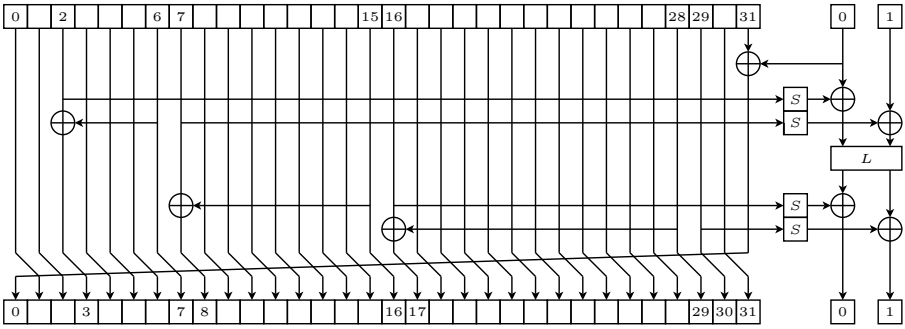


Figure 1 – State Update during the Initialization of Enocoro-128v2. Indices of buffer (on the left) refer to b-variables, indices of the state (on the right) refer to a-variables.

Output Function Out**.**

After 96 initialization rounds, the Enocoro-128v2 output function outputs the lower byte of the state.

$$Out(S^{(t)}) = a_1^{(t)} \ .$$

Several results [13, 16, 17, 20, 26] on differential and linear cryptanalysis have already been published for different versions of Enocoro. In this paper, we consider the most recent version Enocoro-128v2 [25, 26] as an example to illustrate our technique. Watanabe *et al.* already showed that at least $2^{177.8}$ chosen IVs are required for a differential attack on Enocoro-128v2 [26]. For a linear attack, Konosu *et al.* [17] showed that 2^{216} known IVs are required for an attack on the 64-round variant Enocoro-128v1.1. Although these results are already sufficient to prove the security of Enocoro-128v2 against linear and differential cryptanalysis, we explain in this paper how to prove the security against these attacks in a much easier way.

4 Differential Cryptanalysis of Enocoro-128v2

Our technique is now used to find the minimum number of active S-boxes for the stream cipher Enocoro-128v2. We consider an idealized variant of Enocoro-128v2, for which the minimum number of active S-boxes is a lower bound for the real Enocoro-128v2. In this idealized variant of Enocoro-128v2, the S-boxes can map any non-zero input difference to any non-zero output difference. The same holds for the L-function, with the restriction that the branch number is 3.

For this idealized variant of Enocoro-128v2, we have written a program to calculate the minimum number of active S-boxes. We present our problem as a mixed-integer linear programming (MILP) problem, and use CPLEX to solve

it. The solution corresponds to the minimum number of differentially active S-boxes for Enocoro-128v2. It is used to prove the security of the cipher against differential cryptanalysis, using a similar proof as for the block cipher AES [11,12]. Note that an actual characteristic with the given number of active S-boxes may or may not exist, depending on the specific S-box and L-function that is used. This is not a concern for us, as our goal is to prove a security bound against differential cryptanalysis.

4.1 Constructing the MILP Program

Enocoro-128v2 has eight XOR operations and one linear transformation L in each round. We represent the differential behavior of each of these operations by a set of linear inequality constraints, as described in Sect. 2. Let us take the first round of Enocoro-128v2 as an example. The initial difference vector in the buffer and states is represented by the binary variables $(x_0, x_1, \ldots, x_{31})$ and (x_{32}, x_{33}) respectively. Let us consider the XOR operation which has the rightmost byte of buffer b, i.e. b_{31}, and state byte a_0 as inputs. These correspond to binary variables x_{31} and x_{32} respectively, the input difference vector for this XOR operation. From the update function, we can obtain the corresponding value of the leftmost byte of buffer b, i.e. b_0, after the first round. Let the corresponding output difference vector be x_{34}, which is the first new binary variable that we introduce. After introducing a binary dummy variable d_0, this XOR operation can be described by the constraints:

$$\begin{aligned} x_{31} + x_{32} + x_{34} &\geq 2d_0 \ , \\ d_0 &\geq x_{31} \ , \\ d_0 &\geq x_{32} \ , \\ d_0 &\geq x_{34} \ . \end{aligned}$$

We now consider the second XOR operation, for which buffer b_2 (input to the first S-box) and the state a_0 are the inputs. Because the S-box is bijective, it is not only the case that the zero input difference results in a zero output difference, but also that a non-zero input difference results in a non-zero output difference. We find that (x_2, x_{32}) is the difference vector of the second XOR operation. The second new variable, x_{35}, will be the output difference vector for this second XOR operation. Similarly, for the third XOR operation, the input difference vector is (x_7, x_{33}) (corresponding to (b_7, a_1)), and the output difference vector is x_{36}. Given two binary dummy variables d_1 and d_2 for the second and third XOR operation respectively, we again obtain four linear constraints for every XOR operation.

From the structure of the linear transformation of Enocoro-128v2, we know that (x_{35}, x_{36}) is the input difference vector for the linear transformation L in the first round. By introducing a new binary variable d_3, the relations between the output difference vector (x_{37}, x_{38}) and the input difference vector (x_{35}, x_{36}) are

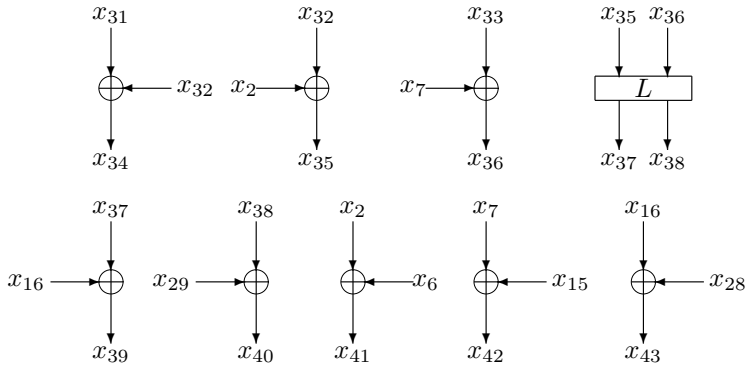


Figure 2 – Difference Vectors for Nine Operations in the First Round

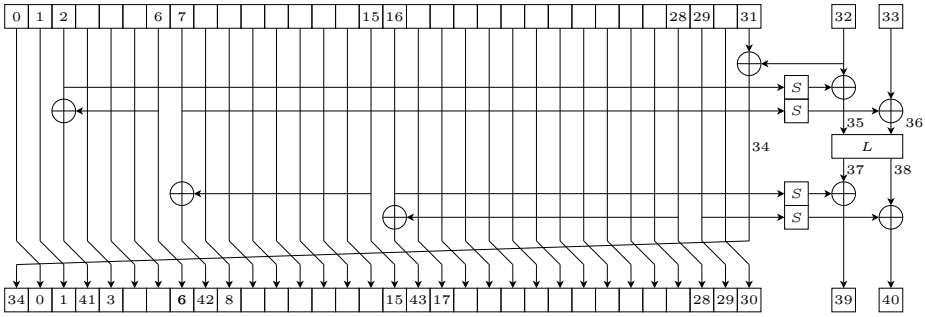


Figure 3 – Differential State Update during the Initialization of Enocoro-128v2. The indices refer to x-variables.

easily described by the following constraints:

$$\begin{aligned} x_{35} + x_{36} + x_{37} + x_{38} &\geq 3d_3 \ , \\ d_3 &\geq x_{35} \ , \\ d_3 &\geq x_{36} \ , \\ d_3 &\geq x_{37} \ , \\ d_3 &\geq x_{38} \ . \end{aligned}$$

The other five XORs in the first round are represented in a similar way. The new variables x_{39}, x_{40}, x_{41}, x_{42} and x_{43} are shown in Fig. 2. These constraints result in the binary dummy variables d_4, d_5, d_6, d_7, d_8. For all the eight XORs and one linear transformation L, ten new binary variables $x_{34}, x_{35}, \ldots, x_{43}$ and nine binary dummy variables $d_0, d_1, \ldots, d_8$ are required. Therefore, a system of

$4 \cdot 8 + 5 \cdot 1 = 37$ constraints is obtained to describe all the nine operations in the first round (and also every subsequent round) of Enocoro-128v2. The detailed input and output vectors for all the nine operations are shown in Fig. 2.

After one round the difference vector for buffer and state will be

$$(x_{34}, x_0, x_1, x_{41}, x_3, \ldots, x_6, x_{42}, x_8, \ldots, x_{15}, x_{43}, x_{17}, \ldots, x_{30})$$

and (x_{39}, x_{40}) respectively. All binary x_i-variables obtained for the first round are illustrated in Fig. 3. Therefore, using this technique we can represent the differential update of Enocoro-128v2 for any round with a system of linear constraints.

4.2 The Minimum Number of Active S-boxes for Differential Cryptanalysis

We now focus on the variables that represent the S-box inputs in every round. Note that x_2, x_7, x_{16}, and x_{29} correspond to the input differences of the S-boxes, and therefore determine if the S-box is active or not. Let D_i include the four indices of variables that represent the four S-box inputs in the i-th round ($1 \leq i \leq 96$). The 96 sets include the indices for variables that represent the four S-box inputs in each round. They can easily be obtained from Sect. 4.1, and are as follows:

$$\begin{aligned}
D_1 &= \{2, 7, 16, 29\} \ , \\
D_2 &= \{1, 6, 15, 28\} \ , \\
D_3 &= \{0, 5, 14, 27\} \ , \\
D_4 &= \{34, 4, 13, 26\} \ , \\
D_5 &= \{44, 3, 12, 25\} \ , \\
&\vdots \\
D_{96} &= \{954, 941, 902, 863\} \ .
\end{aligned}$$

Let k_N be the number of active S-boxes for N rounds of Enocoro-128v2. If

$$I_N = \bigcup_{1 \leq i \leq N} D_i \ ,$$

then

$$k_N = \sum_{i \in I_N} x_i$$

will be the number of active S-boxes in N rounds of Enocoro-128v2. To avoid the trivial case where no S-boxes are active, we add an extra linear constraint to specify that least one S-box is active. If we can minimize the linear function $k_N = \sum_{i \in I_N} x_i$, it will give us the minimum number of active S-boxes for N rounds of Enocoro-128v2. This will provide a security bound for Enocoro-128v2 against differential cryptanalysis. The objective function $k_N = \sum_{i \in I_N} x_i$ is a linear

function, constrained by a system of $37N$ linear inequalities. If all variables must be binary variables, this corresponds to an ILP program.

It is easy to verify that the maximum differential probability for the 8-bit S-box of Enocoro-128v2 is $2^{-4.678}$. As the IV is limited to 64 bits, there are at most 2^{64} IV pairs for any given difference (if the key is fixed). Because there exists a generic attack with a data complexity of 2^{64} IVs (obtaining the entire codebook under one key), attacks requiring 2^{64} IVs or more should not be feasible. Therefore, we do not consider attacks using more than 2^{64} IVs, even in a related-key setting. If the number of active S-boxes in the initialization rounds is at least $14 > 64/4.678$, we consider the cipher to be resistant against differential cryptanalysis. Because we allow differences in both the key and the IV, our results hold both in single-key and related-key settings. We note that typically, differential and linear cryptanalysis are used to attack a few more rounds than the number of rounds of the characteristic. The cipher must also be resistant against other types of attacks and add extra rounds to provide a security margin. For these reasons, more rounds should be used than suggested by our analysis.

In order to optimize the MILP program, we use CPLEX. The experiments are implemented on a 24-core Intel Xeon X5670 @ 2.93 GHz, with 16 GB of RAM. Because this computer is shared with other users, execution times may be longer than necessary, which is why we do not give timing information for all problem instances. We found that it takes about 52.68 seconds to show that the minimum number of active S-boxes for 38 rounds of Enocoro-128v2 is 14. Therefore, 38 rounds of Enocoro-128v2 or more are secure against differential cryptanalysis. The minimum number of active S-boxes for each round of Enocoro-128v2 are listed in Table 1.

We would like to point out to the reader, that the seemingly complex bookkeeping of variable indices should not be a concern for the cryptanalyst who wishes to use this technique. The MILP linear constraints can be generated by a small computer program. This program keeps track of the next unused x- and d-variables. It is then easy to replace every XOR and L function operation in the reference implementation of the cipher by a function to generate the corresponding constraints, and every S-box application by a function that constructs the objective function. For a typical cipher, this should not require more than half an hour of work for a minimally experienced programmer.

If all d-variables are restricted to binary variables, as well as variables x_0 up to x_{33}, the constraints ensure that the optimal solution for all other x_i-variables will be binary as well. Therefore, similar to Borghoff's suggestion in [6], we might solve an MILP program where only the d-variables and x_0 up to x_{33} are binary variables, instead of a pure ILP program. We find that Borghoff's observation can give dramatic speed-ups in some cases: for 72 rounds, it takes 5,808.15 seconds using an MILP, compared 342,747.78 seconds using a pure ILP. However, our MILP program for 38 rounds takes longer: 75.68 seconds instead of 52.68 seconds. Explaining this phenomenon seems to be a useful direction for future work.

Table 1 – Minimum Number of Differentially Active S-boxes $\min(k_N)$ for N rounds of Enocoro-128v2

N	$\min(k_N)$	N	$\min(k_N)$	N	$\min(k_N)$	N	$\min(k_N)$	N	$\min(k_N)$
1	0	21	2	41	16	61	25	81	39
2	0	22	3	42	17	62	26	82	39
3	0	23	3	43	18	63	27	83	40
4	0	24	3	44	18	64	27	84	40
5	0	25	4	45	18	65	28	85	40
6	0	26	5	46	19	66	29	86	41
7	0	27	7	47	20	67	30	87	42
8	0	28	8	48	20	68	30	88	43
9	0	29	8	49	21	69	30	89	43
10	0	30	8	50	22	70	31	90	44
11	0	31	8	51	22	71	32	91	44
12	0	32	9	52	22	72	34	92	45
13	1	33	9	53	22	73	35	93	45
14	1	34	10	54	22	74	35	94	46
15	1	35	11	55	22	75	36	95	47
16	1	36	12	56	22	76	37	96	47
17	1	37	13	57	23	77	37		
18	1	38	14	58	23	78	38		
19	1	39	15	59	24	79	38		
20	2	40	15	60	24	80	38		

5 Linear Cryptanalysis of Enocoro-128v2

We will use our technique to analyze an ideal variant of Enocoro-128v2 for linear cryptanalysis. Similarly as for differential cryptanalysis, the real Enocoro-128v2 will have at least as many linearly active S-boxes as the idealized one, and therefore can be used to prove a security bound.

5.1 Constructing the MILP Program

We now illustrate our technique by presenting the constraints for the first round of the stream cipher Enocoro-128v2 for linear cryptanalysis. For the initial state, let the linear mask vector for the buffer be $(y_0, y_1, \ldots, y_{31})$, and for the state be (y_{32}, y_{33}). Consider the three-forked branch, which has the state byte a_0 as the input linear mask and buffer byte b_{31} as one output linear mask. We obtain the first new binary variable y_{34} as the other output vector. The input and output linear mask vector for this three-forked branch are then y_{32} and (y_{31}, y_{34}) respectively. By introducing the binary dummy variable l_0, the four constraints describing the

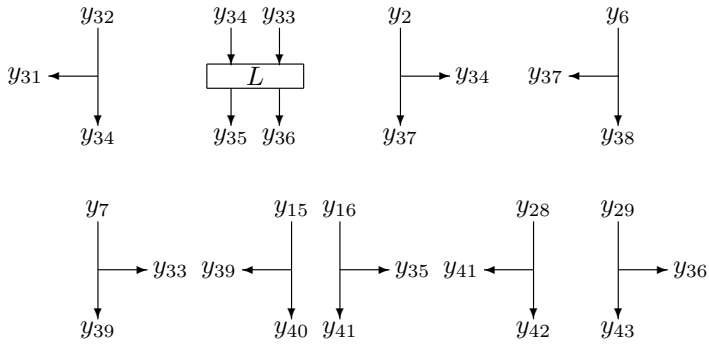


Figure 4 – Linear Mask Vectors for Nine Operations in the First Round

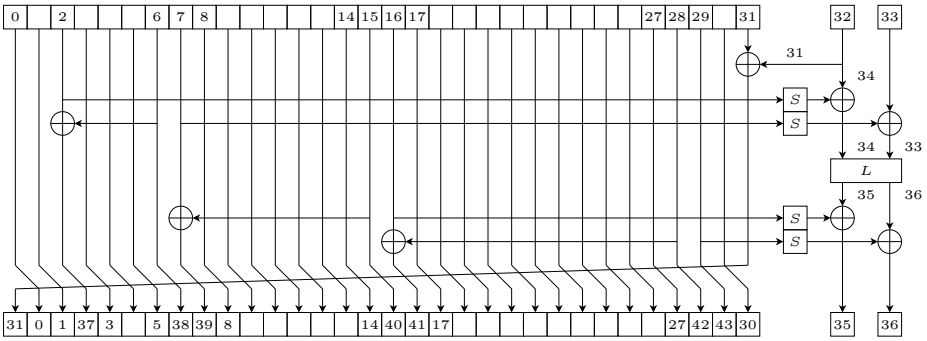


Figure 5 – Linear Mask Vectors Update during the Initialization of Enocoro-128v2. The indices refer to y-variables.

three-forked branch can be described as follows:

$$\begin{aligned} y_{31} + y_{32} + y_{34} &\geq 2l_0 \ , \\ l_0 &\geq y_{31} \ , \\ l_0 &\geq y_{32} \ , \\ l_0 &\geq y_{34} \ . \end{aligned}$$

For the XOR operation, the two inputs and the output all have the same linear mask. The bijectiveness of the S-box implies the linear mask at the output will be non-zero if and only if the input mask is non-zero. Therefore, the linear transformation L has an input linear mask vector of (y_{34}, y_{33}), and an output linear mask vector of (y_{35}, y_{36}). Using a new binary dummy variable l_1, the constraints

describing the L transformation are:

$$\begin{aligned} y_{34} + y_{33} + y_{35} + y_{36} &\geq 3l_1 \ , \\ l_1 &\geq y_{34} \ , \\ l_1 &\geq y_{33} \ , \\ l_1 &\geq y_{35} \ , \\ l_1 &\geq y_{36} \ . \end{aligned}$$

As an Enocoro-128v2 round contains eight three-forked branch operations and one linear transformation L, ten new binary variables $y_{34}, y_{35}, \ldots, y_{43}$, as well as nine binary dummy variables $l_0, l_1, \ldots, l_8$ are introduced. Therefore, $4 \cdot 8 + 5 \cdot 1 = 37$ constraints are required to describe the propagation of linear masks for the first round (as well as any subsequent round) of Enocoro-128v2. The input and output linear mask vectors for all nine operations in the first round are shown in Fig. 4. The linear mask vector for the buffer and state after one round are

$$(y_{31}, y_0, y_1, y_{37}, y_3, \cdots, y_5, y_{38}, y_{39}, y_8, \cdots, y_{14}, y_{40}, y_{41}, y_{17}, \cdots, y_{27}, y_{42}, y_{43}, y_{30})$$

and (y_{35}, y_{36}) respectively. They are shown in Fig. 5.

5.2 The Minimum Number of Active S-boxes for Linear Cryptanalysis

Using the technique in the previous section, we can represent any number of rounds of Enocoro-128v2. We now explain how to calculate the number of active S-boxes. Let L_i include all indices of the four variables representing the input linear mask vector of S-boxes in the i-th round ($1 \leq i \leq 96$). We then obtain the following 96 sets:

$$\begin{aligned} L_1 &= \{34, 33, 35, 36\} \ , \\ L_2 &= \{44, 36, 45, 46\} \ , \\ L_3 &= \{54, 46, 55, 56\} \ , \\ L_4 &= \{64, 56, 65, 66\} \ , \\ L_5 &= \{74, 66, 75, 76\} \ , \\ &\vdots \\ L_{96} &= \{984, 976, 985, 986\} \ . \end{aligned}$$

Let m_N be the number of active S-boxes for N rounds of Enocoro-128v2. If

$$J_N = \bigcup_{1 \leq j \leq N} L_j \ ,$$

then

$$m_N = \sum_{j \in J_N} y_j$$

will be the number of active S-boxes for N rounds of Enocoro-128v2. By minimizing the linear objective function m_N, we obtain the minimum number of linearly active S-boxes for N rounds of Enocoro-128v2.

The maximum correlation amplitude of the 8-bit S-box of Enocoro-128v2 is $C_{\max} = 2^{-2}$. For the same reasons as for differential cryptanalysis, we limit the number of IVs to 2^{64}. Let us denote the minimum number of active S-boxes by a. From the limit on the number of IVs, we then find that resistance against linear cryptanalysis requires [12, pp. 142–143]:

$$C_{\max}^a = (2^{-2})^a \leq 2^{-64/2} \quad .$$

This inequality is satisfied for $a \geq 16$. Therefore, if the number of linearly active S-boxes is at least 16, Enocoro-128v2 can be considered to be resistant against linear cryptanalysis (in both single-key and related-key settings).

If we solve the resulting MILP problem using CPLEX, we find that the minimum number of active S-boxes is 18 for 61 rounds of Enocoro-128v2. This result was obtained after 227.38 seconds. Therefore, we conclude that Enocoro-128v2 with 96 initialization rounds is secure against linear cryptanalysis (in both single-key and related-key settings). Table 2 lists the minimum number of active S-boxes for Enocoro-128v2.

6 Future Work

It is interesting to investigate how the internal parameters of CPLEX can be fine-tuned to calculate bounds against linear and differential cryptanalysis in the fastest possible time. If there are symmetries in the round function, these may be used to speed up the search as well. Similarly, the attacker may improve a given (suboptimal) lower bound for a particular cipher by clocking the round functions forward or backward in order to obtain a lower number of S-boxes. To obtain a rough lower bound for a large number of rounds, the "split approach" (see for example [2]) may be used. For example, if r rounds of a cipher contain at least a active S-boxes, then kr rounds of a cipher must contain at least ka active S-boxes. It is useful to explore how these observations can be applied when CPLEX takes a very long time to execute. Otherwise, the shorter solving time does not compensate for the additional time to construct the program. For ILP programs with a very long execution time, it may be better to calculate the minimum number of active S-boxes using a different technique (e.g. [2]).

The technique in this paper is quite general, and may also be used for truncated differentials, higher-order differentials, impossible differentials, saturation

Table 2 – Minimum Number of Linearly Active S-boxes $\min(m_N)$ for N rounds of Enocoro-128v2

N	$\min(m_N)$	N	$\min(m_N)$	N	$\min(m_N)$	N	$\min(m_N)$	N	$\min(m_N)$
1	0	21	0	41	6	61	18	81	24
2	0	22	0	42	9	62	18	82	27
3	0	23	0	43	9	63	18	83	27
4	0	24	0	44	9	64	18	84	27
5	0	25	0	45	12	65	18	85	27
6	0	26	0	46	12	66	18	86	27
7	0	27	0	47	12	67	18	87	27
8	0	28	0	48	12	68	21	88	27
9	0	29	0	49	12	69	21	89	27
10	0	30	0	50	12	70	21	90	27
11	0	31	0	51	12	71	21	91	27
12	0	32	0	52	15	72	21	92	27
13	0	33	3	53	15	73	21	93	30
14	0	34	6	54	15	74	21	94	30
15	0	35	6	55	15	75	21	95	33
16	0	36	6	56	15	76	24	96	33
17	0	37	6	57	15	77	24		
18	0	38	6	58	15	78	24		
19	0	39	6	59	15	79	24		
20	0	40	6	60	15	80	24		

attacks,... It can also be applied to other ciphers constructed using S-box operations, linear permutation layers, three-forked branches and/or XOR operations. We leave the exploration of these topics to future work as well.

7 Conclusion

In this paper, we introduced a simple technique to calculate the security of many ciphers against linear and differential cryptanalysis. The only requirement is that the cipher is composed of a combination of S-box operations, linear permutation layers and/or XOR operations. Our technique involves writing a simple program to generate a mixed-integer linear programming (MILP) problem. The objective function of the MILP program is the number of linearly or differentially active S-boxes, which we want to minimize. This MILP problem can then easily be solved using an off-the-shelf optimization package, for example CPLEX. The result can be used to prove the security of a cryptosystem against linear and differential cryptanalysis.

Our technique can be applied to a wide variety of cipher constructions. As an

example, we apply the technique in this paper to the stream cipher Enocoro-128v2. We prove that for Enocoro-128v2 38 rounds are sufficient for security against differential cryptanalysis, and 61 rounds against linear cryptanalysis. These results are valid both in single-key and related-key models. As Enocoro-128v2 consists of 96 initialization rounds, this proves the security of Enocoro-128v2 against linear and differential cryptanalysis.

We would like to point out that only little programming is required to obtain this result. A minimally experienced programmer can modify the reference implementation of a cipher, in order to generate the required MILP program in about half an hour. In the case of Enocoro-128v2, it takes CPLEX less than one minute on a 24-core Intel Xeon X5670 processor to prove security against differential cryptanalysis, and less than four minutes to prove security against linear cryptanalysis. We note that because very little programming is required, both the time spent on cryptanalysis and the possibility of making errors are greatly reduced.

Acknowledgments. The authors would like to thank their colleagues at COSIC, as well as the anonymous reviewers for their detailed comments and suggestions. Special thanks to Hirotaka Yoshida for reviewing an earlier draft of this paper.

References

[1] E. Biham and A. Shamir. Differential Cryptanalysis of DES-like Cryptosystems. *J. Cryptology*, 4(1):3–72, 1991.

[2] A. Biryukov and I. Nikolic. Search for Related-Key Differential Characteristics in DES-Like Ciphers. In A. Joux, editor, *FSE*, volume 6733 of *Lecture Notes in Computer Science*, pages 18–34. Springer, 2011.

[3] A. Bodganov. Personal Communication, 2011.

[4] A. Bogdanov. *Analysis and Design of Block Cipher Constructions*. PhD thesis, Ruhr University Bochum, 2009.

[5] A. Bogdanov. On unbalanced Feistel networks with contracting MDS diffusion. *Des. Codes Cryptography*, 59(1-3):35–58, 2011.

[6] J. Borghoff, L. R. Knudsen, and M. Stolpe. Bivium as a Mixed-Integer Linear Programming Problem. In M. G. Parker, editor, *IMA Int. Conf.*, volume 5921 of *Lecture Notes in Computer Science*, pages 133–152. Springer, 2009.

[7] C. Bouillaguet, P.-A. Fouque, and G. Leurent. Security Analysis of SIMD. In A. Biryukov, G. Gong, and D. R. Stinson, editors, *Selected Areas in Cryptography*, volume 6544 of *Lecture Notes in Computer Science*, pages 351–368. Springer, 2010.

[8] J. P. Cameron McDonald, Chris Charnes. An Algebraic Analysis of Trivium Ciphers based on the Boolean Satisfiability Problem. Cryptology ePrint Archive, Report 2007/129, 2007. http://eprint.iacr.org/.

[9] J. Daemen and C. S. K. Clapp. Fast Hashing and Stream Encryption with PANAMA. In S. Vaudenay, editor, *FSE*, volume 1372 of *Lecture Notes in Computer Science*, pages 60–74. Springer, 1998.

[10] J. Daemen, R. Govaerts, and J. Vandewalle. Resynchronization Weaknesses in Synchronous Stream Ciphers. In *EUROCRYPT*, pages 159–167, 1993.

[11] J. Daemen and V. Rijmen. The Wide Trail Design Strategy. In B. Honary, editor, *IMA Int. Conf.*, volume 2260 of *Lecture Notes in Computer Science*, pages 222–238. Springer, 2001.

[12] J. Daemen and V. Rijmen. *The Design of Rijndael: AES - The Advanced Encryption Standard*. Springer, 2002.

[13] M. Hell and T. Johansson. Security Evaluation of Stream Cipher Enocoro-128v2. CRYPTREC Technical Report, 2010.

[14] D. W. K. Muto and T. Kaneko. Strength evaluation of Enocoro-128 against LDA and its Improvement. In *Symposium on Cryptography and Information Security*, pages 4A1–1, 2008. (in Japanese).

[15] M. Kanda. Practical Security Evaluation against Differential and Linear Cryptanalyses for Feistel Ciphers with SPN Round Function. In D. R. Stinson and S. E. Tavares, editors, *Selected Areas in Cryptography*, volume 2012 of *Lecture Notes in Computer Science*, pages 324–338. Springer, 2000.

[16] K. M. Kazuto Okamoto and T. Kaneko. Security evaluation of Pseudorandom Number Generator Enocoro-80 against Differential/Linear Cryptanalysis (II). In *Symposium on Cryptography and Information Security*, pages 20–23, 2009. (in Japanese).

[17] K. Konosu, K. Muto, H. Furuichi, D. Watanabe, and T. Kaneko. Security evaluation of Enocoro-128 ver.1.1 against resynchronization attack. IEICE Technical Report, ISEC2007-147, 2008. (in Japanese).

[18] M. Matsui. Linear Cryptoanalysis Method for DES Cipher. In *EUROCRYPT*, pages 386–397, 1993.

[19] M. Matsui. On Correlation Between the Order of S-boxes and the Strength of DES. In *EUROCRYPT*, pages 366–375, 1994.

[20] K. Muto, D. Watanabe, and T. Kaneko. Security evaluation of Enocoro-80 against linear resynchronization attack. Symposium on Cryptography and Information Security, 2008. (in Japanese).

[21] H. Raddum. Cryptanalytic Results on Trivium. eSTREAM report 2006/039, 2006. http://www.ecrypt.eu.org/stream/triviump3.html.

[22] L. Schrage. *Optimization Modeling with LINGO*. Lindo Systems, 1999. http://www.lindo.com.

[23] K. Shibutani. On the Diffusion of Generalized Feistel Structures Regarding Differential and Linear Cryptanalysis. In A. Biryukov, G. Gong, and D. R. Stinson, editors, *Selected Areas in Cryptography*, volume 6544 of *Lecture Notes in Computer Science*, pages 211–228. Springer, 2010.

[24] D. Watanabe and T. Kaneko. A construction of light weight Panama-like keystream generator. In *IEICE Technical Report, ISEC2007-78*, 2007. (in Japanese).

[25] D. Watanabe, K. Okamoto, and T. Kaneko. A Hardware-Oriented Light Weight Pseudo-Random Number Generator Enocoro-128v2. In *The Symposium on Cryptography and Information Security*, pages 3D1–3, 2010. (in Japanese).

[26] D. Watanabe, T. Owada, K. Okamoto, Y. Igarashi, and T. Kaneko. Update on Enocoro Stream Cipher. In *ISITA*, pages 778–783. IEEE, 2010.

[27] S. Wu and M. Wang. Security Evaluation against Differential Cryptanalysis for Block Cipher Structures. Cryptology ePrint Archive, Report 2011/551, 2011. http://eprint.iacr.org/.

A Number of Active S-boxes for AES

The four-round propagation theorem of AES [12] proves that the number of active S-boxes in a differential or linear characteristic of four AES rounds is at least 25. Combined with the properties of the AES S-box, this result was used in the AES design document to prove the resistance against linear and differential attacks. In this section, we illustrate our technique by applying it to the block cipher AES. We not only confirm the four-round propagation theorem, but also determine the minimum number of active S-boxes for up to 14 rounds in Table 4.

An AES round update consists of four operations: AddRoundKey (AR), SubBytes (SB), ShiftRows (SR) and MixColumns (MC). The update of the first AES round is shown in Table 3. Every variable corresponds to a byte of the AES state. The variable is 1 if the difference is non-zero, and 0 if the difference is zero. All variables corresponding to the inputs of the SubByte operations are summed in the objective function, this corresponds to the number of active S-boxes. The linear function used in the MixColumns operation has a differential as well as a linear branch number of 5.

A program was written in C to generate the constraints for this optimization problem in the CPLEX LP format. To illustrate the simplicity of our technique,

we provide this program (including source code comments) below in full. None of the optimization problems in Table 4 took longer than 0.40 seconds to solve, using only a single core of our 24-core Intel Xeon X5670 processor.

Table 3 – The Variables in the First Round Update of AES

$$\begin{bmatrix} x_0 & x_4 & x_8 & x_{12} \\ x_1 & x_5 & x_9 & x_{13} \\ x_2 & x_6 & x_{10} & x_{14} \\ x_3 & x_7 & x_{11} & x_{15} \end{bmatrix} \xrightarrow{\text{SB}} \begin{bmatrix} x_0 & x_4 & x_8 & x_{12} \\ x_1 & x_5 & x_9 & x_{13} \\ x_2 & x_6 & x_{10} & x_{14} \\ x_3 & x_7 & x_{11} & x_{15} \end{bmatrix} \xrightarrow{\text{SR}} \begin{bmatrix} x_0 & x_4 & x_8 & x_{12} \\ x_5 & x_9 & x_{13} & x_1 \\ x_{10} & x_{14} & x_2 & x_6 \\ x_{15} & x_3 & x_7 & x_{11} \end{bmatrix} \xrightarrow{\text{MC}} \begin{bmatrix} x_{16} & x_{20} & x_{24} & x_{28} \\ x_{17} & x_{21} & x_{25} & x_{29} \\ x_{18} & x_{22} & x_{26} & x_{30} \\ x_{19} & x_{23} & x_{27} & x_{31} \end{bmatrix}$$

Table 4 – Minimum Number of Differentially or Linearly Active S-boxes $\min(k_N)$ for N rounds of AES

N	1	2	3	4	5	6	7	8	9	10	11	12	13	14
$\min(k_N)$	1	5	9	25	26	30	34	50	51	55	59	75	76	80

```
#include <stdio.h>
int i,j,r;
const int ROUNDS = 4; /* number of rounds */
int next = 0; /* next unused state variable index */
int dummy = 0; /* next unused dummy variable index */

void ShiftRows(int a[4][4]) {
  int tmp[4];
  for(i = 1; i < 4; i++) {
    for(j = 0; j < 4; j++) tmp[j] = a[i][(j + i) % 4];
    for(j = 0; j < 4; j++) a[i][j] = tmp[j];
  }
}
void MixColumns(int a[4][4]) {
  for(j = 0; j < 4; j++) {
    for (i = 0; i < 4; i++) printf("x%i + ",a[i][j]);
    for (i = 0; i < 3; i++) printf("x%i + ",next+i);
    printf("x%i - 5 d%i >= 0\n",next+3,dummy);

    for(i = 0; i < 4; i++)
      printf("d%i - x%i >= 0\n",dummy,a[i][j]);
    for(i = 0; i < 4; i++)
      printf("d%i - x%i >= 0\n",dummy,a[i][j]=next++);
    dummy++;
  }
}
```

```
int main() {
  int a[4][4]; /* the bytes of the AES state */
  for (i = 0; i < 4; i++)
    for (j = 0; j < 4; j++)
      a[i][j] = next++; /* initialize variable indices */

  printf("Minimize\n"); /* print objective function */
  for (i = 0; i < ROUNDS*16-1; i++) printf("x%i + ",i);
  printf("x%i\n\n",ROUNDS*16-1);

  printf("Subject To\n"); /* round function constraints */
  for (r = 0; r<ROUNDS; r++) { ShiftRows(a); MixColumns(a); }

  /* at least one S-box must be active */
  for (i = 0; i < ROUNDS*16-1; i++) printf("x%i + ",i);
  printf("x%i >= 1\n\n",ROUNDS*16-1);

  printf("Binary\n"); /* binary constraints */
  for (i = 0; i < 16; i++) printf("x%i\n",i);
  for (i = 0; i < dummy; i++) printf("d%i\n",i);
  printf ("End\n");
  return 0;
}
```

Curriculum Vitae

Nicky Mouha was born on November 23, 1986 in Tongeren, Belgium. Already during his high school years at Onze-Lieve-Vrouwehumaniora Tongeren (currently known as viio humaniora Tongeren), he excelled in 2003 by reaching the second place in the semifinal of the Flemish Chemistry Olympiad. In the final round, he finished fourth. In the same year, he also competed in the final round in the Belgian Federal Parliament of the school debating competition organized by Junior Chamber International Vlaanderen.

From 2003 to 2006, Nicky studied Bachelor of Science in Electrical Engineering (Option: Electrical Engineering – Computer Science) at KU Leuven, Belgium. Subsequently from 2006 to 2008, he studied Master of Science in Electrical Engineering (Option: Multimedia – Signal Processing) at the same university. He was the only student of the Department of Electrical Engineering (ESAT) to graduate *summa cum laude* in July 2008.

In October 2008, he joined the research group COSIC (Computer Security and Industrial Cryptography) of ESAT under the supervision of Prof. Bart Preneel. The main focus of his Ph.D. research was hash functions, but he obtained several results on block ciphers and stream ciphers as well. His Ph.D. research was supported by a personal grant from IWT-Vlaanderen.

In 2011, he performed research at Tsinghua University, P.R. China as the first non-Chinese Ph.D. student to work under the supervision of Prof. Xiaoyun Wang.

During his Ph.D. studies, Nicky was very active in organizing conferences and workshops. He organized ECRYPT II Bounds for Symmetric Constructions together with Andrey Bogdanov in 2010. In 2011, he was a member of the Organizing Committee of the ECRYPT II Hash Workshop. He also co-organized the COSIC Course 2011. Together with Péla Noë and Saartje Verheyen, he was the official responsible for the practical organization of CARDIS 2011, DPM 2011, EuroPKI 2011, FAST 2011, SETOP 2011 and TrustED 2011. He was also very active in the organization of FSE 2009, RFIDSec09, ECRYPT II Hash^3: Proofs, Analysis, and Implementation and ESORICS 2011.

Starting from 2010, he volunteers at KU Leuven Residence Management as a resident assistant. Studentenwijk Arenberg consists of 820 student rooms. He is personally responsible for the supervision of 110 student rooms, but also organizes events together with his colleagues for all students of Studentenwijk Arenberg.

Arenberg Doctoral School of Science, Engineering & Technology
Faculty of Engineering
Department of Electrical Engineering (ESAT)
Computer Security and Industrial Cryptography (COSIC)
Kasteelpark Arenberg 10 — 2446, 3001 Heverlee, Belgium