KU LEUVEN

ARENBERG DOCTORAL SCHOOL
Faculty of Engineering Science

Cryptanalysis techniques for lightweight symmetric-key primitives

Liliya Kraleva

Supervisors:
Prof. dr. ir. Vincent Rijmen
Prof. dr. Svetla Petkova-Nikova

Dissertation presented in partial fulfilment of the requirements for the degree of Doctor of Engineering Science (PhD): Electrical Engineering

November 2022

Cryptanalysis techniques for lightweight symmetric-key primitives

Liliya KRALEVA

Examination committee:
em. Prof. dr. ir. Yves Willems, chair
Prof. dr. ir. Vincent Rijmen, supervisor
Prof. dr. Svetla Petkova-Nikova, supervisor
Prof. dr. ir. Bart Preneel
Prof. dr. ir. Luc Van Eycken
Prof. dr. Tomer Ashur
(Eindhoven University of Technology)
Prof. DI dr. Christian Rechberger
(Graz University of Technology)

Dissertation presented in partial fulfilment of the requirements for the degree of Doctor of Engineering Science (PhD): Electrical Engineering

November 2022

© 2022 KU Leuven – Faculty of Engineering Science
Uitgegeven in eigen beheer, Liliya Kraleva, Kasteelpark Arenberg 10, bus 2452, B-3001 Leuven (Belgium)

Alle rechten voorbehouden. Niets uit deze uitgave mag worden vermenigvuldigd en/of openbaar gemaakt worden door middel van druk, fotokopie, microfilm, elektronisch of op welke andere wijze ook zonder voorafgaande schriftelijke toestemming van de uitgever.

All rights reserved. No part of the publication may be reproduced in any form by print, photoprint, microfilm, electronic or any other means without written permission from the publisher.

Preface

Doing a PhD was something I never thought I would do. I am extremely happy I made the step to start a PhD, and not just anywhere, but in the best possible group - COSIC. I am pleased to have had the opportunity to work with so many brilliant colleagues, many of which I can warmly call friends!

I am grateful to all the people I met during the last four and a half years at COSIC, including fellow PhD students, professors, visitors, students, and the people from administration - it is impossible to name all of you, but know I thank you for every conversation we shared.

I would like to express my sincere gratitude to my supervisors Vincent Rijmen and Svetla Nikova - you have been a great support for me during my difficult and happy moments. None of this would have been possible without either of you!

My office mates Siemen Dhooghe and Adrián Ranea - you are the best office mates I could have had! Thank you for all the research discussions we had, all the laughs together and all the support you have given me! The thought that I am leaving you and I would never share an office as great (and awkward) as this one saddens me.

I am grateful to all my collaborators for having the opportunity to learn with you and from you. I am especially grateful to Raluca Posteuca, who I consider as a friend, for all the late night calls, all research discussions and brainstorming sessions, all one-hour arguings about changing a single word in our text, and all philosophical talks. Surviving the lock down was possible a big part because of my daily calls with you!

Finally, big thanks to my family and their encouraging before and during my PhD. Thanks to my husband Nikola for sacrificing so much for me and for following me to Belgium, for always being there for me when I needed him and for giving me all the love in the world and an amazing daughter. Thanks to

my mom for everything she has done for me, and to my brother for giving me an example in life by being the first doctor in the family. My father was my biggest inspiration to follow mathematics and I wish he could have seen me graduating and becoming a doctor too! I love you all and thank you!

Abstract

Technologies have been rapidly developing in the last century. Industry is competing to offer faster, smaller, smarter devices to people and is constantly improving in that direction. The desired optimised devices and services bring new platforms with new requirements for performance. In terms of security this means less resources to perform the very needed encryption, decryption or authentication on the local device. Researchers both from industry and academia are looking for solutions to satisfy the tight requirements for cryptographic algorithms and simultaneously provide the desired level of security. New dedicated designs are being proposed, labeled as "lightweight" due to the limited available resources. However, as fast as technologies develop and chips become smaller, research needs its time in order to deeply study and understand new designs. The main force that moves cryptographic designs forward is the cryptanalysis. In this thesis we focus on cryptanalysis of symmetric-key lightweight ciphers with the purpose of providing a better understanding of their design and security. We analyse several important primitives, namely the block cipher K-Cipher, three recently proposed novel PUFs, the stream cipher SNOW3G, the authenticated encryption SpoC and the MAC algorithm Chaskey. Various cryptanalysis techniques are applied, including differential and rotational techniques. We show that basic cryptanalysis can still be used in analysing new lightweight ciphers and urge for careful analysis before using them. Additionally, we discuss the design and implementation decisions involved in the field of "lightweight" symmetric cryptography and provide detailed explanations on how differential and rotational techniques can be used.

Beknopte samenvatting

In de laatste eeuw zijn onze technologieën snel ontwikkeld. De industrie concurreert voor snellere, kleinere en slimmere apparaten wat hun constant duwt om te verbeteren. De fel gegeerde snelste apparaten en diensten brengen nieuwe platformen met hun eigen eisen voor performantie. In termen van veiligheid wilt dit zeggen dat er minder middelen zijn om de broodnodige versleuteling en authenticatie te doen. Onderzoekers van zowel de industrie als de universiteiten zoeken naar oplossingen die tegelijkertijd voldoen aan de hoge veiligheidseisen als een laag kostenplaatje hebben. De nieuwe toegewijde ontwerpen worden voorgesteld onder de naam "pluimgewicht"sinds deze primitieven zeer licht in hun kost zijn. Desalniettemin, ook al wordt onze technologie steeds sneller en onze chips steeds kleiner, onderzoek heeft zijn tijd nodig om de nieuwe primitieven en hun veiligheid correct te bestuderen. De belangrijkste kracht die nieuwe ontwerpen verder duwt is de cryptanalyse. In deze thesis buigen we ons over de cryptanalyse van symmetrische sleutel pluimgewicht primitieven met als doel hun ontwerp en veiligheid beter te begrijpen. We onderzoeken enkele belangrijke primitieven waaronder het blokcijfer K-Cipher, drie recent voorgestelde PUFs, het stroomcijfer SNOW3G, de geauthentiseerde versleutelmethode SpoC en het MAC algoritme Chaskey. Verscheidene cryptanalyse technieken worden toegepast waaronder differentiële en rotatietechnieken. We tonen aan dat de basistechnieken in de cryptanalyse nog steeds gebruikt kunnen worden om nieuwe primitieven te analyseren en we sporen aan een zorgvuldig onderzoek voor zulke primitieven te gebruiken. Bovendien bespreken we het ontwerp en implementatiebeslissingen verwikkeld in het veld van pluimgewicht symmetrische cryptografie en we bieden verscheidene verklaringen hoe differentiële en rotatietechnieken te gebruiken.

Abbreviations

3GPP 3rd Generation Partnership Project
AEAD Authenticated Encryption with Associated data
AES Advanced Encryption Standard
AES-NI Advanced Encryption Standard New Instructions
AND-RX AND Rotation and XOR
API Application Programming Interfaces
ARX Addition-Rotation-XOR
ARX-C Addition-Rotation-XOR with a Constant

CAESER Competition for Authenticated Encryption: Security, Applicability, and Robustness
CBC Cipher Block Chaining (mode of operation)
CFB Ciphertext Feedback (mode of operation)
CRP Challenge-Response Protocol
CTR Counter (mode of operation)
DDT Difference Distribution Table
DES Data Encryption Standard

EtM Encrypt then MAC
E& M Encrypt and MAC
ETSI European Telecommunications Standards Institute
FSM Finite State Machine
GE Gate Equivalence
IC Integrated Circuit
IoT Internet of Things
IPsec Internet Protocol Security
ISO International Organization for Standardization

LFSR Linear Feedback Shift Register

MAC Message Authentication Code
MILP Mixed Integer Linear Programming
ML Machine Learning
MtE MAC then Encrypt

NAND Not-AND
NIST National Institute of Standards and Technology
NFsR Nonlinear Feedback Shift Register
OFB Output Feedback (mode of operation)
PGP Pretty Good Privacy. Software for secure email communication.
PUF Physically Unclonable Function
RNG Random Number Generator
RX-difference Rotational XOR difference
SAT Boolean Satisfiability
SHA Secure Hash Algorithms
SMT Satisfiability Modulo Theories
SPN Substitution-Permutation Network
SSH Secure Shell Protocol

TMTO Time-Memory Trade-Off
TLS Transport Layer Security
SSL Secure Sockets Layer
WEP Wired Equivalent Privacy
XOR Exclusive OR

Contents

Preface i

Abstract iii

Beknopte samenvatting v

Abbreviations vii

List of Symbols ix

Contents ix

List of Figures xiii

I Introduction 1

1 Introduction 3
1.1 Contributions and structure of the thesis 5

2 Symmetric-key primitives 7
2.1 Block ciphers 7
2.1.1 Feistel 8
2.1.2 SPN 10
2.2 Stream ciphers 11
2.3 Hash functions 12
2.4 MAC algorithms 14
2.5 AEAD 15
2.6 ARX 15
2.7 PUFs 17

3 Lightweight Cryptography 19

3.1 Latency . . . 20
3.2 Throughput . . . 20
3.3 Area . . . 20
3.4 Energy . . . 21
3.5 Power . . . 21
3.6 Trade-offs . . . 21
3.6.1 Trade-offs in hardware architectures . . . 22
3.6.2 Trade-offs from a designer's point of view . . . 23
3.7 How to decide . . . 24

4 Symmetric-key cryptanalysis 25
4.1 Adversarial models . . . 25
4.2 Security claims . . . 27
4.3 Cryptanalysis techniques . . . 28
4.3.1 Differential cryptanalysis . . . 29
4.3.2 Differences over a modular addition . . . 33
4.3.3 Rotational cryptanalysis . . . 37
4.3.4 Linear cryptanalysis . . . 39

5 Contributions 41
5.1 New PUF Designs . . . 41
5.2 K-Cipher . . . 42
5.3 The AEAD algorithm SpoC . . . 42
5.4 The stream cipher Snow . . . 43
5.5 The MAC algorithm Chaskey . . . 44
5.6 Correlation distributions . . . 44

6 Conclusions and future work 47

Bibliography 51

II Papers 67

7 Cryptanalysis of Strong Physically Unclonable Functions 69

8 Differential Cryptanalysis of K-Cipher 89

9 On the SNOW family of stream ciphers 105

10 Cryptanalysis of the permutation based algorithm SpoC 115

11 Rotational cryptanalysis on MAc algorithm Chaskey 139

12 Correlation distribution analysis of a two-round key-alternating block cipher 157

List of Figures

2.1 Scheme of a key-alternating block cipher. 9
2.2 Classical Feistel construction and Generalised Feistel network of type 1, 2 and 3. 10
2.3 SPN structure 11

4.1 ARX primitives with the same number of additions, but different rotational probability 38

Part I

Introduction

Chapter 1

Introduction

"If we knew what it was we were doing, it would not be called research, would it?"

- Albert Einstein

From ancient times people have tried to hide information, to pass messages to others secretly and to protect the messages from those who are not supposed to see it. The ways of doing so have developed through the years to the current state, where complex mathematical algorithms are used to protect messages in such a way that they are authentic, genuine and non-repudiable, meaning that the party receiving the message can be sure it is sent by the person saying, it is not changed on the way and the sender can not claim it has been sent at a different time or not at all. Initially used by military forces and governments, nowadays cryptography is used for the personal protection of every individual and their personal information. Modern cryptography is integrated almost everywhere in our daily lives, starting from applications which need to ensure confidentiality and internet protocols, to encryption of sensitive data and privacy preserving applications. IoT, medical devises, cloud encryptions, car sharing and bank applications are a small number of examples. The significance of these applications underlines the importance of proper protection and reliable security algorithms.

Technology has and is still rapidly developing, bringing new, smaller and faster platforms together with the need of new optimised algorithms. Two main approaches can be discerned to design primitives - one that prioritizes the

performance, meaning that often not-well-studied design choices are made in order to have better performance; and the other which prioritizes the confidence in the security level, meaning that the chosen building blocks have been largely studied and can give us approximations we believe are close to the real security level. On the other hand, what makes us trust algorithms is the third-party analysis. There is the provable security approach, which improves on the lower bound of the security level and there is the cryptanalysis approach that improves the upper bound of the security.

The purpose of cryptanalysis is to build trust in algorithms by extensively analyzing them. Many techniques are used to evaluate the security and new ones are developed every day, relying more and more on automated searches and data training. Primitives keep getting broken with the introduction of new techniques which leads us to the thought that no algorithm is, or can be, 100% secure. The fact that there is no attack on it yet does not necessarily mean that there will not be one tomorrow. So what cryptanalysis actually does is improving the confidence in certain algorithms and their claimed or proven security. Each algorithm can be associated with a security level, usually measured with bits of security, and a confidence level, indicating how confident one is that the given algorithm has the corresponding security level that the designers claim or that is achieved by the most successful attack on it. The confidence comes from several factors such as current attacks, provable security, time span of analysis, design choices. For example, we have AES which has high security with very high confidence. It has been analyzed for the past 20 years, has provable security based on the wide trail strategy and the best attack against it is the biclique attack [34] with complexity $2^{126.18}$ over all rounds of AES-128. That brings quite high confidence that AES is secure. An example cipher with high security, but low confidence is Speck. Even though there are no significant attacks over it, almost a decade after it was proposed it is still not trusted, because it was proposed by NSA. It is speculated, that there might be a backdoor to it and the confidence in the security that it claims to provide is therefore low. New designs that have not been analyzed for a long time also have low confidence. If there is a practical attack for some algorithm it follows that its security level is low. However, the confidence that the security it provides is indeed low would be high.

The field of cryptography differentiates between two main branches, namely the symmetric-key cryptography and asymmetric, or public-key cryptography. The first studies algorithms which use the same key for performing both encryption and decryption, while the second uses different keys.

It can be observed that the security of the two branches is based on different concepts. While an attack on a symmetric-key primitive means anything better than exhaustive search, for public-key primitives this is reduced to solving a

well-studied hard mathematical problem such as number factoring and discrete logarithms. In this thesis we focus on symmetric-key primitives and their cryptanalysis

1.1 Contributions and structure of the thesis

The research in this thesis is directed towards lightweight symmetric-key algorithms and their security. Each publication contributes to the state of the art and improves the confidence in the security level of each analysed algorithm. All analysed primitives are diverse in structure and very significant when the research was conducted. Papers [84, 85, 86, 87, 7] are focused on cryptanalysis, while [84, 88] also have theoretical contributions.

At the time of the paper submissions, the MAC algorithm Chaskey [110] was in the process of standardization and has been since standardized, SpoC [4] was a second round candidate in the NIST Lightweight Competition, SNOW3G was part of the 3GPP telecommunications standard, the PUF algorithms [68, 96, 137] were recently proposed with novel designs and the K-cipher [83] was just published with the intention to be used as underlying algorithm for memory encryption.

The first part of this thesis introduces the necessary background and motivations for the research presented in the papers. Chapter 2 presents the most used cryptographic primitives. Examples of their purposes and the usage of concrete algorithms is presented. Chapter 3 introduces the concept of lightweight cryptography. We give arguments on what makes an algorithm lightweight both from the designer and implementation point of view and present guidance to which algorithms are better suited for different applications. Cryptanalysis techniques applicable to all types of primitives are discussed in Chapter 4. With more details are presented differential and rotational cryptanalysis, however other techniques are discussed as well. Section 4.3.2 introduces a work in progress involving differences over modular addition and how to use them in order to reduce the possible key space of a primitive. The contribution of each published paper is described in Chapter 5. Finally, in Chapter 6 are presented concluding remarks and future work directions.

The second part of this thesis consists of all publications of Liliya Kraleva. They are presented in chronological order starting from the latest, in the version that they were published. Liliya's contribution to each of them is described in the beginning of each chapter.

Chapter 2

Symmetric-key primitives

Symmetric-key cryptography has a major role in almost all applications of cryptography. Symmetric-key primitives are the base of many applications such as bank transactions, online shopping, IoT devices, autonomous cars, and many more. The main security goals achieved by cryptographic primitives are:

- **Confidentiality,** meaning that only the genuine receiver can understand the message, and
- **Authenticity,** meaning that the receiver can detect if someone is impersonating the genuine sender or if the message have been tampered with.

It is important to note that the security goals are different and somehow independent, thus the techniques of achieving each of them are also different. Block and stream ciphers are used to encrypt data and provide secrecy, while MAC algorithms authenticate the data. Authenticated Encryption with Associated Data (AEAD) provide both secrecy and integrity simultaneously. On the other hand, hash functions are used to simplify the authentication of data without the need of a secret key. This chapter continues by briefly recalling some of the most used symmetric primitives and their purpose.

2.1 Block ciphers

Block ciphers are the most widely used primitives of today's cryptography. Well-known examples are the current and former standards AES [3] and DES [112].

Block ciphers are used as encryption algorithms by themselves, or as building blocks to other primitives such as permutations, protocols, hash functions, random number generators (RNG), data entity authentication schemes.

Block ciphers operate with a secret key shared only between two parties, and a message. We call the function $E_k : \mathbb{F}_{2^n} \times \mathbb{F}_{2^{n_k}} \to \mathbb{F}_{2^n}$ an encryption function and $D_k : \mathbb{F}_{2^n} \times \mathbb{F}_{2^{n_k}} \to \mathbb{F}_{2^n}$ a decryption function, where for a fixed key k of size n_k and a plaintext m of size n we have $E_k(m) = c$ and $D_k(c) = m$. In addition, it is satisfied that $D_k(E_k(m)) = m$.

A block cipher is considered as a family of n-bit permutations of size 2^n where each secret key yields one permutation out of the $2^n!$ n-bit permutations.

Block ciphers often perform similar operations multiple times by iterating them, each repetition being called a round. In practice, a different key, also called a round key is added to every round and is derived through the master key from a key schedule. Depending on how the key is added to the round function, we can distinguish a special case of ciphers called *key-alternating* block ciphers. In this class the key is added in particularly simple way by XORing it to the state separately from the round function. In each round a round function and key addition are alternating. The first key addition is at the beginning of the first round, and the last is at the end of the last round. An illustration is shown in Figure 2.1.

Firstly introduced by Shannon [132], *confusion* and *diffusion* are main properties a block cipher needs to posses in order to be secure. Diffusion is the property ensuring that the redundancy in the statistics of the plaintext is dissipated in the statistics of the ciphertext. The change of one bit in the plaintext corresponds to a change in all bits of the ciphertext and vice versa. Confusion means that the relation between the ciphertext and the key is very complex: each bit of the ciphertext should depend on the entire key, and in different ways on different bits of the key. Changing one bit of the key should change the ciphertext completely. These properties are often achieved by the (alternating) use of permutation functions and sboxes.

Typical block ciphers design structures are the Feistel and the Substitution Permutation Network (SPN) structures. They both achieve good confusion and diffusion layers.

2.1.1 Feistel

Feistel networks have been proposed by Host Feistel in 1973. The first algorithm with this construction is Lucifer, the predecessor of DES. Lucifer has been

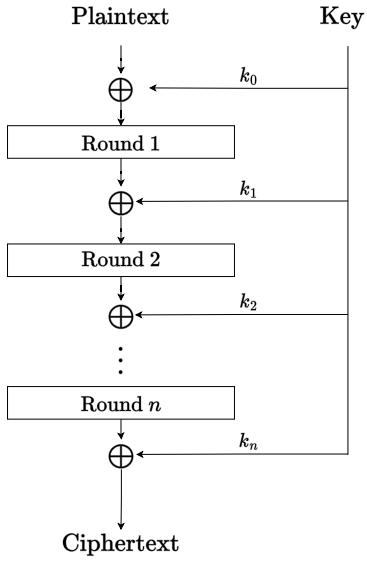

Figure 2.1: Scheme of a key-alternating block cipher.

submitted to the Data Encryption Standard competition and later changed into DES by adding a permutation at the end of each round and reducing the size of the key and the blocks. Lucifer is vulnerable to differential cryptanalysis for half of the keys and an attack could be carried with data complexity 2^{36} [21].

Typical for the Feistel network is that one round only updates half, or part of the input. More precisely, if we split the input into $(X_L, X_R) \in F_{2^n} \times F_{2^n}$, then the output is $(Y_L, Y_R) \in F_{2^n} \times F_{2^n}$, where

$$Y_R = X_L \oplus F(X_R), \quad Y_L = X_R.$$

This is illustrated in Figure 2.2. An advantage of Feistel Networks compared to SPN is that the underlying function does not need to be invertible. This property gives more freedom in the choice and construction of a round function F.

Feistel networks have been extended to Generalized Feistel Networks that are analysed in [64]. Several constructions and their combinations are considered, such as the unbalanced network, alternating, numeric and type 1, 2 and 3 Feistel networks. Unbalanced networks are networks for which X_L and X_R are of different size. Numeric networks use modular addition instead of XOR, while type 1, type 2 and type 3 networks split the input into multiple blocks and

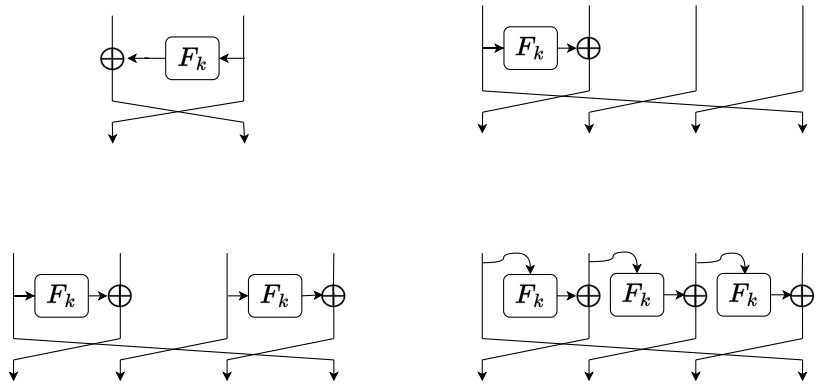

Figure 2.2: Classical Feistel construction and Generalised Feistel network of type 1, 2 and 3.

apply changes to only a few of the blocks, while the rest stays unchanged over one round. Figure 2.2 illustrates a few cases of generalized Feistel networks.

Many widely used algorithms are based on the Feistel construction. The former standard DES has a classical balanced structure. CAST-128 [2] is an ISO standard supported by the PGP 3.0 system for email security and approved for use by the Government of Canada. Twofish [131] is also supported by PGP, as well as the free software KeePass for safe key storage and TrueCrypt for local data encryption. KASUMI [55] is used in the confidentiality and integrity algorithms in 3GPP, in the A5/3 key stream generator and in GPRS in the GEA3 key stream generator. Other algorithms include Piccolo [133], and the ISO standards HIGHT [65] and SEED [118].

2.1.2 SPN

A Substitution Permutation Network (SPN) is a design strategy in which the round function consists of two main layers - a substitution layer and a permutation layer. Usually the Substitution layer divides the n-bit block into smaller m-bit blocks (words) and applies an sbox transformation, also called a look-up table, over the input. Sboxes are usually invertible functions $S : \{0,1\}^m \rightarrow \{0,1\}^l$ which take an m-bit input word and return an l-bit output word. A permutation is also called a linear layer; it consists of liner operations which can be a bit-level permutation or consist of word-level oriented operations. Separated the two layers do not provide enough security, as one

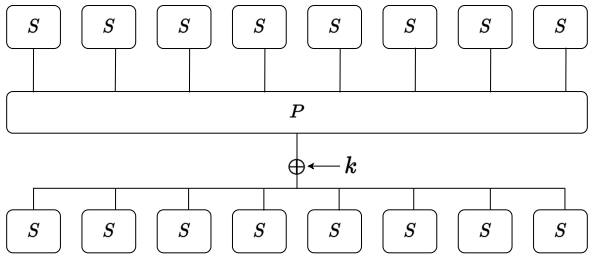

Figure 2.3: SPN structure

can be seen as a substitution cipher and the other as a transposition cipher. However, a well-thought combination of these simple and hardware efficient operations over multiple rounds are considered to perform enough confusion and diffusion to reach a good security level. SPN ciphers are often designed with the wide trail strategy [47] in mind, which provides provable security against major cryptanalysis techniques such as linear and differential cryptanalysis. Ciphers constructed over an SPN are AES [3] and the ISO lightweight standard PRESENT [35].

2.2 Stream ciphers

Contrary to block ciphers which encrypt large blocks of bits, stream ciphers encrypt the message in small units. Usually encrypted bit by bit, these units can also be bytes or words. Stream ciphers imitate the idea of the one-time pad, in which the key is as long as the message and never repeated, however they generally use a small key. Compared to block ciphers, stream ciphers perform with extremely high speed and high throughput, which makes them a desired structure. They are of interest for high speed encryptions needed in satellite communications, network embedded systems, TV channel broadcasting and more. Examples include the SNOW3G [56] cipher as part of the 3GGP used for telecommunications, Salsa20 [24] in TLS, RC4 used in SSL/TLS and A5/1 [39] for mobile communications.

Depending on how the keystream is involved with the input and output bits we can distinguish between synchronous and self-synchronizing stream ciphers. In self-synchronizing stream ciphers each keystream character is derived from a fixed number of preceding ciphertext characters. Such stream ciphers are targeted for channels with low bit error rate but for which bit synchronization can present a problem [103, 107]. If a ciphertext is transmitted with an error or is tampered by a malicious user, only a limited number of bits of the plaintext

will be affected and the cipher will automatically resume to correct decryption quickly, without the need of synchronization flags or additional mechanisms. Although not much attention has been given to self-synchronization stream ciphers, they posses desirable properties. An example of a self-synchronizing stream cipher is the eSTREAM [138] candidate Moustique [46], which was proven insecure soon after the competition [72].

In stream ciphers the keystream is generated independently of the plaintext and is usually produced by a pseudorandom generator parametrized by a secret key. This is the case for most common stream ciphers. They consist of an internal state S of s bits, an initialization function $G : \mathbb{F}_{2^k} \times \mathbb{F}_{2^n} \to \mathbb{F}_{2^s}$, a state update function $F : \mathbb{F}_{2^k} \times \mathbb{F}_{2^s} \to \mathbb{F}_{2^s}$ and an output function $H : \mathbb{F}_{2^k} \times \mathbb{F}_{2^s} \to \mathbb{F}_{2^b}$, where k is the size of the secret key and b the size of the output, also called a keystream word (bit). Then for given input (IV, K) over t clocks of the cipher we have

$$S_0 = G(K, IV),$$
$$z_t = H(K, S_t),$$
$$S_{t+1} = F(K, S_t).$$

Finally, the ciphertext is produced as $c_t = m_t \oplus z_t$.

The design of stream ciphers is either converted from a block cipher, or created from scratch, which makes its analysis more challenging. Some block ciphers can be turned into stream ciphers with the use of appropriate mode of operation, for example CFB, OFB and CTR [121]. Typical ad hoc designs are based on Linear Feedback Shift Registers (LFSRs), Nonlinear feedback shift registers (NFSRs) or table-based constructions. To introduce non-linearity in LFSRs based designs, common techniques are the use of filter function over long LFSR, the combination of several short LFSRs, or irregular clocks, as in the A5/1 cipher [39]. As filtering and combining techniques have been proven vulnerable to fast correlation attacks [104] and algebraic attacks [43], current designs use a Finite State Machine (FSM) to provide non-linearity, as the SNOW [56] and ZUC [57] ciphers. NFSR-based ciphers are Trivium [40] and Grain [63], while table-based ciphers are the RC4 [127] cipher used in protocols such as SSL and TLS, HC-128 [149], and its extension HC-256 [148] from the eSTREAM portfolio.

2.3 Hash functions

Hash functions are keyless algorithms that transform any size message into a fixed length sequence, where the output values cannot be inverted to produce

the original input. The output of a hash function is also referred to as a digest. Their main purpose is to authenticate data without the use of a secret key. Additionally, any change of the input blocks would result in a completely different hash value. As any size input can be digested to a fixed size output, it is more convenient to verify the hash value instead of the whole text. Passwords are usually short, however storing their hash value instead of their encryption protects them from recovering the plain password from the ciphertext due to the one-wayness of hash functions. More examples of applications of hash functions are to generate session IDs for internet applications and data caching, protect sensitive data such as passwords, web analytics, payment details, to add digital signatures to emails, or to verify transactions in the bitcoin protocol.

A secure hash function needs to be *collision resistant*, *pre-image resistant* and *second pre-image resistant*. This means that the problem of finding the original value from the hash value is hard, finding two messages which produce the same hash is hard, and finding a second message that hashes to the same hash as a given one is hard. More precisely, finding x such that $H(x) = y$ for a given y has complexity 2^n, finding x' such that $H(x) = H(x')$ for a given x has complexity 2^n and finding a pair x, x' such that $H(x) = H(x')$ has complexity $2^{n/2}$.

Hash functions are mostly constructed by using a block cipher, like the Skein hash function [59], derived from Threefish. Constructing a hash function from a block cipher is a well investigated approach and all possible constructions have been studied and classified in [122]. Most used constructions for building a hash function from a block cipher are the Davies-Meyer [147] construction used for MD and SHA families of hash functions, and the Merkle-Damgård construction [106] used for the Whirlpool hash function, based on an AES-like cipher [126]. Another approach is to design dedicated functions by constructing the iterated function from scratch.

After the break of the SHA-1 [144] and MD5 [145] hash functions in 2005, where a collision can be found in 15 minutes, a competition for a new standard was started in 2007 by NIST. The candidate Keccak [25] won the SHA-3 competition and became an international standard [114]. The best preimage attack on round-reduced Keccak is from 2021 [95] and uses an algebraic method to efficiently perform an exhaustive search, covering four out of 24 rounds of the algorithm. A widely used hash function is the SHA-2 standard [113], which is not substituted by SHA-3 and is still in use. It is included in most internet protocols such as TLS, IPSec, PGP, SSH, as well as in Bitcoin. Other hash functions are RIPEMD-160 [38], Blake2 [10] and Skein [59].

2.4 MAC algorithms

Message authentication code (MAC) algorithms are used for data authentication over an open communication channel. We cannot prevent the adversary from tampering with the data, but we can guarantee that any such tampering will be detected with high probability by the honest user. The purpose of MAC algorithms is to ensure no third party can change the sent information or impersonate the sender without being detected.

MAC algorithms can be seen as a keyed hash. They take a message m and a secret key k and compute a tag τ , $\tau = MAC(m, k)$. Then the pair (m, τ) is sent to the receiver who verifies if τ is the correct tag corresponding to the message m.

Usually data authentication is a desired property in a combination with secrecy. Classically a MAC algorithm is combined with an encryption algorithm in a MAC-then-encrypt (MtE), Encrypt-then-MAC (EtM) or encrypt-and-MAC (E&M) variations. Algorithms which provide both authentication and encryption simultaneously with a single key are called Authenticated Encryptions (AE) and are described in the next section. MtE generates the tag from the plaintext, appends it to the plaintext and then encrypts both of them. This authentication method is used in the SSL and in an older version of the TLS protocols. EtM first encrypts the message and authenticates the ciphertexts, as it is done in the Internet protocol security (IPsec). E&M computes a tag over the plaintext, encrypts only the plaintext and sends the two together. E&M is implemented in the Secure Shell protocol (SSH). It is shown in [20] that only the EtM variation is generally secure, assuming that the encryption function and the MAC function are secure.

The security goal of MAC algorithms is to provide data authentication, not secrecy. A MAC algorithm is considered secure if an adversary can not provide a valid tag for a message that has not been authenticated yet. This notion can be classified in three cases depending on the strength of the security claim. An *existential tag forgery* is called the creation of *any* pair (m, τ), where τ was not produced by the original sender. The message m could be any random value without particular meaning and is not necessarily under the control of the adversary. A *selective tag forgery* is the production of a valid pair (m, τ), where the message m is chosen by the adversary prior to the attack. Finally, the strongest forgery attack is a *universal forgery*. The adversary is said to conduct a universal tag forgery, if he can compute the tag for any message, including chosen, random or provided by someone else.

Widely used MAC algorithms are CMAC [117] HMAC [141], used in Internet protocols such as SSL, IPsec and SSH, and the recently standardised Chaskey

[110].

2.5 AEAD

Authenticated Encryption (AE) is one of the most common primitives used for protecting data, because it provides both confidentiality and authenticity simultaneously. The separate use of an encryption algorithm and MAC algorithm has proven to be prone to implementation mistakes, as it can be seen from many practical attacks through the years. Additionally, the use of two algorithms implies the need of two secret keys. That motivated the design of algorithms using a single algorithm to perform both encryption and authentication, referred also as dedicated AE. Often there is additional information together with the encrypted message, for example a header, that also needs to be authenticated. To authenticate the data and the additional (associated) data we use Authenticated Encryption with Associated Data (AEAD) schemes [128].

In 2013 the CAESER competition for Authenticated encryption schemes was initiated. The final portfolio with six schemes, some of which Ascon [54], AEGIS [151] and COLM [5] was announced in 2018. With the rise of the Internet of Things (IoT) and the need for lightweight algorithms for constrained devices, NIST announced a call to standardize lightweight Authenticated Encryption schemes in the same year [115]. The ten finalists were announced in 2021 and include Ascon, Elephant [26], PHOTON-Beetle [14], TinyJAMBU [150] and Sparkle [17].

Dedicated AEAD are designed in various ways and use for building blocks block ciphers, stream ciphers, permutations. The sponge construction is often the choice of the designers. A recent paper from 2022 [69] gives a detailed summary of Authenticated Encryption schemes and their design, security and performance properties.

2.6 ARX

ARX stands for Addition, Rotation and XOR. Algorithms designed with only these three operations are referred to as ARX primitives and are a popular choice. The cyclic rotation and modular addition can be implemented very efficiently in hardware and software, bringing high speed and high throughput to their performance. Another advantage compared to other design strategies is that the computation of sboxes through a look-up table is avoided. It is proven that any sbox can be represented by a sequence of only addition, rotation and

XOR operations. Two design strategies for ARX primitives, which are provably secure are proposed in [53]. One is the long trial strategy, inspired by the wide trial strategy for block ciphers, which uses small sboxes composed of the three operations. The proposed SPARX family of ciphers adopts this strategy and remains currently unbroken. The second strategy aims at minimizing the differential probability by exploiting modular addition properties.

The Addition-Rotation-XOR strategy is used to design several primitives. Efficient stream ciphers following the ARX design are Salsa20 [24] and ChaCha [23]. ARX block ciphers are Feal [108], SPARX [53], SPECK [16], TEA [146], XTEA [116] and Hight [65]. The MAC algorithm Chaskey [110] and the hash functions such as the SHA-3 finalists Blake [9] and Skein [59] are also ARX primitives.

ARX structures that additionally add a constant, denoted with ARX-C, are proven to be complete and any function can be described in this setting. That makes the analysis and understanding of ARX-C ciphers of great importance. Another extension of ARX is the replacement of the modular addition with the bitwise AND operation. This is of interest because it simplifies the cryptanalysis and improves the hardware efficiency. Examples are the SHA-3 hash function Keccak [25], the NIST block cipher Simon [16] and the AEAD scheme Sparkle [17].

Even though not fully understood and still analysed, because of the little design strategies for proven security, ARX ciphers are considered to provide significant security. The algebraic degree is usually high only after few rounds since the carry of the modular addition almost reaches its maximal degree. Structural attacks over ARX primitives such as integral [81] and invariant [140] attacks are less likely. With regards to linear and differential cryptanalysis, the security mostly depends on the modular additions, since it is the only non-linear operation. Properties of a single modular additions are well studied and there exist efficient algorithms to compute the probabilities [143, 93]. However, even if over small number of rounds the differential, respectively the linear correlation are high, they quickly decrease over many rounds. Additionally, there is no efficient way to look for differentials over more rounds and the only way to asses the differential probability is to look for the optimal differential. This is usually done with the help of various automated tools. Differential-linear cryptanalysis seems to be the strongest attack against ARX primitives, as argued in [19]. Rotational cryptanalysis can be efficiently prevented by the addition of constants, although [8] shows an efficient way to analyse the presence of constants in ARX ciphers.

2.7 PUFs

Physically Unclonable Functions (PUFs) are a relatively new idea proposed in 2002 [60] as the hardware solution for secure low-cost key generation. To perform cryptographic functionalities, devices need to posses a secret key. Securely storing a secret key in an Integrated Circuit (IC) is expensive. Another option is to simply hard-code it, however this solution is vulnerable to physical attacks such as invasive, semi-invasive, or side-channel attacks, as well as software attacks such as API attacks and malware. PUFs offer alternative solution by generating unpredictable bits by exploiting the randomness introduced by the manufacturing process, also called process variation. As no two devices can be manufactured with precisely the same metrics, the uniqueness of Integrated Circuits (ICs) is also compared to the human biometrics.

Applications of PUFs are mainly for key generation [50] and device authentication [51], where the authentication between two devices is achieved by a Challenge response protocol (CRP). The literature considers *weak* and *strong* PUFs, where the difference between them is not in the provided security, but rather at the size of the challenge domain. Recent designs such as [137, 96], also called cascaded PUFs, adopt block cipher techniques. However, as many of the previously proposed designs were vulnerable to machine learning (ML) attacks, the security analysis of new designs is mostly based on resistance against ML techniques, which proves to be insufficient. Additionally, the designs overuse the available randomness in a sub optimal way.

Although Physically Unclonable Functions are not cryptographic primitives by themselves, we include them in this chapter because recent designs adopt symmetric-key block cipher design techniques. We investigate the security of some cascaded PUF algorithms with clasical cryptanalysis techniques.

Chapter 3

Lightweight Cryptography

Resource constrained devices such as IoT devices, medical devices, RFID tags, bank cards, etc., are rapidly developing and introduce new requirements for implementations. Unfortunately, algorithms such as AES which is the best algorithm for encryption of large amount of data over the Internet or on a local machine, is not performing optimally for platforms with extremely limited resources. Constrained devices require algorithms which are fast, small, or consume very little energy. The so-called *lightweight cryptography* is a branch of cryptography dedicated to lightweight algorithms for recourse constrained devices. It is in principle not different from classical cryptography as the same principles for security, integrity and confidentiality hold, however it is focused on the performance. Depending on the platform, common restrictions implied on lightweight algorithms in hardware implementations are for *low latency* or *speed*, *area*, *throughput*, *power* and *energy consumption*. For software implementations the important metrics for optimization are *execution time*, *run-time memory* (RAM) and *code size* (flash memory). There are no strict performance requirements to classify which ciphers fall into the lightweight category and often an AES implementation is used as a benchmark. Unfortunately, as good as it sounds, a cipher can not be optimized towards all properties mentioned. This chapter explains the most common requirements and the trade-offs between them, both from hardware and designers point of view.

3.1 Latency

Latency refers to the time needed for the execution of a given operation. In cryptography this is usually the time occurring between the moment an input block is injected and the corresponding output block is obtained. Many applications need a fast response to perform properly and the time for this response is the latency of the algorithm. Latency can be calculated in number of clock cycles or seconds, both in hardware and software.

Applications which are highly sensitive to delay and have real-time security needs are live-streamed videos, online gaming, video conversations, storage devices, IoTs devices, autonomous vehicles and many more. An algorithm known to have very low latency is Prince. One of the fist papers analysing the latency and comparing the speed of several algorithms is [76].

3.2 Throughput

Throughput is a metric associated with how much data can be transferred in a given time frame. In cryptography, throughput measures how many encryptions can be performed per unit of time, usually measured per second. Applications processing large volumes of data such as data going into storage or going out of storage, where each data unit needs to be encrypted can benefit from high throughput algorithms. Other applications needing the simultaneous processing of large operations, both data in motion and static, is video transmission, data updated from IoT devices, databases, etc.

3.3 Area

In hardware implementation, the exact chip area is dependent on the technology used for fabricating the chip and the standard cell library [129]. In order to compare implementations independently, the accepted metric is *Gate Equivalent* (GE). GE corresponds to the physical area of a single NAND gate with two inputs. Each operation or logical gate corresponds to some GE value. When we talk about the area of a cipher, we count the number of corresponding GEs needed for its implementation. For example, an implementation of Prince on different libraries has between 1000 and 1160 GE.

Area is shown to be directly related to power. Generally, the more area is taken on the circuit, the more power is needed to run the cryptographic algorithm.

Therefore, devices which require less power consumption are usually optimized for minimum area. The smallest AES implementation with 1600 GE and PRESENT implementation with 1065 GE are proposed in [67]. However, these implementations come with the cost of many clock cycles and therefore large latency.

In software area can mean different things. It is represented by the necessary run-time memory (RAM) and the code size (flash-memory).

3.4 Energy

Energy refers to the total electricity consumed by the circuit to perform an iteration. For cryptographic purposes an iteration is usually taken to be one encryption. Energy is a resource requirement needed specifically for battery powered devices, such as implanted medical devices. The role of energy in cryptographic primitives is investigated in [119].

3.5 Power

Power is measured usually in milliWatts mW and can be seen as a performance characteristic, depending on the transistors and circuit design. It is closely related to energy as power is defined as the energy consumed over certain time unit, however they represent completely different implementation properties. In practice, low power does not necessarily mean low energy consumption.

3.6 Trade-offs

Unfortunately, as good as it sounds, a cipher can not posses all mentioned properties. Often area and power are closely related and better throughput often means more area. The concrete measurements of area, latency, throughput depend first on the design of the algorithm and second, on the implementation architecture. In this section we look into different architectures and how they influence the trade-offs, as well as the design strategies which make an algorithm "lightweight".

3.6.1 Trade-offs in hardware architectures

Several implementation strategies are used for achieving different optimization goals. The most common strategies are *round-based* implementations, *loop unrolling*, *pipelining* techniques, *parallel* architecture and *serialization*. It is often necessary to explore more architectures to achieve the most appropriate trade-off and performance.

Round-based architectures are a direct implementation approach and perform one round of the algorithm per clock cycle, computing the output bits of the round in parallel. Those implementations are typically quite balanced in terms of area, latency and throughput. Round-based implementations are considered as a reference point of comparison with other architectures.

When more rounds are performed at once the architecture is referred to as loop unrolling. A factor of N indicates the number of loops executed in one clock cycle. For round-based implementation, this number is one. Executing more loops in a single clock cycle means lower latency, but simultaneously this implies more area.

Parallel implementations allow the encryption of multiple blocks simultaneously. This highly improves the throughput, as more data is encrypted at the same time. This however does not change the time needed for one encryption, hence the latency rests unchanged. In terms of area, the implementation of two encryptions means double the area, so it increases proportionally with the throughput. Parallelized architectures show a clear trade-off between throughput and area.

Pipelining is a technique where multiple instructions overlap during implementation. The pipeline reads one instruction while previous instructions are being executed in other segments of the pipeline. The pipeline will be the most efficient if the instruction cycle is divided into segments of equal duration. This technique has a higher clock frequency and improves the throughput at the cost of area and latency.

In a serial implementation only a small part of the state is updated in one clock cycle, typically computing only one or a few sboxes at once. The whole output is then updated in several clocks. This technique performs at high clock frequency. The advantage of serialization is the small circuit size.

A comparison between AES implementations in different architectures can be seen in [15], together with comparisons to several lightweight block ciphers. It is shown that the smallest AES implementation is also by far the most power efficient, however it has the largest energy consumption due to the high number of cycles required. The presented parallel AES implementations differ in the way

the sbox is implemented and this is significantly reflected in the area, latency and energy.

3.6.2 Trade-offs from a designer's point of view

Lightweight algorithms follow the same design structures and examples of all can be found. What makes a cipher lightweight in those structures depends on the choice of "light" rounds and the number of rounds. For SPN structures optimizations can be done both in the permutation and substitution layers. The permutation layer is usually implemented as a matrix multiplication. The choice of a matrix with small coefficients or coefficients with low Hamming weight might lead to better performance. In particular, the MDS matrices are known to provide optimal diffusion, however they are more expensive to implement. Additionally, for the substitution layer an important metric is the size of the sbox. While in classical ciphers 8-bit or 16-bit sboxes are preferred, for lightweight algorithms the usual choice is 4-bit or 6-bit sboxes. Note, that small sbox size might require more rounds in order to achieve the same security level. Lightweight ciphers following the Feistel structure optimize the round function F. Decisions that contribute to the optimization are small sbox size and operations that are "cheap" in hardware, such as the AND gate. Algorithms following the ARX or AND-RX structure are in general considered lightweight for software as all involved operations are efficiently implemented. Modular addition particularly is extremely cheap in software and its nonlinearity highly contributes to the security. Additional optimizations can be done with the choice of word length and rotational amount. A detailed discussion on the design choices for lightweight primitives is presented in [32].

Lightweight ciphers are often designed with specific application in mind. Ciphers, dedicated to low-latency requirements aim for security achieved over the smallest number of rounds with the cost of more area. An example is the Prince cipher, which has strong substitution and linear layers which achieve the desired security level in 12 rounds. Prince has very low latency, but large area due to the large sboxes. Contrary, if an algorithm is designed for low area applications, then the strategy would be to use small sboxes and easy linear operations at the cost of more rounds. An example is the low area cipher PRESENT. Ciphers aiming for low power also optimize for area. Throughput on the other hand can not be achieved at design level. This metric follows from the chosen hardware implementation, where parallel architectures score the best in maximizing the throughput. Optimizing for energy depends on the power and latency of the cipher. Low power does not always mean low energy and vice versa, therefore this metric is more difficult to achieve.

3.7 How to decide

As AES is considered the best cipher in terms of security, a substantial effort has been channeled towards implementing it in a way suitable for constrained resources, both in hardware and software. For software implementations the Advanced Encryption Standard New Instructions (AES-NI) implementation was integrated in Intel processors since 2010 and performs significantly better than AES. It is currently implemented also in AMD and ARM processors. AES-NI performs particularly good for parallelizable modes such as CBC-decrypt and CTR with speed-up up to ten times. Ciphers performing with small ROM space over less CPU time are SIMON and Piccolo. For hardware the AES implementations are still not optimal due to the heavy round operations and other algorithms are often preferred for extremely restricted resources.

The best choice of a cipher and its implementation is very application dependent. The Internet of Things (IoT) provides a major diversity of connected devices which are part of our daily lives and most have different requirements. In terms of energy supply we can distinguish between *passive* and *active* devices. Passive devices do not have their own power resource and are powered by a wireless or physical contact, while active ones are powered by their own supply, like a battery. Usually requirements towards active devices are more relaxed on the power, however highly optimized for energy and overall execution time.

For example, a passive device such as a contactless bank card benefits from an extremely low power algorithm. When in use it is powered by a reader device and is hence supported by a continuous small amount of power. An appropriate algorithm would be for instance one with a very light and simple round function, but over many rounds in order to reach the security goals. One example of such a cipher is PRESENT. Other ciphers performing with low power consumption are SIMON, SPECK and Piccolo. These ciphers are also very efficient in terms of area. Power efficient algorithms are not necessarily appropriate for active devices such as a pacemaker, which is extremely energy restricted. The overall energy consumption of low power operations, however, over longer time can be greater than heavier round function consuming more power, but over a small number of rounds. The battery of a pacemaker should last at least 10 years before being replaced. An example of energy efficient cipher is Midori [13]. Ciphers with efficient implementations toward low latency are Prince, Mantis [18] and Qarma [11]. The most throughput efficient architecture is the parallel architecture. An example of a cipher that can be implemented with high throughput is Midori. A detailed comparison between the hardware performance of several algorithms is given in [44, 52].

Chapter 4

Symmetric-key cryptanalysis

The design of cryptographic primitives is a complex process including developing the algorithmic steps, defining the adversarial model, providing security claims and proofs, and considering the respective use cases and implementation. As discussed before, the purpose of cryptanalysis is to build a deeper understanding of the security of the algorithms and build more trust in them and their claimed security level, or correct them. In this chapter we focus on the adversarial models and discuss various cryptanalysis techniques, attack models and attack complexities.

4.1 Adversarial models

An adversary or an attacker is a third party with malicious intentions trying to reveal secret information such as the key or the encrypted message. In practice the adversary may have more power than simply observing the ciphertexts. For example, an adversary can have access to the encryption oracle and request the encryption of certain messages. Based on the access of the adversary to the plaintext and ciphertext, we can classify the following adversarial models:

Ciphertext-only attack. In this setting the adversary is completely passive and has information only about a number of ciphertexts. A good cipher should not leak any information about the plaintext and the ciphertext should appear as a random sequence of bits.

Known-plaintext attack. The adversary knows pairs of plaintexts and their corresponding ciphertexts. Examples are the attacks on the Enigma during WW2, when phrases commonly appearing in the conversations like the weather forecast or geographical names were targeted.

Chosen-plaintext attack. The adversary has access to the encryption oracle and is able to choose any text to be encrypted. An *adaptive chosen-plaintext* allows to choose the next block based on information derived from the already encrypted blocks.

Chosen-ciphertext attack. The adversary has the power to ask for the decryption of a ciphertext of his liking. Such attacks are usually performed on systems with public-key encryption. An *adaptive chosen-ciphertext* allows the choice of new ciphertexts for decryption to be based on the previously obtained data.

Primitives can be additionally analysed in several key models. Some platforms assume that the key is known or under the control of the adversary and hence can be chosen or influenced in certain way. Such are the *related-key*, *weak-key* and the *key-less* settings.

Related keys are keys with any mathematical relation between them. It could be a straight forward relation like having certain bits fixed to the same values, or more complex ones. Exploiting the key schedule could lead to a collision in the round keys, meaning that distinct, however related with some property master keys could lead to the generation of equal round keys.

One of the first papers considering related keys is [27], in which Biham shows that some key schedules produce round keys with obvious relations between them. He further proposes a chosen plaintext attack to recover all round keys by only choosing the relation between the master keys. Independently in the same year Knudsen proposes in [77] pairs of related keys such that they produce 13 equal subkeys for DES. Then the encryption of related plaintexts leads to related ciphertexts under the related keys with some probability.

The significance of investigating the related-key setting can be seen for example in the work of [136]. The WEP protocol (Wired Equivalent Privacy) with underlying RC4 stream cipher is fully broken under the related-key adversarial model due to the easy way of exploiting key-IV pairs which are related in a simple way.Another use case of related keys is when a block cipher is used as a hash functions or in a multi-user scenario, where a relation between the keys

used by different users can be exploited. AES-256 is shown to be vulnerable in the related-key setting in [31]. The authors propose a distinguisher over the full rounds of AES-256 with data complexity 2^{120} which can be continued as a key recovery attack or to construct pseudo-collisions (related collisions between the outputs of two related inputs over related keys) when the algorithm is used as a hash function in the Davies-Meyer mode.

Weak-keys refers to a set of keys that satisfy some property, which causes undesirable effect over the ciphertext, or causes a collision in some intermediate states of the cipher. For example, DES has weak keys for which the plaintext and the ciphertext functions are the same, leading to 2^{32} fixed points for each of these keys. Additionally, there is a set of semi-weak pairs, for which the encryption with one is the same as the decryption with the other and vice versa [109]

The key-less model includes the known key and chosen key scenarios, where the adversary has knowledge of the key. The aim is to determine a plaintext such that it is equal to the ciphertext or to distinguish the used cipher when the key is known. This model is applicable when a cipher is used for a hash function. An attack over a key-less model is generally hard to define, however, key-less distinguishers are considered useful for cryptanalysists and often analysis of such models can lead to an improvement to other settings. For example, techniques similar to the ones described in [80] for a known-key distinguisher are used for the rebound attack of Whirlpool [90]. Later, the authors of [91] present a meaningful application of the key-less model and apply it to the block cipher Present. The possibility of distinguishing any block cipher in the key-less model in the case a convenient linear relation is present for large set of keys is introduced.

The related key and weak-key models are considered in the cryptanalysis of Chaskey and SpoC [84, 86]. In [84] the relation between the keys is chosen so that the rotational property between the plaintexts holds after the key addition. In [86] we additionally consider related nonce together with the related key scenario.

4.2 Security claims

The straightforward approach to recover the secret information, or to *break* a cipher is to try all possible values of the secret key and is called a *brute force attack*. This is however infeasible for large keys and requires 2^{n_k} encryptions

with n_k the size of the key. What is considered as breaking a cipher in modern cryptography is finding an attack that can recover key bits, distinguish the cipher from a random permutation, or give other useful information/properties with less effort than for a brute force attack.

The effort required to perform an attack is also called the *complexity* of the attack. The actual time needed for the completion of an attack depends on factors like the algorithm itself and the used machine and implementation. Cryptanalysis measures the complexity in a way that abstracts real time and instead focuses on a countable metric like number of encryptions. The usual considered complexities are *data*, *time* and *memory* complexity. Data complexity is the number of encryptions and decryptions of the algorithm required to execute the attack. Time complexity measures the extra time needed due to additional or offline computations, where the time needed for memory access is often ignored. Memory is indicated in the cases some information has to be stored in order to complete the attack. If not specified otherwise, time and memory complexities are considered to be only what is required for an encryption and are the same as the data complexity. Depending on the complexity, attacks are discussed as practical and theoretical attacks. Practical attacks imply that the required time and resources to perform the attack are reasonable and can be experimentally run. Theoretical attacks have high complexity and can not be empirically verified.

Due to the enormous research efforts dedicated to the design and cryptanalysis of symmetric-key primitives, nowadays breaking a cipher practically is a difficult task. Instead simplified versions of the algorithms are analysed with the goal of understanding better their actual security and vulnerabilities.

The strongest attack on block ciphers is to recover the secret key. However, there are scenarios when recovering less is enough. Important first step is to distinguish the received ciphertext bits from a random sequence of bits. Following a certain property throughout the cipher could result in a relation between some of the output bits, which holds with certain probability. This relation can be exploited to distinguish that a specific algorithm was used to encrypt the collected data. Such an attack is called a *distinguishing attack*.

4.3 Cryptanalysis techniques

There are various techniques that can be performed in order to reveal secret information. The main types of attacks are the mathematical, or also called algorithmic attacks, and the physical attacks. Examples of physical attacks are

side channel [70], timing attacks [82] and fault attacks [22]. In this book we will discuss only algorithmic techniques.

Generic attacks are such that can be performed over a structure regardless of the details of the algorithm or how the key is inserted to the algorithm. For example, brute force is a generic attack that can always be performed no matter how the key is used. If we get a reasonable plaintext for some key, we assume this key is correct or is a possible key. Other generic attacks are the birthday paradox, collision and preimage attacks on hash functions and the Time-Memory Trade-Off attacks (TMTO).

Statistical attacks follow a property between the input and output bits that happens with a given probability. Both highly probabilistic and impossible events can be exploited to gain additional information on the plaintext. Collecting enough ciphertexts would eventually lead to a pair for which this property is satisfied. Statistical attacks include differential cryptanalysis, linear cryptanalysis and rotational cryptanalysis.

4.3.1 Differential cryptanalysis

Differential cryptanalysis is one of the most powerful cryptanalysis techniques against symmetric primitives. It was introduced by Biham and Shamir in 1990 and is the first attack that fully breaks the Data Encryption Standard (DES) - the most used block cipher at that time [30]. This technique is a chosen-plaintext attack that follows two inputs differing usually only in a small number of bits throughout the whole algorithm. Often of more interest are input differences resulting in certain output differences with probability significantly more than $\frac{1}{2^n}$, however small probabilities or even transitions with probability zero can also be exploited in order to reveal partial information about the secret key [28, 139].

Let X and X' be two plaintexts. The difference between two plaintexts, or two ciphertexts, is defined in [89] as $\Delta X = X \otimes X^{'-1}$, where $\otimes$ denotes a specified group operation on the set of plaintexts (respectively, set of ciphertexts) and $X^{'-1}$ denotes the inverse of the element $X^{'}$ in the group. Usually the XOR operation is used when performing differential cryptanalysis, due to the common integration of the key material to the state with XOR, however other operations such as modular addition are also used [94, 42]. In this thesis we consider differences with respect to XOR, with the exception of Section 4.3.2, where a case of modular addition is considered. When analysing a single round of the algorithm, the XOR of two inputs is invariant to the XORing of the key and linear to the permutation and some other operations which makes it easy to follow the knowledge of such difference in the inputs over the round operations.

The output difference of an sbox cannot be known exactly, however it follows some statistical distribution and can be predicted with some probability.

Let us consider an r-round iterated cipher and assume we can observe the intermediate differences before and after each round for a pair of plaintexts. A *differential characteristic* over r rounds of an iterated cipher is defined as the sequence of intermediate differences $(a = a_1, a_2 \dots a_r = b)$, where each (a_i, a_{i+1}) represents the input and output difference of one round. The probability of a characteristic over linear operations is either zero or one and can be easily determined. When non-linear operations are involved the probability has to be evaluated. For sboxes this is achieved with the help of the Difference Distribution Table (DDT) of the sbox. The DDT consists of the number of times an input difference a goes to an output difference b and is calculated over all possible input values X. Then, assuming that the cipher is Markov and the round keys are independent [89], the probability of a characteristic is computed by multiplying the probabilities of all single rounds.

$$Pr(a_1, a_2, \dots a_r) = \prod_{i=2}^{r} Pr(a_{i-1}, a_i).$$

In practice the keys are derived from a key schedule and are obviously dependent, however setting such assumptions makes the analysis feasible and gives relatively good approximations. Many works investigating the gap between theoretical and practical results are published, for example [1, 6, 100].

Practically only the input and output differences of the algorithm can be observed and the intermediate differences are not of importance. We call a *differential* the set of all characteristics with input difference a that yields an output difference b and denote it by (a, b). The probability of the differential (a, b) is calculated as the sum of the probabilities of all characteristics having a as input and b as output difference.

$$Pr(a, b) = \sum_{a_2, \dots a_{r-1}} Pr(a_1 = a, \dots, a_r = b).$$

Computing this probability is however a hard problem for differentials over multiple rounds as the computational complexity is too high.

Once an r-round characteristic is constructed for an $r + 1$-round cipher with satisfying probability, the adversary chooses N pairs of plaintexts with the decided input difference and collects their corresponding ciphertexts. Recovering the last round key requires a partial decryption of the ciphertext over the last round. For each possible round key value a counter is set and is incremented

if the input difference to the last round corresponds to the expected output difference of the characteristic. This step is used to filter the possible keys and the value with the highest counter is considered as the correct key, or if several such occur, as a candidate key.

Our works [86, 7] use differential techniques to analyse the sponge construction SpoC and the block cipher K-Cipher. We construct differential characteristics in [86] and exploit them to perform a tag forgery attack and a message recovery attack for SpoC. In [7] differential characteristics are used as a filtering step to recover secret material in the form of key bits and random unknown values. The non-linear layer of K-Cipher uses a different sbox depending on a secret random value added to the layer. This randomness is supposed to restrict the power of differential cryptanalysis. However, the structure of the cipher allows for the construction of a generalised difference distribution table that combines all possible DDTs. Therefore, the constructed characteristics over the Generalised DDT hold for any possible sbox.

Challenges in differential cryptanalysis. Computing the probability of a differential is a hard problem since the computation of an exact probability assumes we know all intermediate characteristics and their corresponding probabilities. Instead, the characteristic with the highest probability, called an optimal characteristic is often used as an approximation of the differential as it is believed it dominates the differential. However, this is not true in general and leads to imprecise estimates of the probability in many cases. A recent paper from 2018 [6] investigates the gap between the probabilities of the most probable characteristic and the differential that contains it. They use an automated tool to enumerate a large set of characteristics contributing to the probability of a differential and thus better approximate its probability. This approach is applied to multiple block ciphers. One result is that, for example, an 8 round version of Skinny-64 has an optimal characteristic with probability 2^{-72}, while the differential containing it has probability of $2^{-56.93}$.

Automated search. Automated tools such as the SAT (Boolean Satisfiability), SMT (Satisfiability Modulo Theories) and MILP (Mixed Integer Linear Programming) based solvers offer an efficient search for optimal characteristics. They mostly target ARX ciphers as the rotation, XOR and modular addition can be easily represented as Boolean expressions. A disadvantage of most developed tools is that they are either focused on specific cryptographic algorithm(s) or offer limited functionalities [33, 98, 111, 130, 134]. The recently published tool CASCADA [124] covers large functionalities such as finding differential, related-key differential, impossible-differential, related-key

impossible-differential, rotational-XOR (RX), impossible-RX, linear, and zero-correlation distinguishers. Additionally, it is easy to modify or add additional ciphers. We have used an early version of CASCADA, the ARXpy tool [123] in our analysis of SpoC in order to look for the optimal differential characteristic of the underlying cipher Simeck.

Related Techniques. Differential cryptanalysis has been largely analysed and further developed, leading to a numerous variants like Truncated differentials [78], the Boomerang attack [142], Rectangle attack [29] and Impossible differentials [28].

One of the first modifications of the differential attack is the truncated differentials [78]. If (a, b) is an r-round differential, a' is a subsequence of a and b' is a subsequence of b, then (a', b') is called an r-round truncated differential. It shows that only a part of the difference can be predicted and used to retrieve information on the key.

Higher order differentials make use of the derivatives of cryptographic functions. For $(S, +)$ and $(T, +)$ being Abelian groups and a function $f : S \to T$, the derivative of f at point $a \in S$ is $\Delta_a f(x) = f(x + a) - f(x)$, so the rth derivative of f at the point $a_1, \ldots, a_r \in S$ is $\Delta^{(r)}_{a_1,\ldots,a_r} f(x) = \Delta_{a_r}(\Delta_{a_1,\ldots,a_{r-1}} f(x))$. Note that the first order derivative corresponds to characteristics and differentials. A subset of second order differential attacks is formed by the boomerang attack. In [142] Wagner introduced the boomerang attack showing that one not always needs a significant probability to perform a differential attack. Instead, the cipher is split into two parts and differentials over each part are separately attacked. A more detailed description goes as follows: Let us take four plaintexts P, Q, P', Q' and their corresponding ciphertexts C, C', D, D'. The encryption algorithm is decomposed to $E_1 \cdot E_0$. Let us cover the pair P, P' with a characteristic for E_0 and the pairs P, Q and P', Q' with a characteristic for E_1^{-1}. Then the pair Q, Q' is perfectly set up to use the characteristic for E_0^{-1}. This attack was improved into rectangle differentials [29], where the improvement is in the way of choosing the four plaintexts.

In impossible differentials we can take advantage of events that can never happen. A differential predicts that particular differences cannot occur and therefore if a particular guess for (a part of) the key would lead to the conclusion that such a difference has occurred then it is for sure a wrong key. One of the first papers that uses impossible differentials is [28] for an attack on Skipjak. Improbable Differential [37] analyses differentials which are less probable when the correct key is used rather than a random one. Such can be an extension for impossible or truncated differentials.

4.3.2 Differences over a modular addition

Modular addition is a commonly used operation in several cryptographic primitives and its use is not limited to ARX algorithms. Differential and linear properties over modular addition have been extensively studied, for example in [93]. An interesting question is to investigate if we can define some restrictions on the key bits when the key is added with modular addition, assuming that we already know the difference before and after the modular addition. Such analysis can be applied to key-recovery attacks. In the cases when there is no secret key, or the value is known the theory can be applied to the analysis of hash functions and other primitives.

This section presents a research direction that we started. It was motivated by the idea of phantom differentials described in [100]. The goal of this research is to investigate a phantom characteristic in the context of ARX ciphers and to analyse various common situations involving a modular addition and a key. We start with the analysis of a simple modular addition and then we complicate it by adding more variables. We derive a system of constraints that needs to be satisfied in order to have a valid key for a given differential. Here we look into the equations $(x+y)\oplus(x+(y\oplus\beta)=\gamma$, $((x\oplus k)+y)\oplus((x\oplus\alpha\oplus k)+(y\oplus\beta))=\gamma$ and $(x+k+y)+((x\oplus\alpha)+k+(y\oplus\beta))=\gamma$.

Similar models are considered in [120], where the *differential equations addition* (DEA) $(x+y)\oplus(x+(y\oplus\beta))=\gamma$ and $(x+y)\oplus((x\oplus\alpha)+(y\oplus\beta))=\gamma$ are analysed. In the adaptive query model the search space of the secret (x,y) can be minimized with queries (α,β) with minimum number of queries $(n-t-1)$ for the first equation, where t is the position of the least significant '1' of x, and three queries for the second equation when the size is $n>2$. We analyse them from a different angle.

First, let us repeat the well-known formula for modular addition [41]. For given $z=x+y$, where $x=(x_0,x_1,\ldots,x_n)$ and $y=(y_0,y_1,\ldots,y_n)$, $x,y,z\in\mathbf{F}_{2^n}$ we have that

$$z_i = x_i \oplus y_i \oplus x_{i-1}y_{i-1} \oplus \sum_{j=0}^{i-2} x_i y_j \prod_{l=j+1}^{i-1} (x_l \oplus y_l), \quad i = 1,\ldots n-1, \tag{4.1}$$

$$z_0 = x_0 \oplus y_0.$$

Let $R(x,y)_i$ denotes the carry. Then

$$R(x,y)_i = x_i y_i \oplus \sum_{j=0}^{i-1} x_j y_j \prod_{l=j+1}^{i} (x_l \oplus y_l), \quad i = 0, \dots n-1. \tag{4.2}$$

Recursively, we can write the addition and the carry as:

$$z_i = x_i + y_i + R(x,y)_i, \quad i = 0, \dots n-1,$$

$$R(x,y)_i = x_i y_i \oplus (x_i y_i) R(x,y)_{i-1}, \quad i = 1, \dots n-1,$$

$$R(x,y)_0 = 0.$$

Let us have a minimalistic situation $x + y = z$ and let us know the difference α at the input x and the difference γ at the output z. If we additionally know a value x for which this difference holds, we can express the bits of y in terms of x, α and γ.

The rest of this section follows unfinished research ideas and results developed by us.

Lemma 4.1. *Assume that $x + y = z$, $x, y, z \in \mathbf{F}_2^n$. The values of y which satisfy the equations $x + y = z$ and $(x \oplus \alpha) + y = z \oplus \gamma$ for input and output differences α, γ must also satisfy the following restrictions:*

$$\alpha_0 = \gamma_0,$$

$$y_{i-1}(\alpha_{i-1} \oplus \sum_{j=0}^{i-2} x_j y_j \prod_{l=j+1}^{i-2} (x_l + y_l) \oplus \sum_{j=0}^{i-2} x'_j y_j \prod_{l=j+1}^{i-2} (x'_l + y_l)) =$$

$$= \alpha_i \oplus \gamma_i \oplus x_{i-1} \sum_{j=0}^{i-2} x_j y_j \prod_{l=j+1}^{i-2} (x_l + y_l) \oplus x'_{i-1} \sum_{j=0}^{i-2} x'_j y_j \prod_{l=j+1}^{i-2} (x'_l + y_l),$$

where $i = 1 \dots n-1$ and for simplicity $x' = x \oplus \alpha$.

Proof. This can be easily derived from Eqn. 4.1. Let us apply Eqn. 4.1 for z_0 and $z_0 \oplus \gamma_0$. Then :

$$\begin{cases} z_0 & = & x_0 \oplus y_0 \\ z_0 \oplus \gamma_0 & = & x_0 \oplus \alpha_0 \oplus y_0 \end{cases},$$

$\Rightarrow \gamma_0 = \alpha_0$.

Consequently,

$$\begin{cases} z_1 & = & x_1 \oplus y_1 \oplus x_0 y_0 \\ z_1 \oplus \gamma_1 & = & x_1 \oplus \alpha_1 \oplus y_1 \oplus (x_0 \oplus \alpha_0) y_0 \end{cases},$$

$\Rightarrow y_0 \alpha_0 = \alpha_1 \oplus \gamma_1$.

$$\begin{cases} z_2 & = & x_2 \oplus y_2 \oplus x_1 y_1 \oplus x_0 y_0 (x_1 \oplus y_1) \\ z_2 \oplus \gamma_2 & = & x_2 \oplus \alpha_2 \oplus k_2 \oplus (x_1 \oplus \alpha_1) y_1 \oplus (x_0 \oplus \alpha_0) y_0 (x_1 \oplus \alpha_0 \oplus y_1) \end{cases},$$

$\Rightarrow y_1(\alpha_1 + x_0 y_0 + x_0' y_0) = \alpha_2 \oplus \gamma_2 + x_0 y_0 x_1 + x_0' y_0 x_1'$.

Repeating this process for z_n proves the lemma. □

Example: Take $\alpha = 0100$ and $\gamma = 0100$. Then for $x = 1100$ we have the following equations for the bits of y:

$$\begin{cases} \alpha_0 & = & \gamma_0 \\ y_0 \cdot 0 & = & 0 \\ y_1 \cdot 0 & = & 0 \\ y_2 & = & 0 \end{cases}.$$

The described restrictions reduce the possible values of y. If we consider y to be the secret key, then we get restrictions on the key bits and thus reduce the complexity for key-recovery attacks. Alternatively, we can express the equations in terms of the carry. However, the system of equations is not included here.

Now let us consider the following system, that needs to be satisfied depending on the bits of the key k:

$$\begin{cases} (x \oplus k) + y & = & \mathrm{z} \\ (x \oplus \alpha \oplus k) + (y \oplus \beta) & = & z \oplus \gamma \end{cases}.$$

The restrictions on the key k that satisfy the difference $(\alpha, \beta) \to \gamma$ can be described as follows:

Lemma 4.2. *Let us have $(x \oplus k) + y = z$, $x, k, y, z \in \mathbf{F}_{2^n}$. The values of k satisfying the equations $(x \oplus k) + y = z$ and $((x \oplus \alpha \oplus k) + (y \oplus \beta) = z \oplus \gamma$ must also satisfy the following system of restrictions:*

$$\gamma_0 = \alpha_0 + \beta_0,$$

$$k_{i-1}(\beta_{i-1} + \sum_{j=0}^{i-2}(x_j + k_j)y_j \prod_{l=j+1}^{i-2}(x_l + k_l + y_l) + \sum_{j=0}^{i-2}(x'_j + k_j)y'_j \prod_{k=j+1}^{i-2}(x'_l + k_l + y'_l) =$$

$$= \alpha_i + \beta_i + \gamma_i + (x_{i-1} + y_{i-1}) \sum_{j=0}^{i-2}(x_j + k_j)y_j \prod_{l=j+1}^{i-2}(x_l + k_l + y_l) +$$

$$+ (x'_{i-1} + y'_{i-1}) \sum_{j=0}^{i-2}(x'_j + k_j)y'_j \prod_{l=j+1}^{i-2}(x'_l + k_l + y'_l),$$

where i=1, . . . n-1, $x' = x \oplus \alpha$, $y' = y \oplus \beta$ *and* $z' = z \oplus \gamma$.

Proof. Similarly to the proof of Lemma 4.1, the proof follows from Eqn. 4.1. □

We can translate the results of Lemma 4.1 and Lemma 4.2 by following the idea of phantom differentials in [100]. Let us be given a right pair $(x, x') \Rightarrow (z, z')$. Then we can describe the key that will satisfy the differential transition for the right pair. For the key values that do not satisfy the restrictions we say we have a phantom characteristic, as the differential transition is still valid.

Another model worth analysing is $x + k + y = z$. As this involves two modular additions the deriving of equations restricting the bits of the key k is more complicated. The way we approached this problem is to divide the two additions as

$$x + k = w,$$

$$w + y = z.$$

Then following a technique similar to the one used in Lemma 4.1 and 4.2 we can derive a system of restrictions. The full formula has not been proven yet and is hence not added here.

This research can be continued in several ways. First, note that the last equations on the bits of the key in Lemmas 4.1 and 4.2 take bits of the input and output differences with indexes out of the considered set. This border case remains to be investigated. Further, in some situations it is easier to work with the carry of modular addition. Although we derived equations to express the bits of the key in terms of the carry, these equations have not been implemented and verified empirically, hence not presented here. The same goes

for the situation $x + k + y = z$. Additionally, the idea of translating phantom differentials in terms of having a right pair (input,output) and a key which does or does not fulfill the given system of restrictions is not clearly defined yet. One interpretation is that we will have a phantom differential $(\alpha \to \gamma)$ if there is a key k for which the equations of Lemma 4.1 do not hold for any value of x. This idea can find more interpretations and applications after further consideration.

4.3.3 Rotational cryptanalysis

Rotational cryptanalysis is a probabilistic chosen plaintext technique which targets pairs of plaintexts or keys, such that one is derived from the other by a rotation. The pair $(m, m \lll l)$ for m and its cyclic rotation by l bits is called a *rotational pair*. If the corresponding ciphertexts also form a rotational pair we say that the *rotational property* holds. When the rotational amount l is known from the context or not relevant, for simplicity we use the notation $\overleftarrow{m}$.

Rotational cryptanalysis is a technique mostly applied to ARX primitives. The advantage of rotational cryptanalysis is that a rotational pair is propagated linearly through all bit-wise operations, specifically through an XOR and rotation, but not through a modular addition. In [49] the probability of having a rotational pair of the outputs of modular addition, when the inputs form a rotational pair is calculated and is shown that it depends on the rotational constant and the size of the word. More precisely, the probability is

$$\Pr((x + y) \lll l = x \lll l + y \lll l) = \frac{1}{4}(1 + 2^{l-n} + 2^{-l} + 2^{-n}). \tag{4.3}$$

Keys related to each other by rotation have been used in [27] to exploit the key-schedule of Loki. One of the first works considering a rotational pair of plaintexts is [79]. There is shown that the inner state of Shabal stays unchanged with the choice of rotated components of the input and is hence considered as a fixed point. In [135] the authors use rotational input pairs in the security evaluation of the algorithm SEA. Later, the term rotational cryptanalysis is defined in [73], where the relation is taken also over the plaintexts and used for composing rotational characteristic over Threefish. In [75] is noted that the rotational probability can not be correctly estimated by multiplying the round probabilities, because consecutive modular additions have lower probability. The authors of [73] later correct the formula for computing the rotational probability and prove in [74] that the rotational probability does not depend only on the number of modular additions, but also on their positions. They introduce chains of modular additions, where the output of one is input to the next addition

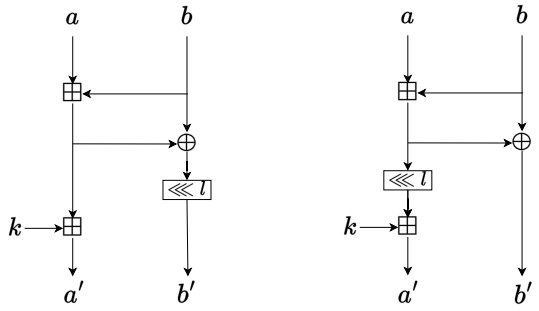

Figure 4.1: ARX primitives with the same number of additions, but different rotational probability

and give a formula for calculating the exact rotational probability of multiple modular additions connected in a chain.

The exact formula is given by the following lemma:

Lemma 4.3. *[74, Lemma 2] Let $a_1, \ldots, a_s$ be n-bit words chosen at random and let l be a positive integer such that $0 < l < n$. Then*

$$\Pr([(a_1 + a_2) \lll l = a_1 \lll l + a_2 \lll l] \wedge$$

$$\wedge\, [(a_1 + a_2 + a_3) \lll l = a_1 \lll l + a_2 \lll l + a_3 \lll l] \wedge$$

$$\ldots$$

$$[(a_1 + \ldots a_s) \lll l = a_1 \lll l + \ldots a_s \lll l]) =$$

$$= \frac{1}{2^{ns}} \binom{s + 2^l - 1}{2^l - 1} \binom{s + 2^{n-l} - 1}{2^{n-l} - 1}.$$

It is further shown that a chain is broken by an XOR or rotation. Figure 4.1 illustrates two ARX primitives which have the same number of modular additions, however their rotational probabilities have different values, because one forms a chain of two additions and the other does not. The final rotational probability is then calculated as the sum of all single and chain probabilities in the primitive. Note, that the probability drops over the use of more chained additions.

Rotational cryptanalysis can be combined with other cryptanalysis techniques to enhance the analysis. In [75] rotational cryptanalysis is combined with the rebound attack [105] to analyse the compression function of Skein [59]. The

rotational distinguisher considers plaintext pairs with corrections to deal with the constant addition. For self-rotational constants such that $c = c \lll l$ the rotational property holds as $(m \oplus c) \lll l = m \lll l \oplus c \lll l$, however for other constants a more sophisticated approach is required. When the constant is not self-rotational and the modular addition does not keep the rotational property, a correction to the input pair is added to compensate the errors. Constant addition is further analysed by Ashur and Liu. In [8] a *rotational XOR difference*, shortly called RX-difference, is defined to be the triplet $((c_1, c_2), l)$, such that $(x \oplus c_1, x \lll l \oplus c_2)$ is a rotational XOR pair. For constants $c_1, c_2, d_1, d_2, e_1, e_2$, the probability of $\overbrace{(x \oplus c_1) + (y \oplus d_1) \oplus e_1 = (\overleftarrow{x} \oplus c_2) + (\overleftarrow{y} \oplus d_2) \oplus e_2}$ can be calculated and a formula to do so is proven. The RX-differences are applied on Speck and further improved in [99].The constant addition in rotational cryptanalysis is analysed in [101] in the context of AND-RX primitives such as Simon.

Recently Liu et al. analyse in [97] differential-linear distinguisher, where the differential part is substituted by rotational XOR-difference and exploits the bias of $\gamma \cdot (rot(E(x)) \oplus E(rot(x) \oplus \delta))$, for encryption function E and some rotational operation $rot(\cdot)$, δ being the rotational XOR difference $(x \lll l) \oplus x'$ and γ being the linear mask. The authors further introduce formulas for estimating the distinguisher's probability and give a link between linear cryptanalysis and rotational XOR cryptanalysis. The proposed improved technique significantly improves the previous results on FRIET and Xoodoo.

A straightforward rotational cryptanalysis is applied on the MAC algorithm Chaskey in our work [84]. The key schedule allows weak keys with certain relation to be established in order to keep the rotational property between the plaintext pairs after the key addition. The algorithm has multiple chains of 2 modular additions and 2 single additions, where the probability of the chained additions is calculated with Lemma 1 and experimentally verified up to 5 rounds out of a total of 12 rounds. In our work on K-Cipher [7] rotational probabilities over modular addition are used to enhance the key-recovery algorithm.

4.3.4 Linear cryptanalysis

Linear Cryptanalysis is a known plaintext attack introduced by Matsui [102]. By analysing the correlation between inputs and outputs of cipher components, we can construct characteristics with high correlations. The bits are selected by a multiplication with a selection vector, having ones only on the chosen positions. We call a hull the set of all characteristics starting and finishing with the same input and output selection vectors. If the number of characteristics is large, then the probability of a hull is a random variable and its distribution

can be estimated to a normal distribution. Useful hulls have a correlation value close to 1 or -1. They can be used to recover the last subkey k_r from the r-th round. Once this key is determined we can unwind the rounds one by one and recover more bits of the key.

An early improvement of linear cryptanalysis is multiple linear cryptanalysis. Multiple linear cryptanalysis [71] uses several independent approximations targeting the same key bits, however different input and output bits are involved. The use of more approximations offers improvement to the data complexity of the attack. Later, this idea is followed by Multidimensional linear cryptanalysis [12], where a linear subspace of linear approximations is called a multidimensional linear approximation.

A generalization of linear cryptanalysis is partitioning cryptanalysis [62]. It considers plaintexts in some particular block of a partition of the plaintexts set. More recently, the zero correlation attack [36] investigates all hulls with correlation zero.

Relevant to linear cryptanalysis is the probability distribution of correlation values. A detailed analysis is presented in [48], where the fixed-key correlations of key-alternating ciphers are analysed and approximated with a gamma distribution. In [1] Abdelraheem et. al. show that contrary to some of the results in [48], there are cases where the distribution of correlations are not as expected. Our work [88] further investigates this matter with respect to a cipher the round function of which is a power function of the type $x^{(2^t+1)2^s}$ for t, s positive integers. For two rounds of such cipher we show that there are only five possible correlation values and classify the possible distributions according to the input selection vector. More specifically, we show that all input selection vectors in the same cyclotomic coset lead to equal distributions, when the output selection vector is set to 1. This can be applied to any other output selection vector.

Chapter 5

Contributions

This chapter discusses the contributions of each of the papers presented in Part II of this book with more details. Each paper has contributed to the state of the art concerning the security of the chosen algorithms. The ciphers we considered are of significant importance. At the time of the paper submissions, the Chaskey MAC algorithm [110] was in the process of standardization, SpoC [4] was a second round candidate of the NIST Lightweight Competition, SNOW3G was part of the 3GPP telecommunications, the PUF algorithms [68, 96, 137] were recently proposed with novel design strategies and the K-cipher [83] was just published with the intention to be used as underlying algorithm for memory encryption.

5.1 New PUF Designs

PUFs are a smart way to take advantage of the randomness generated during the manufacturing process of chips. This randomness is harvested and used to generate a secret key or to achieve other security functions such as device authentication. During the last years the design of PUFs has been influenced by symmetric-key cryptographic algorithms. The PUFs analysed here [96, 137] are such examples adopting the SPN structure.

We analyse three recently proposed PUF designs in [87]. A common misjudgement in all is that the main security evaluation performed by the designers is against machine learning attacks, which seems to miss simple algorithmic attacks such as divide-and-conquer techniques. Our work gives

examples of classical cryptographic attacks with practical complexities that perform better than the evaluated machine learning techniques. We believe even stronger attacks are possible, however we did not pursue better results as we consider it is not necessary. Instead, we provide a guideline towards a secure PUF design from a symmetric-key point of view. Our paper underlines the importance of detailed cryptanalysis even in the cases a certain attack approach is viewed as superior. We encourage the use of cryptographic design techniques in future PUF designs and point out this should be done with more care and that the security goals and design choices should be clearly defined.

5.2 K-Cipher

Low-latency ciphers are desired primitives for many time-sensitive applications. Among the many proposed algorithms, only a small number offer variable input size and variable sbox size. This is the motivation behind K-Cipher - a recently proposed algorithm by Intel researchers [83]. Small-size inputs are needed in the new memory encryption mechanism C^3 [92], which encrypts the pointer to the memory instead of previously used techniques.

We analyse K-cipher and propose a differential attack that has a practical complexity of 2^{31} data for any choice of input size, where the key length is either 96 or 128 bits. Even though the complexity is the same for each version, the attack runs with different actual time depending on the key and block size. For example, it takes around 30 minutes on a desktop computer to recover all secret bits for a 24-bit input with 8-bit sbox version and around 60 minutes for 128-bit input with 16-bit sbox.

The proposed attack fully breaks the K-cipher algorithm and recovers all secret information, namely the key and the unknown random values. This gives us very high confidence that the use of K-Cipher offers less than 32-bits of security. Additionally, we point out that the algorithm overuses randomness, which does not contribute to the security. The impractical use of randomness is an issue also pointed out in the analysis of the PUF designs [87].

5.3 The AEAD algorithm SpoC

SpoC is a second round candidate to the NIST Lightweight competition. It adopts a sponge construction for an AEAD. The candidates of the competition were evaluated according to multiple criteria, most important one being their security. Additional criteria such as Side-Channel Resistance, Nonce-Misuse

Security, RUP Security, Impacts of State Recovery, Post-Quantum Security were added during the evaluation process. Several independent groups performed benchmarking regarding the hardware and software performance of the candidate algorithms. Our paper and a paper by Naya-Plasencia et al. [66] in which the underlying SliSCP-light permutation is analysed were reported as a third-party security evaluation of SpoC. Performance-wise Spoc did not rank between the best algorithms, apart from using the least RAM according to the benchmarking of Renner et al. [125]. SpoC did not continue further in the competition.

Our paper [86] proposes attacks on SpoC in several adversarial models. We performed related-key related-nonce differential selective tag-forgery attacks on round-reduced versions of SpoC, as well as message and key recovery attacks. We describe a 6- and 7-round characteristics, out of 18 rounds over the two versions of the permutation which lead to tag forgeries with data complexity of $2^{106.14}$ on SpoC-128 and $2^{108.2}$ on SpoC-64. A 9-round characteristic is used for message recovery with complexity $2^{110.8}$ for SpoC-128. Additionally, we propose a generic time-memory trade-off key-recovery attack on the full version of SpoC-64 with complexity of 2^{67} data and 2^{110} table entries as memory complexity. This is the strongest attack among our propositions and the strongest model. Even with the predicted security threshold of 112 bits of security instead of 128, as required by the competition, our attack shows the mode of operation of SpoC has vulnerabilities and cannot be considered secure.

5.4 The stream cipher Snow

The 3rd Generation Partnership Project (3GPP) was formed in 1998 to unite seven different standardization organizations around the world in order to develop protocols and technical specifications for mobile telecommunications. SNOW3G is a word-oriented synchronous stream cipher, used as a core of the integrity and confidentiality algorithms UEA2 and UIA2 of the 3GPP UMTS system.

Our work [85] analyses several versions of the SNOW family in terms of differential cryptanalysis. Considered are SNOW3G, the previous standard SNOW2 and their XOR versions, where the modular addition in the FSM is substituted by XOR. Motivated by previous results, we complete them by correcting an unnecessary simplification and extend the analysis with two rounds. We provide a comparison between SNOW2 and SNOW3G, both in their original specifications, and in their simplified versions. An estimation of the minimum number of active sboxes is given by using the branch number of the used MDS matrix. This observation was later found as not correctly defined, however

a correction can be easily made. Our analysis does not identify significant weaknesses and declares that the algorithm provides its claimed security with high confidence.

5.5 The MAC algorithm Chaskey

Chaskey-12 is a MAC algorithm designed for 32-bit microcontrollers, where the number 12 indicates the number of rounds. It is an ISO standard since 2019 together with LightMAC and Tsudik's keymode. These mechanisms can be used as data authentication mechanisms to verify that data has not been altered in an unauthorized manner and that a message has been originated by an entity in possession of the secret key.

We analyze the security of Chaskey-12 with respect to rotational cryptanalysis in [84]. Our results are the first covering all 12 rounds of the algorithm. We consider a weak-key related-key scenario, where a weak-key class of size 2^{120} is identified. We propose a distinguishing attack with complexity 2^{86} and a key-recovery attack with complexity 2^{120}, assuming we are in the weak-key class. In a multi-user scenario this can be exploited to a tag forgery attack. There is room for improvement in the presented rotational attack, for example by considering partitioning of the output, which is left as a future work. We note that having chained modular additions improves the resistance against rotational cryptanalysis rather than single modular additions. The longer the chain of modular additions, the lower the probability of the rotational property.

Even though by definition having complexity less than brute force proves the cipher being broken, the related-key and weak-key scenarios are considered overly strong adversarial models and we can consider that the attack is theoretical and does not really compromise the security.

5.6 Correlation distributions

When analysing a cryptographic primitive we are often forced to make assumptions in order to make the analysis feasible. Often the simulations give results negligibly different from the theoretical results, which is why researchers keep simplifying the model. Some of the most common assumptions are that the probability of a differential is approximated by its expected probability. Another one is that we have independently and uniformly chosen round keys or that distinct linear trails are independent. The hypothesis of stochastic equivalence is known to be the assumption that the probability of a given characteristic varies

negligibly for different keys and is applied often when analyzing Markov ciphers [89]. These at first sight ordinary assumptions create a gap between the practical and theoretical security of ciphers. In [1] the authors analyze a block cipher with unexpected behavior. That is the Cube key-alternating block cipher with round function $f(x) = x^3$. Its plaintext-ciphertext correlations for two rounds take only 5 different values, while in [48] it is suggested this should be a large number. This is a small but important example to demonstrate the mismatch between assumptions and the reality. Another examples of different behavior in [1] are PRESENT and PRINTcipher. The latter has a strong linear relation for any number of rounds. Clustering of trails is observed that also happens to other ciphers with symmetries such as PRINTcipher. In 2020 Liu et al. [100] investigate the phenomenon of differential characteristics that have probability zero $DP(a, b) = 0$, while it is expected to be nonzero $EDP(a, b) > 0$. By accepting the stochastic equivalence we ignore the fact that the characteristics depend on the key-schedule and as a result invalid differentials can be easily mistaken for real.

Our work [88] investigates theoretical aspects of correlation distributions closely related to linear cryptanalysis. It continues the idea presented in [1] and analyzes the correlation distribution of the plaintext-ciphertext pair over all possible keys for two rounds of the Cube cipher with round function $f(x) = x^3$. These correlations are fully derived and classified for any pair of input and output masks and block size n (for odd n). The results are further generalized for a large class of power mappings as round functions of the type $f(x) = x^{(2^t+1)2^s}$ with t, s integers.

Chapter 6

Conclusions and future work

This thesis focuses on cryptanalysis techniques of symmetric-key primitives. Cryptanalysis contributes to the security evaluation and confidence of algorithms, and improving design techniques to be more resistant towards certain attack methods. Due to the large efforts in that direction, currently we have a portfolio of cryptographic algorithms that can be considered quite secure. Once for an "old" cipher as DES was enough to construct a simple differential characteristic to break it, while new ciphers are designed with differential cryptanalysis in mind and is often impossible to construct straightforward characteristics. With this comes the need for more sophisticated analysis of new algorithms. Automated tools are a recent approach that looks more exhaustively for characteristics with non-trivial structure, for example having differences with multiple active bits. This powerful approach is still improving, including more ciphers and cryptanalsyis techniques. Our work on the authenticated encryption cipher SpoC combines an approach of tool-based search and a heuristics search for differential characteristics. We first use a tool to find the best characteristic over the underlying round function based on Simeck. Then we link the corresponding one-round characteristics to form the final differential characteristic over the permutation with the help of graphs. The characteristics constructed are further exploited for several attack scenarios.

Another modern cryptanalysis approach is the use of machine learning to enhance distinguishing attacks. Several works combining neural training and differential techniques have shown better results than previously published attacks, starting with the pioneering work of Gohr in 2019 [61]. Machine learning cryptanalysis techniques have taken a big part particularly in the design of Physically Unclonable Functions, as during the last years this has

been the strongest attack approach against them in the literature. However, we show that simple cryptanalysis techniques are not to be underestimated. Even in the presence of what is considered as a superior cryptanalysis technique, one should not overlook the power of classical techniques. We use differential-based approaches to divide the input space into equivalence classes for two of the three analysed PUFs in [87]. This leads to practical attacks faster than the estimated machine learning attacks by the designers.

Good design requires deep understanding on the existing attacks and how each of the building components link together. Careless design both using known building blocks or novel strategies can possess vulnerabilities not visible at first sight. The design of PUFs has recently been influenced by symmetric key techniques. However, it seems the symbiosis of these two subfields has not yet matured enough. The latest PUFs lack clear definitions of their targeted security and thorough cryptanalysis rationale. While we encourage the use of symmetric-key techniques in future PUF designs, we urge for more care in the process. Another recent lightweight algorithm that can be easily broken with simple techniques is the K-Cipher. There again overuse of randomness can be observed leading to a waste of resources which does not contribute to the security. We completely disregard the random values added to the non-linear layer by constructing a generalized difference distribution table (GDDT). Depending on the random value we construct all possible DDTs and combine them into GDDT with a non-zero entry only if it is non-zero for all DDTs. This way the constructed characteristics are valid regardless of the random value used.

Cryptanalysis reveals specific vulnerabilities in algorithms. A natural continuation of the work of a cryptanalysist is to either improve on an existing algorithm with a certain vulnerability in mind or to design a new one if no reasonable tweak is possible. Based on the research conducted in this thesis several directions towards design of primitives can be taken. More particularly:

Design of a cascaded PUF. Motivated by our recent work on attacks on cascaded PUFs [87], a future work direction is to design a symmetric-key based PUF which will use the randomness provided by the manufacturing process of an integrated circuits efficiently. Few problems of the latest algorithms is that the security design decisions are not motivated, security goals are not clearly defined and the algorithms do not provide sufficient diffusion. We noted in our analysis several directions in which the algorithms can be improved and this matter requires further investigation. In the case that the vulnerabilities mentioned are acceptable for a PUF, this can be achieved with less resources and the algorithms can be optimised.

Design of algorithm for memory encryption The C^3 mechanism for memory encryption [92] requires a lightweight encryption algorithm with the flexibility of having small block size. As there are not many ciphers that satisfy this requirement, the K-cipher [83] was designed with the purpose of fulfilling this need. Our attack on K-cipher shows that it is not secure and the large amount of random bits does not contribute to the security of the algorithm. Therefore, a future research direction is to propose a more secure design which takes small block sizes and is suitable to be used in the C^3 mechanism and other applications with similar requirements.

A different direction, however of an interest to us is the following:

Phantom characteristics for ARX primitives In Section 4.3.2 we introduced a new concept of using differences over modular addition in order to reduce the possible key space. A possible application is when the key is added with a modular addition. This research was motivated by the phantom differential characteristics introduced by Liu et al. [100] and can be continued and further developed. We aim at defining the concept of phantoms in the context of ARX primitives.

Bibliography

[1] Abdelraheem, M. A., Ågren, M., Beelen, P., and Leander, G. On the distribution of linear biases: Three instructive examples. In *Advances in Cryptology - CRYPTO* (2012), vol. 7417 of *Lecture Notes in Computer Science*, Springer, pp. 50–67.

[2] Adams, C. The CAST-128 encryption algorithm. *RFC 2144* (1997), 1–15.

[3] Advanced Encryption Standard (AES). National Institute of Standards and Technology (NIST), FIPS PUB 197, U.S. Department of Commerce, Nov. 2001.

[4] AlTawy, R., Gong, G., He, M., Jha, A., Mandal, K., Nandi, M., and Rohit, R. SpoC:An Authenticated Cipher Submission to the NIST LWC Competition, 2019. `https://csrc.nist.gov/CSRC/media/Projects/lightweight-cryptography/documents/round-2/spec-doc-rnd2/spoc-spec-round2.pdf`.

[5] Andreeva, E., Bogdanov, A., Datta, N., Luykx, A., Mennink, B., Nandi, M., Tischhauser, E., and Yasuda, K. COLM v1. Submission to CAESAR competition. http://ascon.iaik.tugraz.at, 2015.

[6] Ankele, R., and Kölbl, S. Mind the gap - A closer look at the security of block ciphers against differential cryptanalysis. In *Selected Areas in Cryptography - SAC* (2018), vol. 11349 of *Lecture Notes in Computer Science*, Springer, pp. 163–190.

[7] Ashur, T., Kraleva, L., Mahzoun, M., and Posteuca, R. Differential Cryptanalysis of K-Cipher. In *27th IEEE International Symposium on Computers and Communications (ISCC)* (2022), p. 11.

[8] Ashur, T., and Liu, Y. Rotational cryptanalysis in the presence of constants. *IACR Trans. Symmetric Cryptol. 2016*, 1 (2016), 57–70.

[9] Aumasson, J., Meier, W., Phan, R. C., and Henzen, L. The hash function BLAKE. https://doi.org/10.1007/978-3-662-44757-4.

[10] Aumasson, J.-P., Neves, S., Wilcox-O'Hearn, Z., and Winnerlein, C. BLAKE2: simpler, smaller, fast as MD5, 2013. https://www.blake2.net/blake2.pdf.

[11] Avanzi, R. The QARMA block cipher family. almost MDS matrices over rings with zero divisors, nearly symmetric even-mansour constructions with non-involutory central rounds, and search heuristics for low-latency s-boxes. *IACR Trans. Symmetric Cryptol. 2017*, 1 (2017), 4–44.

[12] Baignères, T., Junod, P., and Vaudenay, S. How far can we go beyond linear cryptanalysis? In *Advances in Cryptology - ASIACRYPT* (2004), vol. 3329 of *Lecture Notes in Computer Science*, Springer, pp. 432–450.

[13] Banik, S., Bogdanov, A., Isobe, T., Shibutani, K., Hiwatari, H., Akishita, T., and Regazzoni, F. Midori: A block cipher for low energy. In *Advances in Cryptology - ASIACRYPT 2015(2)* (2015), vol. 9453 of *Lecture Notes in Computer Science*, Springer, pp. 411–436.

[14] Bao, Z., Chakraborti, A., Datta, N., Guo, J., Nandi, M., Peyrin, T., and Yasuda, K. PHOTON-Beetle Authenticated Encryption. Submission to the NIST Lightweight Cryptography project, 2019. https://www.isical.ac.in/~lightweight/beetle/.

[15] Batina, L., Das, A., Ege, B., Kavun, E. B., Mentens, N., Paar, C., Verbauwhede, I., and Yalçın, T. Dietary recommendations for lightweight block ciphers: Power, energy and area analysis of recently developed architectures. In *Radio Frequency Identification - Security and Privacy Issues 9th International Workshop, RFIDsec* (2013), vol. 8262 of *Lecture Notes in Computer Science*, Springer, pp. 103–112.

[16] Beaulieu, R., Shors, D., Smith, J., Treatman-Clark, S., Weeks, B., and Wingers, L. SIMON and SPECK: block ciphers for the internet of things. *IACR Cryptol. ePrint Arch.* (2015), 585.

[17] Beierle, C., Biryukov, A., dos Santos, L. C., Grossschädl, J., Perrin, L., Udovenko, A., Velichkov, V., Wang, Q., Moradi, A., and Shahmirzadi, A. R. Sparke-suite. Submission to the NIST Lightweight Cryptography project, 2019. https://sparkle-lwc.github.io/.

[18] Beierle, C., Jean, J., Kölbl, S., Leander, G., Moradi, A., Peyrin, T., Sasaki, Y., Sasdrich, P., and Sim, S. M. The SKINNY family

of block ciphers and its low-latency variant MANTIS. In *Advances in Cryptology - CRYPTO(2)* (2016), vol. 9815 of *Lecture Notes in Computer Science*, Springer, pp. 123–153.

[19] Beierle, C., Leander, G., and Todo, Y. Improved differential-linear attacks with applications to ARX ciphers. In *Advances in Cryptology - CRYPTO (3)* (2020), vol. 12172 of *Lecture Notes in Computer Science*, Springer, pp. 329–358.

[20] Bellare, M., and Namprempre, C. Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. In *Advances in Cryptology - ASIACRYPT* (2000), vol. 1976 of *Lecture Notes in Computer Science*, Springer, pp. 531–545.

[21] Ben-Aroya, I., and Biham, E. Differential Cryptanalysis of Lucifer. *J. Cryptol. 9*, 1 (1996), 21–34.

[22] Benoît, O. *Fault Attack*. Springer US, Boston, MA, 2005, pp. 218–219. https://doi.org/10.1007/0-387-23483-7_157.

[23] Bernstein, D. J. The ChaCha family of stream ciphers, 2008. http://cr.yp.to/chacha.html.

[24] Bernstein, D. J. The Salsa20 Family of Stream Ciphers. In *New Stream Cipher Designs - The eSTREAM Finalists*, vol. 4986 of *Lecture Notes in Computer Science*. Springer, 2008, pp. 84–97.

[25] Bertoni, G., Daemen, J., Hoffert, S., l Peeters, M., Assche, G. V., and Keer, R. V. The Keccak reference (2011). http://keccak.noekeon.org, 2011. http://keccak.noekeon.org.

[26] Beyne, T., Chen, Y. L., Dobraunig, C., and Mennink, B. Elephant. Submission to the NIST Lightweight Cryptography Project, 2019. https://www.esat.kuleuven.be/cosic/elephant/.

[27] Biham, E. New types of cryptanalytic attacks using related keys. *J. Cryptol. 7*, 4 (1994), 229–246.

[28] Biham, E., Biryukov, A., and Shamir, A. Cryptanalysis of Skipjack Reduced to 31 Rounds Using Impossible Differentials. In *Advances in Cryptology - EUROCRYPT* (1999), vol. 1592 of *Lecture Notes in Computer Science*, Springer, pp. 12–23.

[29] Biham, E., Dunkelman, O., and Keller, N. The Rectangle Attack - Rectangling the Serpent. In *Advances in Cryptology - EUROCRYPT* (2001), vol. 2045 of *Lecture Notes in Computer Science*, Springer, pp. 340–357.

[30] Biham, E., and Shamir, A. Differential cryptanalysis of DES-like cryptosystems. In *Advances in Cryptology – CRYPTO'90* (Santa Barbara, CA, USA, Aug. 11–15, 1991), A. J. Menezes and S. A. Vanstone, Eds., vol. 537 of *Lecture Notes in Computer Science*, Springer, Heidelberg, Germany, pp. 2–21.

[31] Biryukov, A., Khovratovich, D., and Nikolic, I. Distinguisher and related-key attack on the full AES-256. In *Advances in Cryptology - CRYPTO* (2009), vol. 5677 of *Lecture Notes in Computer Science*, Springer, pp. 231–249.

[32] Biryukov, A., and Perrin, L. State of the art in lightweight symmetric cryptography. *IACR Cryptol. ePrint Arch.* (2017), 511.

[33] Biryukov, A., Velichkov, V., and Corre, Y. L. Automatic search for the best trails in ARX: application to block cipher Speck. In *FSE* (2016), vol. 9783 of *Lecture Notes in Computer Science*, Springer, pp. 289–310.

[34] Bogdanov, A., Khovratovich, D., and Rechberger, C. Biclique cryptanalysis of the full AES. In *Advances in Cryptology - ASIACRYPT* (2011), vol. 7073 of *Lecture Notes in Computer Science*, Springer, pp. 344–371.

[35] Bogdanov, A., Knudsen, L. R., Leander, G., Paar, C., Poschmann, A., Robshaw, M. J. B., Seurin, Y., and Vikkelsoe, C. PRESENT: an ultra-lightweight block cipher. In *Cryptographic Hardware and Embedded Systems - CHES* (2007), vol. 4727 of *Lecture Notes in Computer Science*, Springer, pp. 450–466.

[36] Bogdanov, A., and Rijmen, V. Linear hulls with correlation zero and linear cryptanalysis of block ciphers. *Des. Codes Cryptogr. 70*, 3 (2014), 369–383.

[37] Borst, J., Knudsen, L. R., and Rijmen, V. Two attacks on reduced IDEA. In *Advances in Cryptology - EUROCRYPT* (1997), vol. 1233 of *Lecture Notes in Computer Science*, Springer, pp. 1–13.

[38] Bosselaers, A. The hash function RIPEMD-160. `https://homes.esat.kuleuven.be/~bosselae/ripemd160.html`.

[39] Briceno, M., Goldberg, I., and Wagner, D. A pedagogical implementation of the GSM A5/1 and A5/2 "voice privacy" encryption algorithms, 1999. https://cryptome.org/gsm-a512.htm.

[40] Cannière, C. D., and Preneel, B. Trivium. In *New Stream Cipher Designs - The eSTREAM Finalists*, vol. 4986 of *Lecture Notes in Computer Science*. Springer, 2008, pp. 244–266.

[41] Cho, J. Y., and Pieprzyk, J. Multiple modular additions and crossword puzzle attack on nlsv2. In *ISC* (2007), vol. 4779 of *Lecture Notes in Computer Science*, Springer, pp. 230–248.

[42] Civino, R., Blondeau, C., and Sala, M. Differential attacks: using alternative operations. *Des. Codes Cryptogr. 87*, 2-3 (2019), 225–247.

[43] Courtois, N. T., and Meier, W. Algebraic attacks on stream ciphers with linear feedback. In *Advances in Cryptology - EUROCRYPT* (2003), vol. 2656 of *Lecture Notes in Computer Science*, Springer, pp. 345–359.

[44] CRYPTREC Lightweight Cryptography Working Group. CRYPTREC cryptographic technology guideline (lightweight cryptography), 2017. Available at: `http://www.cryptrec.go.jp/report/cryptrec-gl-2016-e.pdf`.

[45] Daemen, J. Limitations of the Even-Mansour Construction. In *ASIACRYPT* (1991), vol. 739 of *Lecture Notes in Computer Science*, Springer, pp. 495–498.

[46] Daemen, J., and Kitsos, P. The Self-synchronizing Stream Cipher Moustique. In *New Stream Cipher Designs - The eSTREAM Finalists*, vol. 4986 of *Lecture Notes in Computer Science*. Springer, 2008, pp. 210–223.

[47] Daemen, J., and Rijmen, V. The wide trail design strategy. In *Cryptography and Coding, 8th IMA International Conference* (2001), vol. 2260 of *Lecture Notes in Computer Science*, Springer, pp. 222–238.

[48] Daemen, J., and Rijmen, V. Probability distributions of correlation and differentials in block ciphers. *J. Math. Cryptol. 1*, 3 (2007), 221–242.

[49] Daum, M. *Cryptanalysis of Hash functions of the MD4-family.* PhD thesis, Ruhr University Bochum, 2005.

[50] Delvaux, J., Gu, D., Schellekens, D., and Verbauwhede, I. Helper data algorithms for puf-based key generation: Overview and analysis. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. 34*, 6 (2015), 889–902.

[51] Delvaux, J., Peeters, R., Gu, D., and Verbauwhede, I. A survey on lightweight entity authentication with strong pufs. *ACM Comput. Surv. 48*, 2 (2015), 26:1–26:42.

[52] Diehl, W., Farahmand, F., Yalla, P., Kaps, J., and Gaj, K. Comparison of hardware and software implementations of selected

lightweight block ciphers. In *27th International Conference on Field Programmable Logic and Applications, FPL* (2017), IEEE, pp. 1–4.

[53] Dinu, D., Perrin, L., Udovenko, A., Velichkov, V., Grossschädl, J., and Biryukov, A. Design strategies for ARX with provable bounds: Sparx and LAX. In *Advances in Cryptology - ASIACRYPT (1)* (2016), vol. 10031 of *Lecture Notes in Computer Science*, pp. 484–513.

[54] Dobraunig, C., Eichlseder, M., Mendel, F., and ffer, M. S. Ascon *v1. [online]* (2014). Available: http://competitions.cr.yp.to/round1/asconv1.pdf.

[55] ETSI. Specification of the 3GPP confidentiality and integrity algorithms; Document 2: Kasumi specification. `https://www.etsi.org/deliver/etsi_ts/135200_135299/135202/17.00.00_60/ts_135202v170000p.pdf`.

[56] ETSI. Specification of the 3GPP Confidentiality and Integrity Algorithms UEA2 & UIA2. Document 2: SNOW3G Specification, 2006. Technical report. `https://www.gsma.com/aboutus/wp-content/uploads/2014/12/snow3gspec.pdf`.

[57] ETSI. Specification of the 3GPP Confidentiality and Integrity Algorithms 128-EEA3 & 128-EIA3. Document 2: ZUC Specification, 2011. Technical report.

[58] Even, S., and Mansour, Y. A Construction of a Cipher from a Single Pseudorandom Permutation. *J. Cryptology 10*, 3 (1997), 151–162.

[59] Ferguson, N., Bauhaus, S. L., Schneier, B., Hifn, D. W., Bellare, M., Kohno, T., Callas, J., and Walker, J. The Skein Hash Function Family. `https://www.schneier.com/wp-content/uploads/2016/02/skein.pdf`.

[60] Gassend, B., Clarke, D. E., van Dijk, M., and Devadas, S. Silicon physical random functions. In *Proceedings of the 9th ACM Conference on Computer and Communications Security, CCS* (2002), ACM, pp. 148–160.

[61] Gohr, A. Improving attacks on round-reduced Speck32/64 using deep learning. In *Advances in Cryptology - CRYPTO(2)* (2019), vol. 11693 of *Lecture Notes in Computer Science*, Springer, pp. 150–179.

[62] Harpes, C., and Massey, J. L. Partitioning cryptanalysis. In *FSE* (1997), vol. 1267 of *Lecture Notes in Computer Science*, Springer, pp. 13–27.

[63] Hell, M., Johansson, T., Maximov, A., and Meier, W. The Grain Family of Stream Ciphers. In *New Stream Cipher Designs - The eSTREAM Finalists*, vol. 4986 of *Lecture Notes in Computer Science*. Springer, 2008, pp. 179–190.

[64] Hoang, V. T., and Rogaway, P. On generalized Feistel networks. In *Advances in Cryptology – CRYPTO 2010* (Santa Barbara, CA, USA, Aug. 15–19, 2010), T. Rabin, Ed., vol. 6223 of *Lecture Notes in Computer Science*, Springer, Heidelberg, Germany, pp. 613–630.

[65] Hong, D., Sung, J., Hong, S., Lim, J., Lee, S., Koo, B., Lee, C., Chang, D., Lee, J., Jeong, K., Kim, H., Kim, J., and Chee, S. HIGHT: A new block cipher suitable for low-resource device. In *Cryptographic Hardware and Embedded Systems - CHES* (2006), vol. 4249 of *Lecture Notes in Computer Science*, Springer, pp. 46–59.

[66] Hosoyamada, A., Naya-Plasencia, M., and Sasaki, Y. Improved Attacks on sLiSCP Permutation and Tight Bound of Limited Birthday Distinguishers. *IACR Trans. Symmetric Cryptol. 2020*, 4 (2020), 147–172.

[67] Jean, J., Moradi, A., Peyrin, T., and Sasdrich, P. Bit-sliding: A generic technique for bit-serial implementations of spn-based primitives - applications to aes, PRESENT and SKINNY. In *Cryptographic Hardware and Embedded Systems - CHES* (2017), vol. 10529 of *Lecture Notes in Computer Science*, Springer, pp. 687–707.

[68] Jeloka, S., Yang, K., Orshansky, M., Sylvester, D., and Blaauw, D. A sequence dependent challenge-response PUF using 28nm sram 6t bit cell. In *2017 Symposium on VLSI Circuits* (2017), pp. C270–C271.

[69] Jimale, M. A., Z'aba, M. R., Kiah, M. L. M., Idris, M. Y. I. B., Jamil, N., Mohamad, M. S., and Rohmad, M. S. Authenticated encryption schemes: A systematic review. *IEEE Access 10* (2022), 14739–14766.

[70] Joye, M., and Olivier, F. *Side-Channel Analysis*. Springer US, Boston, MA, 2005, pp. 571–571. https://doi.org/10.1007/0-387-23483-7_394.

[71] Jr., B. S. K., and Robshaw, M. J. B. Linear cryptanalysis using multiple approximations. In *CRYPTO* (1994), vol. 839 of *Lecture Notes in Computer Science*, Springer, pp. 26–39.

[72] Käsper, E., Rijmen, V., Bjørstad, T. E., Rechberger, C., Robshaw, M. J. B., and Sekar, G. Correlated keystreams in Moustique.

In *Progress in Cryptology - AFRICACRYPT* (2008), vol. 5023 of *Lecture Notes in Computer Science*, Springer, pp. 246–257.

[73] Khovratovich, D., and Nikolic, I. Rotational Cryptanalysis of ARX. In *FSE* (2010), vol. 6147 of *Lecture Notes in Computer Science*, Springer, pp. 333–346.

[74] Khovratovich, D., Nikolic, I., Pieprzyk, J., Sokolowski, P., and Steinfeld, R. Rotational Cryptanalysis of ARX Revisited. In *FSE* (2015), vol. 9054 of *Lecture Notes in Computer Science*, Springer, pp. 519–536.

[75] Khovratovich, D., Nikolic, I., and Rechberger, C. Rotational Rebound Attacks on Reduced Skein. *J. Cryptol. 27*, 3 (2014), 452–479.

[76] Knezevic, M., Nikov, V., and Rombouts, P. Low-latency encryption - is "lightweight = light + wait"? In *Cryptographic Hardware and Embedded Systems - CHES* (2012), vol. 7428 of *Lecture Notes in Computer Science*, Springer, pp. 426–446.

[77] Knudsen, L. R. New potentially 'weak' keys for DES and LOKI (extended abstract). In *EUROCRYPT* (1994), vol. 950 of *Lecture Notes in Computer Science*, Springer, pp. 419–424.

[78] Knudsen, L. R. Truncated and Higher Order Differentials. In *FSE* (1994), vol. 1008 of *Lecture Notes in Computer Science*, Springer, pp. 196–211.

[79] Knudsen, L. R., Matusiewicz, K., and Thomsen, S. S. Observations on the Shabal keyed permutation. *Available online* (2009). http://www2.mat.dtu.dk/people/oldusers/S.Thomsen/shabal/shabal.pdf.

[80] Knudsen, L. R., and Rijmen, V. Known-key distinguishers for some block ciphers. In *Advances in Cryptology - ASIACRYPT* (2007), vol. 4833 of *Lecture Notes in Computer Science*, Springer, pp. 315–324.

[81] Knudsen, L. R., and Wagner, D. A. Integral cryptanalysis. In *FSE* (2002), vol. 2365 of *Lecture Notes in Computer Science*, Springer, pp. 112–127.

[82] Koeune, F. *Timing Attack*. Springer US, Boston, MA, 2011, pp. 1303–1305. ttps://doi.org/10.1007/978-1-4419-5906-5_138.

[83] Kounavis, M. E., Deutsch, S., Ghosh, S., and Durham, D. K-cipher: A low latency, bit length parameterizable cipher. In *2020 IEEE Symposium on Computers and Communications (ISCC)* (2020), pp. 1–7.

[84] Kraleva, L., Ashur, T., and Rijmen, V. Rotational cryptanalysis on MAC algorithm chaskey. In *ACNS (1)* (2020), vol. 12146 of *Lecture Notes in Computer Science*, Springer, pp. 153–168.

[85] Kraleva, L., and Li, C. On the SNOW Stream Ciphers. *In Proceedings of the 41st Symposium on Information Theory in the Benelux, R. Van Sloun, and B. Škoric (eds.), Werkgemeenschap voor Informatie- en Communicatietheorie,* (2021), 116–123.

[86] Kraleva, L., Posteuca, R., and Rijmen, V. Cryptanalysis of the Permutation Based Algorithm SpoC. In *INDOCRYPT* (2020), vol. 12578 of *Lecture Notes in Computer Science*, Springer, pp. 273–293.

[87] Kraleva, L., Posteucca, R., Mahzoun, M., and Ashur, T. Pufs. To be submitted.

[88] Kraleva, L., Rijmen, V., and Manev, N. Correlation Distribution Analysis of a Two-Round Key-Alternating Block Cipher. *Tatra Mountains Mathematical Publications 73(1)* (2019), 109–130.

[89] Lai, X., Massey, J. L., and Murphy, S. Markov ciphers and differential cryptanalysis. In *EUROCRYPT* (1991), vol. 547 of *Lecture Notes in Computer Science*, Springer, pp. 17–38.

[90] Lamberger, M., Mendel, F., Rechberger, C., Rijmen, V., and Schläffer, M. Rebound distinguishers: Results on the full whirlpool compression function. In *Advances in Cryptology - ASIACRYPT* (2009), vol. 5912 of *Lecture Notes in Computer Science*, Springer, pp. 126–143.

[91] Lauridsen, M. M., and Rechberger, C. Linear distinguishers in the key-less setting: Application to PRESENT. In *FSE* (2015), vol. 9054 of *Lecture Notes in Computer Science*, Springer, pp. 217–240.

[92] LeMay, M., Rakshit, J., Deutsch, S., Durham, D. M., Ghosh, S., Nori, A., Gaur, J., Weiler, A., Sultana, S., Grewal, K., and Subramoney, S. Cryptographic capability computing. In *MICRO '21: 54th Annual IEEE/ACM International Symposium on Microarchitecture* (2021), ACM, pp. 253–267.

[93] Lipmaa, H., and Moriai, S. Efficient algorithms for computing differential properties of addition. In *FSE* (2001), vol. 2355 of *Lecture Notes in Computer Science*, Springer, pp. 336–350.

[94] Lipmaa, H., and Moriai, S. Efficient algorithms for computing differential properties of addition. In *FSE* (2001), vol. 2355 of *Lecture Notes in Computer Science*, Springer, pp. 336–350.

[95] Liu, F., Isobe, T., Meier, W., and Yang, Z. Algebraic Attacks on Round-Reduced Keccak. In *Information Security and Privacy - 26th Australasian Conference, ACISP* (2021), vol. 13083 of *Lecture Notes in Computer Science*, Springer, pp. 91–110.

[96] Liu, K., Fu, Z., Li, G., Pu, H., Guan, Z., Want, X., Chen, X., and Shinohara, H. 36.3 A Modeling Attack Resilient Strong PUF with Feedback-SPN Structure Having $< 0.73\%$ Bit Error Rate Through In-Cell Hot-Carrier Injection Burn-In. In *2021 IEEE International Solid- State Circuits Conference (ISSCC)* (2021), vol. 64, pp. 502–504.

[97] Liu, Y., Sun, S., and Li, C. Rotational Cryptanalysis from a Differential-Linear Perspective - Practical Distinguishers for Round-Reduced FRIET, Xoodoo, and Alzette. In *Advances in Cryptology - EUROCRYPT (1)* (2021), vol. 12696 of *Lecture Notes in Computer Science*, Springer, pp. 741–770.

[98] Liu, Y., Wang, Q., and Rijmen, V. Automatic search of linear trails in ARX with applications to SPECK and chaskey. In *Applied Cryptography and Network Security Conference, ACNS* (2016), vol. 9696 of *Lecture Notes in Computer Science*, Springer, pp. 485–499.

[99] Liu, Y., Witte, G. D., Ranea, A., and Ashur, T. Rotational-XOR Cryptanalysis of Reduced-round SPECK. *IACR Trans. Symmetric Cryptol. 2017*, 3 (2017), 24–36.

[100] Liu, Y., Zhang, W., Sun, B., Rijmen, V., Liu, G., Li, C., Fu, S., and Cao, M. The phantom of differential characteristics. *Des. Codes Cryptogr. 88*, 11 (2020), 2289–2311.

[101] Lu, J., Liu, Y., Ashur, T., Sun, B., and Li, C. Rotational-XOR Cryptanalysis of Simon-Like Block Ciphers. In *Information Security and Privacy - 25th Australasian Conference, ACISP* (2020), vol. 12248 of *Lecture Notes in Computer Science*, Springer, pp. 105–124.

[102] Matsui, M. Linear cryptanalysis method for DES cipher. In *Advances in Cryptology - EUROCRYPT* (1993), vol. 765 of *Lecture Notes in Computer Science*, Springer, pp. 386–397.

[103] Maurer, U. M. New approaches to the design of self-synchronizing stream ciphers. In *Advances in Cryptology - EUROCRYPT* (1991), vol. 547 of *Lecture Notes in Computer Science*, Springer, pp. 458–471.

[104] Meier, W., and Staffelbach, O. Fast correlation attacks on certain stream ciphers. *J. Cryptol. 1*, 3 (1989), 159–176.

[105] Mendel, F., Rechberger, C., Schläffer, M., and Thomsen, S. S. The rebound attack: Cryptanalysis of reduced whirlpool and grøstl. In *FSE* (2009), vol. 5665 of *Lecture Notes in Computer Science*, Springer, pp. 260–276.

[106] Merkle, R. Secrecy, authentication, and public key systems. *Stanford Ph.D. thesis* (1979), 13–15.

[107] Millerioux, G., and Guillot, P. Self-synchronizing stream ciphers and dynamical systems: State of the art and open issues. *Int. J. Bifurc. Chaos 20*, 9 (2010), 2979–2991.

[108] Miyaguchi, S. The FEAL cipher family. In *CRYPTO* (1990), vol. 537 of *Lecture Notes in Computer Science*, Springer, pp. 627–638.

[109] Moore, J. H., and Simmons, G. J. Cycle structures of the DES with weak and semi-weak keys. In *Advances in Cryptology - CRYPTO* (1986), vol. 263 of *Lecture Notes in Computer Science*, Springer, pp. 9–32.

[110] Mouha, N., Mennink, B., Herrewege, A. V., Watanabe, D., Preneel, B., and Verbauwhede, I. Chaskey: An Efficient MAC Algorithm for 32-bit Microcontrollers. In *Selected Areas in Cryptography* (2014), vol. 8781 of *Lecture Notes in Computer Science*, Springer, pp. 306–323.

[111] Mouha, N., and Preneel, B. Towards finding optimal differential characteristics for ARX: Application to Salsa20. Cryptology ePrint Archive, Report 2013/328, 2013. `https://eprint.iacr.org/2013/328`.

[112] National Institute of Standards and Technology (NIST). FIPS 46: Data Encryption Standard (DES)., 1979. `http://csrc.nist.gov/publications/fips/fips46-3/fips46-3.pdf`.

[113] National Institute of Standards and Technology (NIST). FIPS 180-4: Secure Hash Standard (SHS), 20015. `https://www.nist.gov/publications/secure-hash-standard`.

[114] National Institute of Standards and Technology (NIST). FIPS 202: SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions, 2015. , `https://csrc.nist.gov/publications/detail/fips/202/final`.

[115] National Institute of Standards and Technology (NIST). Lightweight Cryptography Competition, 2019. `https://csrc.nist.gov/projects/lightweight-cryptography`.

[116] Needham, R. M., and Wheeler, D. J. TEA extensions. *Computer Laboratory, University of Cambridge (Technical report)* (October 1997). http://www.cix.co.uk/~klockstone/xtea.pdf.

[117] NIST Special Publication 800-38B. Recommendation for block cipher modes of operation: the CMAC mode for authentication, May 2005.

[118] Park, J., Lee, S., Kim, J., and Lee, J. The SEED encryption algorithm. *RFC 4009* (2005), 1–17.

[119] Patrick, C., and Schaumont, P. The role of energy in the lightweight cryptographic profile, 2016.

[120] Paul, S., and Preneel, B. Solving systems of differential equations of addition. In *Information Security and Privacy, 10th Australasian Conference, ACISP* (2005), vol. 3574 of *Lecture Notes in Computer Science*, Springer, pp. 75–88.

[121] Preneel, B. *Modes of Operation of a Block Cipher*. Springer US, Boston, MA, 2011, pp. 789–794.

[122] Preneel, B., Govaerts, R., and Vandewalle, J. Hash functions based on block ciphers: A synthetic approach. In *Advances in Cryptology - CRYPTO* (1993), vol. 773 of *Lecture Notes in Computer Science*, Springer, pp. 368–378.

[123] Ranea, A., Liu, Y., and Ashur, T. An easy to use tool for Rotational-XOR Cryptanalysis of ARX Block Ciphers, 2017. https://github.com/ranea/ArxPy.

[124] Ranea, A., and Rijmen, V. CASCADA: characteristic automated search of cryptographic algorithms for distinguishing attacks. *IACR Cryptol. ePrint Arch.* (2022), 513.

[125] Renner, S., Pozzobon, E., and Mottok, J. NIST LWC Software Performance Benchmarks on Microcontrollers, 2021. Project webpage. https://lwc.las3.de/.

[126] Rijmen, V., and Barreto, P. S. L. M. The WHIRLPOOL Hash Function. https://web.archive.org/web/20171129084214/http://www.larc.usp.br/~pbarreto/WhirlpoolPage.html.

[127] Rivest, R. The RC4 encryption algorithm. *RSA Data Security Inc.* (1992). This document has not been made public.

[128] Rogaway, P. Authenticated-encryption with associated-data. In *Conference on Computer and Communications Security, CCS* (2002), ACM, pp. 98–107.

[129] Rolfes, C., Poschmann, A., Leander, G., and Paar, C. Ultra-lightweight implementations for smart devices - security for 1000 gate equivalents. In *CARDIS* (2008), vol. 5189 of *Lecture Notes in Computer Science*, Springer, pp. 89–103.

[130] Sasaki, Y., and Todo, Y. New impossible differential search tool from design and cryptanalysis aspects - revealing structural properties of several ciphers. In *Advances in Cryptology - EUROCRYPT (3)* (2017), vol. 10212 of *Lecture Notes in Computer Science*, pp. 185–215.

[131] Schneier, B., Kelsey, J., Whiting, D., Wagner, D. A., Hall, C., and Ferguson, N. The Twofish encryption algorithm: a 128-bit block cipher, 1999.

[132] Shannon, C. E. Communication theory of secrecy systems. *Bell Syst. Tech. J. 28*, 4 (1949), 656–715.

[133] Shibutani, K., Isobe, T., Hiwatari, H., Mitsuda, A., Akishita, T., and Shirai, T. Piccolo: An ultra-lightweight blockcipher. In *Cryptographic Hardware and Embedded Systems - CHES* (2011), vol. 6917 of *Lecture Notes in Computer Science*, Springer, pp. 342–357.

[134] Song, L., Huang, Z., and Yang, Q. Automatic differential analysis of ARX block ciphers with application to SPECK and LEA. In *Information Security and Privacy - 21st Australasian Conference, ACISP(2)* (2016), vol. 9723 of *Lecture Notes in Computer Science*, Springer, pp. 379–394.

[135] Standaert, F., Piret, G., Gershenfeld, N., and Quisquater, J. SEA: A scalable encryption algorithm for small embedded applications. In *Smart Card Research and Advanced Applications CARDIS* (2006), vol. 3928 of *Lecture Notes in Computer Science*, Springer, pp. 222–236.

[136] Stubblefield, A., Ioannidis, J., and Rubin, A. D. Using the Fluhrer, Mantin, and Shamir Attack to Break WEP. In *Proceedings of the Network and Distributed System Security Symposium, NDSS* (2002), The Internet Society.

[137] Suresh, V. B., Kumar, R., Anders, M., Kaul, H., De, V., and Mathew, S. A 0.26% BER, 10^{28} Challenge-Response Machine-Learning Resistant Strong-PUF in 14nm CMOS Featuring Stability-Aware Adversarial Challenge Selection. In *IEEE Symposium on VLSI Circuits, VLSI Circuits* (2020), IEEE, pp. 1–2.

[138] The eSTREAM project. `http://www.ecrypt.eu.org/stream/`.

[139] Tezcan, C. The improbable differential attack: Cryptanalysis of reduced round CLEFIA. In *Progress in Cryptology - INDOCRYPT* (2010), vol. 6498 of *Lecture Notes in Computer Science*, Springer, pp. 197–209.

[140] Todo, Y., Leander, G., and Sasaki, Y. Nonlinear Invariant Attack - Practical Attack on Full SCREAM, iSCREAM, and Midori64. In *Advances in Cryptology - ASIACRYPT (2)* (2016), vol. 10032 of *Lecture Notes in Computer Science*, pp. 3–33.

[141] Turner, J. The keyed-hash message authentication code (HMAC). *FIPS PUB 198-1, National Institute of Standards and Technology (NIST) (July 2008)* (2008).

[142] Wagner, D. A. The Boomerang Attack. In *FSE* (1999), vol. 1636 of *Lecture Notes in Computer Science*, Springer, pp. 156–170.

[143] Wallén, J. Linear approximations of addition modulo 2^n. In *FSE* (2003), vol. 2887 of *Lecture Notes in Computer Science*, Springer, pp. 261–273.

[144] Wang, X., Yin, Y. L., and Yu, H. Finding collisions in the full SHA-1. In *Advances in Cryptology - CRYPTO* (2005), vol. 3621 of *Lecture Notes in Computer Science*, Springer, pp. 17–36.

[145] Wang, X., and Yu, H. How to break MD5 and other hash functions. In *Advances in Cryptology - EUROCRYPT* (2005), vol. 3494 of *Lecture Notes in Computer Science*, Springer, pp. 19–35.

[146] Wheeler, D. J., and Needham, R. M. TEA, a Tiny Encryption Algorithm. In *FSE* (1994), vol. 1008 of *Lecture Notes in Computer Science*, Springer, pp. 363–366.

[147] Winternitz, R. S. A secure one-way hash function built from DES. In *Proceedings of the 1984 IEEE Symposium on Security and Privacy* (1984), IEEE Computer Society, pp. 88–90.

[148] Wu, H. A new stream cipher HC-256. In *FSE* (2004), vol. 3017 of *Lecture Notes in Computer Science*, Springer, pp. 226–244.

[149] Wu, H. The stream cipher HC-128. In *New Stream Cipher Designs - The eSTREAM Finalists*, vol. 4986 of *Lecture Notes in Computer Science*. Springer, 2008, pp. 39–47.

[150] Wu, H., and Huang, T. TinyJAMBU: A Family of Lightweight Authenticated Encryption Algorithms (Version 2). Submission to the NIST Lightweight Cryptography Project, 2019. `https://csrc.nist.gov/CSRC/media/Projects/lightweight-`

cryptography/documents/finalist-round/updated-spec-doc/tinyjambu-spec-final.pdf.

[151] Wu, H., and Preneel, B. AEGIS: A Fast Authenticated Encryption Algorithm. In *Selected Areas in Cryptography* (2013), vol. 8282 of *Lecture Notes in Computer Science*, Springer, pp. 185–201.

Part II

Papers

Chapter 7

Cryptanalysis of Strong Physically Unclonable Functions

This research is an invited paper at the IEEE Open Journal of the Solid-State Circuits Society (OJ-SSCS) and is currently under revision. A version of this paper was previously submitted, however rejected, from IEEE International Solid- State Circuit Conference 2022 (ISSCC) and Cryptographic Hardware and Embedded Systems (CHES) 2022. This submissions have steered some discussions about the security and design of PUFs, which led to the invitation at OJ-SSCS.

I have worked mostly on the first and second described PUFs. I contributed to the attack ideas, implementations and writing of the attacks on those two algorithms. On the 3rd PUF I took part mostly in discussions and idea exchanges. Section 5 of the paper presenting a parallel between PUFs and symmetric-key cryptography was written by my co-authors.

Cryptanalysis of Strong Physically Unclonable Functions

Liliya Kraleva[1], Mohammad Mahzoun[2], Raluca Posteuca[1], Dilara Toprakhisar[1], Tomer Ashur[1], and Ingrid Verbauwhede[1]

[1] imec-COSIC, KU Leuven, Leuven, Belgium,
[2] Eindhoven University of Technology, The Netherlands
Corresponding author: raluca.posteuca@esat.kuleuven.be

Abstract. Physically Unclonable Functions (PUFs) are being proposed as a low cost alternative to permanently store secret keys or provide device authentication without requiring non-volatile memory, large e-fuses or other dedicated processing steps. In the literature, PUFs are split into two main categories. The so-called strong PUFs are mainly used for authentication purposes, hence also called authentication PUFs. They promise to be lightweight by avoiding extensive digital post-processing and cryptography. The so-called weak PUFs, also called key generation PUFs, can only provide authentication when combined with a cryptographic authentication protocol. Over the years, multiple research results have demonstrated that Strong PUFs can be modeled and attacked by machine learning techniques. Hence, the general assumption is that the security of a strong PUF is solely dependent on its security against machine learning attacks. The goal of this paper is to debunk this myth, by analyzing and breaking three recently published Strong PUFs (Suresh *et al.*, VLSI Circuits 2020; Liu *et al.*, ISSCC 2021; and Jeloka *et al.*, VLSI Circuits 2017). The attacks presented in this paper have practical complexities and use generic symmetric key cryptanalysis techniques.

Keywords: Cascaded PUFs · Cryptanalysis · Physically Unclonable Functions · PUFs · Strong PUF

1 Introduction

Physically Unclonable Functions (PUFs) are the method of choice for hardware applications requiring device authentication. Since securely storing a secret key in an Integrated Circuit (IC) is expensive and simply hard-coding it is vulnerable to physical attacks, PUFs offer a third option: as the manufacturing process of ICs is subject to environmental variances, one can parameterize a cryptographic algorithm by harvesting the resulting randomness.

PUF taxonomy distinguishes between two types of Physically Unclonable Functions, namely Weak- and Strong PUFs. While both types are described in

the literature as *Challenge-Response Protocols*, they differ by the challenge domain's size, i.e., the number of *Challenge Response Pairs* (CRPs). Weak PUFs support a relatively small number of CRPs, while the number of CRPs supported by a Strong PUF is much larger. Thus, Weak PUFs are usually used for storing a (small number of) cryptographic key(s), whereas Strong PUFs are often perceived as a building block in an authentication protocols.

The focus of this paper is on analysing the security of Strong PUFs as a device implementing a random n-to-m function. Broadly speaking, the workings of such a device consists of an n-bit challenge, and an m-bit response; in Strong PUF-literature, typically, $m = 1$. To compute the response, the PUF uses a finite amount of intrinsic randomness harvested from some physical properties of the hardware implementing it (*e.g.* the start-up value of a SRAM or the delay of a multiplexer in case of an arbiter PUF or the frequency of a ring oscillator). Following a sequence of successful attacks using Machine Learning (ML) techniques, the most recent trend is to build Strong PUFs from an IC template cascading non-linear components. The behavior of the non-linear components is determined by the intrinsic randomness mentioned above and the intuition behind this approach is that the cascade amplifies the non-linear effect. It appears from the literature that this approach indeed thwarts machine learning attacks; however, as we show in this paper it is not enough by itself to provide the desired properties.

1.1 PREVIOUS WORK

The first PUF allowing for a large number of CRPs (thus initiating the research line on Strong PUFs) was the *arbiter PUF* introduced in [Jae04]. The arbiter PUF exploits signal delay variations unique to each integrated circuit to parameterize a challenge-response protocol (*i.e.*, assigning a unique behavior to each device). The paper further introduces the adversary objectives and capabilities. It is assumed that the adversary has physical access to the IC and their goal is to clone the PUF. The PUF is considered secure if the adversary is unable to:[3]

- Apply an exhaustive search over the entire CRP space;
- Produce a counterfeit which perfectly simulates the behaviour of the attacked PUF;
- Apply a timing attack based on a measurement of the delays in the attacked strong PUF, followed by a prediction of the outputs;
- Apply non-invasive attacks (*e.g.*, algorithmic attacks); in this case the adversary models the PUF and tries to predict the response associated with a specific challenge with "very high" probability.[4]

A series of strong PUFs were subsequently proposed in the literature, *e.g.*, ring oscillator [SD07], XOR-Arbiter [SD07], or the OT-based PUFs [Rüh10]. At

[3] We stress that these attack scenarios are reproduced from [Jae04] and are therefore informal. We provide a formal discussion in Section 5

[4] The term *very high probability* is used in the original paper. We interpret it to mean better than a random guess.

CCS'10, Rührmair *et al.* [RSS+10] developed a machine learning model attempting to predict the response of previously published strong PUFs. This resulted in breaking all Strong PUFs published until that point.

Other important work in the field of Strong PUFs' analysis is presented by Delvaux et al. in [DGSV14], in which the authors analyse eight Strong PUFs in an integrated framework. This work was then extended in [DPGV15], in which eleven more Strong PUFs were added to the integrated framework, showing numerous security and practicality issues for all the nineteen analysed primitives. Both papers focus on the analysis of Strong PUFs as authentication protocols, underlining that the CRP size of all the analysed Strong PUFs is not suitable for ensuring a sufficient level of security. The conclusion of these two papers is that "proper compensation seems to be in conflict with the lightweight objective" of a Strong PUF.

Moreover, [Del19] presents an analysis of five Arbiter-PUF based authentication protocols, using machine learning techniques, concluding that the use of lightweight obfuscation logic provides insufficient protection against machine-learning attacks for all five analysed primitives.

Consequently, the primary focus in PUF-design has shifted towards explicitly showing resistance to ML-modeling. A recent trend in this direction are *cascaded Strong PUFs* suggested in [VPPK16]. The idea is to use a composition of random sub-functions (*i.e.*, a cascade), conjecturing that the overall non-linear effect thwarts ML-modeling. The three PUFs we investigate in this paper are all of this type. Besides the three primitives addressed in this paper, many more strong PUFs are proposed in the literature, always aiming at increasing machine learning resistance. One such example is the "Double-Arbiter" PUF [MYIS15], which introduces a new type of Strong PUF based on the Arbiter PUF. The aim of this new primitive is to increase the unpredictability of the response, which is measured as the tolerance of the primitive to machine learning attacks. Another example of a cascaded PUF is the LP-PUF introduced in [Wis21]. The LP-PUF consists of three layers, namely an Arbiter layer, a mixing layer and a XOR layer, leading to the generation of 1-bit responses. As the author observes, the structure of the LP-PUF resembles the Substitution-Permutation-Network (SPN), a technique used for the design of block ciphers. The same observation could apply for many cascaded Strong PUFs presented in the literature, leading to a natural inclusion of symmetric-key cryptanalysis techniques in the security evaluation of a Strong PUF. Although ML represents an important technique to asses the security of a primitive, it represents only a first step in the security analysis. ML is a black box approach and it does not take into account the structure of a Strong PUF. In this work we go beyond this approach, by using tools from symmetric-key cryptanalysis and by taking insights from the description of the primitive. More precisely, in this paper we choose to analyse three representative strong PUFs (Suresh *et al.* [SKA+20], VLSI Circuits 2020; Liu *et al.* [LFL+21], ISSCC 2021; and Jeloka *et al.* [JYO+17], VLSI Circuits 2017) because they have nice CMOS circuit implementations.

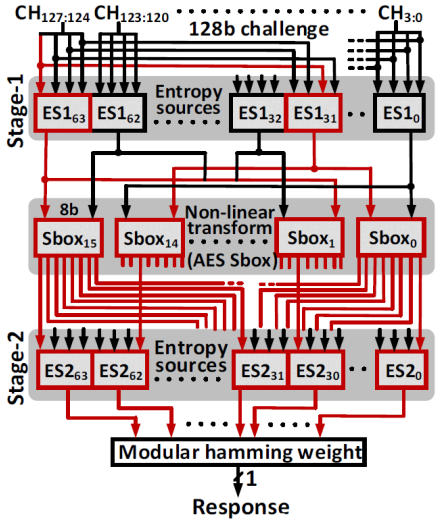

Fig. 1. A schematic description of the PUF. The figure is copied without any modifications from [SKA+20] and we use it under the provisions of fair use.

1.2 OUR CONTRIBUTION

This paper serves as a tutorial for the design of secure PUFs from a symmetric-key point of view. We motivate the need for such a tutorial by investigating—from an algorithmic point of view—three recently published cascaded Strong PUFs and show that they all exhibit undesirable properties undermining their security.

We stress that our aim with this paper is to provide general guidelines to the design of secure algorithms and that we did not attempt to provide a thorough cryptanalysis of the three algorithms. Indeed, more powerful attacks probably exist, and directly fixing the issues we raise is unlikely to result in secure devices.

2 SURESH *ET AL.*

In this section we present details of the strong PUF proposed by Suresh *et al.* in [SKA+20]. We then show that the cascaded nature of this algorithm collapses the 2^{128} CRP domain into a much smaller subspace of equivalence classes.

2.1 DESCRIPTION

Suresh et al. [SKA+20] offer a cascaded algorithm in 14nm CMOS with a claimed $10^{28} \approx 2^{93}$ CRP space.It takes a 128-bit challenge and returns a 1-bit response. The algorithm is abstracted into three layers as depicted in Figure 1.

The first layer consists of 64 random 4-to-1 functions (denoted ES1 boxes in the original paper). It takes a 256-bit input and returns a 64-bit output. The second layer involves 16 AES Sboxes, each taking an 8-bit input and returning an 8-bit output. The third layer again consists of 64 random 4-to-1 functions (denoted ES2 boxes in the original paper). Finally, the 1-bit response is taken as the parity of the third layer's output.

The original paper also refers to the first and third layers as Stage 1 and Stage 2, respectively. In lieu of a complete specification, which was not provided in [SKA+20], we proceed by assuming that the bit permutations are according to the wirings illustrated in Figure 1.

2.2 INTUITION

According to [SKA+20], the PUF offers a challenge space in the range 10^{28}–$10^{31} \approx 2^{93}$–2^{103}. As this is the only quantifiable claim, we understand it to be the advertised security. Since the response of the PUF consists of only a single bit, predicting the output with probability better than $\frac{1}{2}$ amounts to a successful attack.

Our first observation is that the first layer is an entropy choke point, *i.e.*, no matter how much randomness was invested in it, it cannot cascade more than 64 bits of entropy to the rest of the device. We say that two inputs are in the same *collision class* if they result in the same output after the first layer (Stage 1). Such two values will result in the same computation throughout the rest of the device and subsequently the same response for both inputs.

Our second observation is that each pair $(ES1_{32+n}, ES1_n)$ of ES1 functions is isolated from all other ES1 functions. Therefore, we consider an alternative representation where the first layer consists of 32 functions each mapping a 4-bit input to a 2-bit output. Then the output values are 00, 01, 10, and 11 and they induce four equivalence classes each containing four values on average.

2.3 RECOVERING THE EQUIVALENCE CLASSES

By using the two observations presented in the previous section we show how $2^{11.8} \approx 10^{3.55}$ queries are enough to group the 128-bit input into 2^{64} sets. Each of these sets contains on average 2^{64} values all resulting in the same response. Thus, learning the response to one challenge leaks the response to all the other $2^{64} - 1$ values from the same equivalence class.

Without loss of generality, we consider the equivalence classes of the pair $(ES1_{63}, ES1_{31})$. The adversary fixes the last 124 bits of the input to an arbitrary value and iterates the first four. Two sets S^0 and S^1 are initialized and each 4-bit input is added to the set S^i if and only if the response is i. However, those are not yet the desired equivalence classes. Having the same response for two different plaintexts can be caused by any of the following three reasons:

- a collision after Stage 1;
- a collision after Stage 2, without a collision after Stage 1;

– the same Hamming weight after Stage 2, regardless of the state after Stage 1 or Stage 2.

The last two cases are *false positives* that need to be filtered out. In order to do so, the last 124 bits are fixed to a different arbitrary value.

Note that the elements in the same class will always be mapped together to one of the sets (but not necessarily the same one for different values of the last 124 bits). The equivalence classes are then computed by identifying which values are always mapped together to the same set. The probability of recovering the correct equivalence classes is approximately 99% when the process is repeated 7 times. For comparison, even three repetitions result in a success probability of 65% to identify the right set. The probability was computed empirically, using Monte Carlo simulations.

The complexity of determining an equivalence class for a single pair of ES1 functions with 99% success rate is $7 \cdot 16 = 2^{6.8} \approx 10^{2.05}$chosen queries. This is repeated for each pair of ES1 functions independently, leading to an overall complexity of $32 \times 2^{6.8} = 2^{11.8} = 10^{3.55}$ chosen challenges for recovering all pair-equivalence classes with expected success probability of $(1 - 0.01)^{32} = 0.73$ (or 73%).

The next step after recovering all pair-equivalence classes is to link them into state-equivalence classes. We define a state equivalence class as a set of challenges for which the state after Stage 1 is equal, therefore leading to the same response for both inputs.

Since each pair-equivalence class has an average of four elements, the average number of elements in one state-equivalence class is $4^{32} = 2^{64} \approx 10^{19.27}$. Additionally, the state after Stage 1 is 64-bit long, leading to $2^{64} \approx 10^{19.27}$ different state-equivalence classes.

At this point, learning the response to any challenge leaks the response to all other challenges within the same state-equivalence class. Note that the classes are built such that on observing a new challenge it requires negligible effort to identify which state-equivalence class it belongs to.

2.4 DISCUSSION

Our approach exploits the fact that the input space collapses from 2^{128} to 2^{64} after Stage 1. We see that by investing a relatively small amount of effort (the analysis of $2^{11.8}$ chosen challenges) an adversary can learn the outputs of Stage 1 for any new challenge, effectively removing the first layer of random functions. Moreover, if the adversary learns the response associated to a challenge, then they also know that the same response is associated to all the challenges in the same state-equivalence class with the initial one. This is an undesirable behavior; or alternatively, in case this is an acceptable behavior, it can be achieved using cheaper components.

Note that we did not investigate the properties of the other layers, and it is likely that this basic attack can be improved further.

3 LIU *ET AL.*

The second design we analyse is [LFL+21] due to Liu et al. We show that this device is vulnerable to two generic attacks. The first attack can be applied if the the device exhibits an inherent bias, which can easily be detected and exploited. The second attack is independent on the bias of the device, and allows the adversary to guess the responses associated to a group of well chosen challenges.

3.1 DESCRIPTION

Liu et al. propose a cascade of 5-to-5-bit random functions formed in two layers of five with a bit-permutation between them, resulting in a 25-to-25-bit function which we denote by $S(x)$; see Phase 1 in Figure 2. $S(x)$ is used to digest the 100-bit input by first splitting it into four 25-bit blocks, then consuming the blocks iteratively with a feed-backward operation from each block to the next one; see Phase 2 in Figure 2.

A finalization function, which we denote by $C(x)$ is used to compute the response. First, a sequence of 5-to-1-bit random functions is applied to the output coming from the last call to $S(x)$, resulting in a 5-bit output. Then, the parity of these five bits is returned as the response; see Phase 3 in Figure 2.

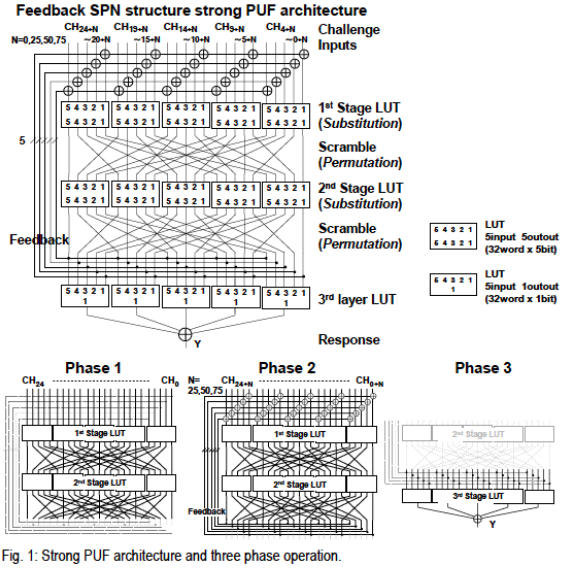

Fig. 2. A schematic description of the PUF. The figure is copied without any modifications from [LFL+21] and we use it under the provisions of fair use.

Formally, denote by $x = (x_0, x_1, x_2, x_3)$ the 100-bit challenge, where x_i is the i-th part of length 25. Then,

$$\mathrm{PUF}(x) = C(S(S(S(S(x_0) \oplus x_1) \oplus x_2) \oplus x_3)) \tag{1}$$

is a succinct description of the function.

As before, we assume that the adversary does not have access to the description of the internal functions. Again the adversary seeks to predict the 1-bit response associated to a newly seen challenge with a probability better than $\frac{1}{2}$.

3.2 INTUITION

From the formal description in Equation (1) arises a natural observation:

Observation 1. Fix the first 75 bits of the challenge, namely x_0, x_1 and, x_2; then, Equation (1) is reduced to

$$PUF(x) = C(S(c \oplus x_3)), \tag{2}$$

where $c = S(S(S(x_0) \oplus x_1) \oplus x_2)$ is fixed but unknown.

For brevity, we define an auxiliary function

$$f(x) = C(S(x)). \tag{3}$$

We underline that $PUF(x) = f(x \oplus c)$, where $c = S(S(S(x_0) \oplus x_1) \oplus x_2)$.

Definition 1. *We define a map of a function f as the ordered set $M_f = \{(x, f(x)), \forall x\}$, where the order is defined as follows:*

$$(x_1, f(x_1)) < (x_2, f(x_2)) \Leftrightarrow x_1 < x_2.$$

Note that the map M_f was constructed such that the input before the last application of the S function takes all possible values. In particular, all the maps have the same elements, but arranged in a different order. This property is formally described in the following general observation:

Observation 2. Let x_0, x_1, and x_2 be randomly chosen and fixed. Then, the map $M_f^c = \{(x \oplus c, f(x \oplus c), \forall x\}$ is an affine translation by $c = S(S(S(x_0) \oplus x_1) \oplus x_2)$ of the map M_f.

By constructing one arbitrary M_f in full, the adversary learns its distribution, *i.e.* the number of 0 or 1 responses. Due to Observation 2, any map M_f^c has the same distribution as M_f. If the device exhibits an inherent bias, then the distribution of M_f can be trivially used by the adversary to predict the output to any challenge not in M_f with probability better than $\frac{1}{2}$. For example, if the number of 0 responses associated to M_f is $2^{25} - 2^{20}$, then the probability of having a 0 response to an arbitrary challenge is $\Pr = \frac{2^{25}-2^{20}}{2^{25}} = 1 - 2^{-5} = 0.96$ (96%).

3.3 ATTACK DESCRIPTION

To construct M_f the adversary first chooses an arbitrary 75-bit value which they use to fix $x_0||x_1||x_2$. Then, iterating over the remaining 25 bits, the adversary queries the PUF $2^{25} \approx 10^{7.53}$ times and records the responses in M_f.

Constructing M_f requires $2^{25} \approx 10^{7.53}$ chosen challenges. The memory complexity for storing M_f can be optimized to 2^{22} bytes, *i.e.*, 4.19MB by querying the challenges in a natural order and indexing the responses accordingly.

Fixing M_f as a reference system, and recalling Observation 2, we see that determining c is sufficient for translating CRPs from M_f to M_f^c. To do so, the adversary observes 25 CRPs from the target equivalence class (*i.e.*, these 25 CRPs share the same value in the first 75 bits). Then, by means of exhaustive search on c, the adversary filters candidates where $M_f[i \oplus c] \neq M_f^c[i]$ using the 25 queries. This exhaustive search could be viewed as solving a system of equations with 25 bit unknown values. Therefore, the minimum number of equations such that this system is independent is 25. Our experiments show that 25 CRPs are enough for the correct c to be the only surviving candidate with high probability. At this point the adversary can determine with full certainty that $M_f^c[i] = M_f[i \oplus c]$ for all values of i. This way the adversary can learn the responses to all the remaining challenges from M_f^c.

3.4 DISCUSSION

Note that the fact that the device is biased is not a problem in itself. Daemen and Rijmen analyzed in [DR07] the bias of ideal m-to-n-bit functions and showed that they are approximately normal distribution with mean 0 and variance 2^{-n}. For an ideal 100-to-1-bit function, such a bias would not be detectable. However, what Observation 2 shows is that this device actually models a random 25-to-1-bit function (in the best case scenario); making it significantly easier to detect the bias.

A second observation that we did not pursue in this paper is that the cascade structure will amplify the bias introduced by the random 5-to-5-bit functions. Each of these functions is an entropy choke point in itself and the cascade will result in bias that is even larger than what is predicted by [DR07] for the ideal case.

Moreover, we see again that by investing a relatively small amount of effort (the processing of $2^{25} + 25$ chosen challenges and an exhaustive search over a space of 2^{25}) an adversary can predict the response to unknown queries with high probability. This is again an undesirable behavior; or alternatively, in case this is an acceptable behavior, it can be achieved using cheaper components.

Note that we did not investigate the properties of the other layers, nor the ones of the component functions, and it is likely that this basic attack can be improved further.

4 JELOKA *ET AL.*

The third design we analyse was introduced by Jeloka *et al.* in [JYO+17]. We show that the responses of this device preserve input correlations with high probability.

4.1 DESCRIPTION

Jeloka *et al.* propose an SRAM-based PUF with a claimed CRP space that grows exponentially in the number of rows and the challenge length. The device has a secret initial state $S \in \mathbb{F}_2^{n\times m}$ and a secret matrix of powers $P \in \mathbb{Z}_n^{n\times m}$. Each column of P is viewed as a permutation on the integers $\{0, 1, \ldots, n-1\}$. Larger numbers are associated with "more power". A challenge $C = \{r_1, \cdots, r_t\} \subseteq \{0, 1, \ldots, n-1\}$ is defined as a sequence of rows, where each two consecutive rows (r_i, r_{i+1}) "fight" and the cell with bigger associated power wins the fight and overwrites its value over that of the other cell. The fight takes place independently for each column. More precisely, in a fight between row i and row j, for each column k, if $P_{i,k} > P_{j,k}$ then $S_{j,k} \leftarrow S_{i,k}$; otherwise $S_{i,k} \leftarrow S_{j,k}$. The corresponding response to a challenge C denoted by $F(C)$, represents the m bits of the last row in the challenge.

[JYO+17] describes the PUF in an abstract way for any dimensions (n, m) and challenge length t (see Algorithm 1). The authors suggest that $n = m = 64$ and $t = 6$ are sufficient to attain some unspecified level of security.

Algorithm 1: Psuedo-code for the PUF from [JYO+17]

```
Input: C = {r_1, ..., r_t}
Output: F(C) := T_{r_t}
T ← S
for (r_i, r_{i+1}) ∈ C do
    for 0 ≤ j ≤ m − 1 do
        if P_{r_i,j} > P_{r_{i+1},j} then
            T_{r_{i+1},j} ← T_{r_i,j}
        else
            T_{r_i,j} ← T_{r_{i+1},j}
        end
    end
end
Return T_{r_t}
```

4.2 INTUITION

We show how two challenges that are "similar" will result in the same response with high probability. Concretely, we show that two challenges (sequences) dif-

fering only in their first value will produce the same 64-bit response with probability $2^{-0.77}$. Note that since the response is 64-bit long, one would expect this probability to be 2^{-64}.

4.3 ATTACK DESCRIPTION

First, we analyze the state S with a single column. In the case where $m = 1$, the response $F(C)$ to the challenge C is a 1-bit value $F(C) = b$.

Observation 3. Let $\bar{X} = (x_1, x_2, x_3, x_4)$ and $F(a, \bar{X})$ be the result of the fight between the rows a, x_1, x_2, x_3, x_4 (a, is the first row of the fight and the output is x_4). Let $G(\bar{X})$ be the result of the fight without considering the first row a. Then

$$F(a, \bar{X}) \neq G(\bar{X}) \iff F(a, \bar{X}) = S_a \text{ and } G(\bar{X}) \neq S_a$$

And the probability that a changes the response is

$$\Pr\left[F(a, \bar{X}) \neq G(\bar{X})\right] = \frac{1}{2} \cdot \frac{1}{5!} = \frac{1}{240}$$

The only case in which the result of two functions is different is when the response is S_a and the powers have the form $P_a > P_{x_1} > P_{x_2} > P_{x_3} > P_{x_4}$ with $p = \frac{1}{120}$ and $F(\bar{X}) \neq S_a$ with $p = \frac{1}{2}$.

Observation 3 shows that the first row in the challenge has small influence on the final response. Therefore, two challenges that only differ in the first row have different responses with the following probability

$$\Pr\left[F(a, \bar{X}) \neq F(b, \bar{X}) = \frac{1}{120}\right]$$

The cases that $F(a, \bar{X}) \neq F(b, \bar{X})$ are as follows:

1. $F(a, \bar{X}) = S_a$ and $F(b, \bar{X}) \neq S_a$ with probability $\frac{1}{240}$.
2. $F(a, \bar{X}) = F(\bar{X})$ and b is the strongest: $F(b, \bar{X}) \neq F(\bar{X})$ with probability $\frac{1}{240}$.

So the total probability that both queries give the same result in a single column is $1 - \frac{1}{120} = \frac{119}{120}$. Since the powers in different columns are independent, the probability of a correct guess for $m = 64$ columns is $\frac{119}{120}^{64} = 2^{-0.77}$.

4.4 DISCUSSION

In this case we see that the PUF does not offer good diffusion properties and that each query allows to predict with high probability the response to multiple other queries. This is an undesirable property as one can expect from an authentication device to produce an uncorrelated response even for correlated entries.

For brevity, we did not model the case where the two challenges differ in positions other than the first one or when they differ in more than one position. As a general observation we offer that the probability for collision is higher when the change occurs in earlier positions. This violates Jeloka *et al.*'s claim that longer sequences result in better security.

5 STRONG PUFS WHEN VIEWED AS SYMMETRIC-KEY ALGORITHMS

The work presented in this paper highlights a gap between two communities concerned with the development and implementation of secure cryptographic algorithms. The disparity is reflected in different areas such as design strategies, security analysis, and even in the way a new algorithm is presented to a larger audience. We discuss some of these differences, hoping to initiate a larger discussion and exchange of ideas between the two communities.

5.1 ABSTRACTION LEVEL

The design process resulting in a secure device involves several levels of abstraction. In the context of this paper, we focus on three of those:

- *Mathematical level.* In this abstraction level, the mathematical properties of abstract classes of functions are investigated. This kind of work involves for example methods from probability theory, combinatorics, statistics, algebra (both linear and modern), Fourier analysis, complexity theory, etc., and is normally published in pure mathematics- or mathematically-oriented cryptography- venues;
- *Algorithmic level.* At this level, the abstract functions are used as conceptual building blocks to form an algorithmic model. In addition to understanding the properties of the building blocks, the designer must also understand how they interact when combined together (for security reasons), and the platform that they will be running on (for efficiency reasons). In addition to the algorithm description itself, the algorithmic model includes a well-defined adversarial model and security claims pertaining to it. This is at the core of what cryptographers do, and a standard practice is to submit such works to cryptographic venues for peer-review and 3rd party evaluation;
- *Implementation level.* Having completed the vetting process of the algorithmic model, the algorithm is implemented first in simulation and later in tangible form. Even assuming the ideal security of the algorithm, the implementation process itself is susceptible to issues undermining the device's security (*e.g.*, timing attacks, side-channels). It is therefore not enough to understand the efficiency metrics and one must also understand the idiosyncrasies of secure implementations.

Strong PUFs aim to achieve goals on the implementation level (*e.g.*, secure key storage). The PUFs we looked into stem from a deep understanding of the execution platform, namely IC, and are therefore very efficient. However, the mathematical and algorithmic levels have been systematically overlooked. This is why Rührmair et al. and Delvaux et al. were able to use machine learning attacks in [RSS+10, Del19] and why this paper can attack more recent works.

5.2 ADVERSARIAL MODELS

To be able to speak of the security offered by an algorithm, the conservation must obtain a shared understanding of what "security" is. As the notion of security only makes sense with respect to the existence of a bad actor (*i.e.*, an adversary), it makes sense to start the discussion there. An inherent imbalance between defenders and attackers (designer and adversaries, respectively) is that attack methods are plentiful, and it is enough for one of them to succeed for the defender to fail their role. Since attackers are creative and resourceful, it is futile to attempt predicting what methods they will use. Instead, cryptographers work with adversarial models. An adversarial model presumes only the capabilities of the adversary, but not how the adversary will use these capabilities.

Depending on the capabilities, an adversary can be classified as either *passive* or *active*. Whereas a passive adversary is capable of only observing the communication channel, an active adversary is additionally able to delete, add, and alter the data sent over the channel. In the context of PUF design, we identified the following relevant models from [MvOV96, KL07]:

- **Passive adversaries**
  - *Ciphertext-only attack:* the adversary can only observe the outputs of the system; thus, the outputs of a secure cryptosystem should provide no information regarding the corresponding inputs, or the secret key/randomness; this model is the easiest to carry out in practice, since the only requirement is passively eavesdropping the communication channel.
  - *Known-plaintext attack:* the adversary is in possession of some inputs and the corresponding outputs generated under the same secret key.
- **Active adversaries**
  - *Chosen-plaintext attack:* the adversary is capable of obtaining the outputs corresponding to inputs of the adversary's choice.
  - *Adaptive chosen-plaintext attack:* the adversary may select the inputs depending on the received outputs from the previous requests.

Since in many cases the input or parts thereof are public (*e.g.*, HTTP headers) known plaintext attacks are often regarded practical. Furthermore, in 2-party authentication protocols such as the ones considered for strong PUFs, both active models also appear reasonably feasible.

5.3 ALGORITHMIC DESCRIPTION

Symmetric-key algorithms normally define the input, output, and key spaces, and an algorithmic description to produce the corresponding output given the input and the secret key. A strong PUF can be modeled in the same way by considering the challenge as the input, the PUF's response as the output, and the intrinsic randomness as the key.

A fundamental assumption in cryptography is the reputed *Kerckhoffs' principle*, which states that "a cryptosystem should be secure even if everything about

the system, except the key, is public knowledge". In the algorithmic description, the key is seen as part of the input that is unknown to the adversary. It follows from Kerckhoffs' principle that the algorithmic description can be provided independent of the key (just like an IC can be manufactured irrespective of the values it will receive through its input wires).

The dissemination of new algorithms is a sort of "conversation" between the designers and their audience. To support this conversation, the designers provide a *design rationale* motivating their decisions. A reference implementation, and/or test vectors are provided to alleviate any ambiguity in the algorithmic description. If additional data was generated or used by the designers in the design process, it is also provided for examination and reproducibility.

5.4 SECURITY CLAIMS

When the key is modeled as an additional input, it becomes apparent that any algorithm using a finite number of secret bits is vulnerable to *brute force attacks* since it is always possible (but not necessarily feasible) to exhaustively iterate the key space. From the observation that the key size provides an upper bound on the effort required for attacking the algorithm arises an intuitive definition to what constitutes an attack.

Definition 2. *A cryptosystem is said to be broken if an adversary can achieve their goals with less effort that would be required if they used a brute force attack.*

Translating this to the case of Strong PUFs, let $\mathcal{C}$ be an arbitrary[5] n-bit challenge, $f(\mathcal{C})$ its 1-bit response, and $\mathcal{Q}$ the set challenges that have already been made and whose responses are known to the adversary. A reasonable security claim would be that if $\mathcal{C} \notin \mathcal{Q}$, no adversary can predict $f(\mathcal{C})$ with probability better than $\frac{1}{2}$, even when $\mathcal{C}$ is chosen after observing all the responses to the challenges in $\mathcal{Q}$ (adaptive chosen challenge); Formally,

$$\Pr[f(\mathcal{C})|\mathcal{C}, Q] = \frac{1}{2}. \tag{4}$$

Another way to view this security claim is through the notion of advantage: for any n-bit uniformly random challenge $\mathcal{C}$,

$$\mathbf{ADV} = \frac{1}{2} + \frac{|Q|}{2^n}. \tag{5}$$

[5] We note the terms *arbitrary* and *random* are not interchangeable in cryptography. A value is said to be *random* if it is sampled rigorously from a given distribution, usually the uniform one; it is said to be *arbitrary* when there is no importance to how it was sampled. For example, one's birthday is an arbitrary value, but for an encryption algorithm to be secure, the key must be chosen randomly from the uniform distribution of k-bit vectors.

Equation (5) captures the intuitive notion that the only way for an adversary to gain any knowledge is by querying device on that specific challenge.[6]

With an adversarial model, an algorithmic description, and a clear notion of security, the designer can provide *security claims*. Such security claim for the PUF presented in Section 2 can take the following form "*the Strong PUF can resist any chosen plaintext attack that runs in time less than* 2^{50} *time and requires less than* 2^{93} *chosen queries*". If an attacker generates all the state-equivalence classes, i.e. requiring 2^{64} queries, they can guess the response associated to any future challenge with probability 1. Therefore, such an attack which uses fewer queries and running time than what is permissible, is interpreted as breaking the algorithm.

A subtle point that we have seen overlooked is that in addition to being correct, the security claim must also be sensible. For example, since a PUF does not have a way to verify that a challenge has been received from a valid server (rather than from an adversary), it does not make sense to ignore chosen and adaptive-chosen challenge attacks. Likewise, our observations in this paper do not invalidate the claims about resistance to ML-attacks, yet these PUFs are not secure and should not be deployed in field settings.

Finally, we note that in light of the Strong PUFs we found in the literature, it does not make sense to consider the complexity of a brute-force attack as a function in the secret/random material. As the amount of randomness used is much larger than the CRP-space, the adversary can clone the PUF trivially by querying it completely. Thus Equations 4–5 are the more natural choice.

5.5 CONCLUDING REMARKS ON SECURITY-EFFICIENCY TRADE-OFFS

Surveying the recent literature on Strong PUFs, we noticed that the general trend in designing them is to employ a series of random-based operations, such as random Boolean functions, Sboxes, or linear layers. While the need for efficient algorithms is understandable, the security/efficiency trade-off must be handled carefully, as operations resulting in an insecure mechanism are by definition inefficient. Moreover, in symmetric-key cryptography, two of the most important properties that are analysed in a new design are the confusion and the diffusion ensured by the component functions of a cipher. These aspects are covered in a series of books such as [DR20, KL14, KR11].

In the PUFs we surveyed in this paper we see that the adversarial model is not stated, and instead, heuristic are used to assess the device's security. Among symmetric-key cryptographers, this approach is considered obsolete. Modern techniques for the design of symmetric-key algorithms build on over 50 years of research in this domain to offer well understood trade-offs between efficiency

[6] Cryptographers sometimes use the word *leakage* to describe "the knowledge an adversary may gain". However, this term is already loaded with meaning in the electrical engineering community, hence we omit it to avoid confusion.

and security. In other words, it is unlikely that a casual, non-systematic approach would yield a secure algorithm (regardless of its efficiency).[7].

We refer readers interested in understanding the state of the art in lightweight cryptography to the Wiki maintained by the cryptography group in the University of Luxembourg [L] noting that after more than a decade of research into this area it is unlikely that the state-of-the-art can be significantly improved without a paradigm shift. Interestingly, we observe that the amount of randomness exploited in PUF designs far exceeds what is common in symmetric-key cryptography. Whereas contemporary symmetric-key primitives have key sizes ranging between 80–256 bits, the algorithms we surveyed above use randomness that is measured in the order of thousands of bits. More randomness is usually associated with better security through an increase in the key size. It would be interesting to explore in future work if a different trade-off can be obtained by fixing the security level and somehow exploiting the additional randomness to improve efficiency.

6 CONCLUSION

PUF design share many common characteristics with the design of symmetric-key cryptographic algorithms. Motivated by undesirable properties we found in three recently published Strong PUFs, we attempted to provide in this paper a tutorial to the approach taken by symmetric-key researchers to ensure the security of their algorithms. We hope that this paper would serve as a starting point for discussion between the two communities.

7 ACKNOWLEDGEMENTS

This research is partially supported by CyberSecurity Research Flanders with reference number VR20192203 and by the Research Council of the KU Leuven C16/15/058. Tomer Ashur is an FWO post-doctoral fellow under Grant Number 12ZH420N. Liliya Kraleva is a PhD fellow by Research Foundation Flanders (FWO). This project was partially supported by Intel through the Intel Project on Cryptographic Frontiers.

[7] For a version of this message, see Bruce Schneier's blogpost `https://www.schneier.com/blog/archives/2015/05/amateurs_produc.html`

Bibliography

[Del19] Jeroen Delvaux. Machine-Learning Attacks on PolyPUFs, OB-PUFs, RPUFs, LHS-PUFs, and PUF–FSMs. *IEEE Transactions on Information Forensics and Security*, 14(8):2043–2058, 2019.

[DGSV14] Jeroen Delvaux, Dawu Gu, Dries Schellekens, and Ingrid Verbauwhede. Secure lightweight entity authentication with strong pufs: Mission impossible? In Lejla Batina and Matthew Robshaw, editors, *Cryptographic Hardware and Embedded Systems - CHES 2014 - 16th International Workshop, Busan, South Korea, September 23-26, 2014. Proceedings*, volume 8731 of *Lecture Notes in Computer Science*, pages 451–475. Springer, 2014.

[DPGV15] Jeroen Delvaux, Roel Peeters, Dawu Gu, and Ingrid Verbauwhede. A Survey on Lightweight Entity Authentication with Strong PUFs. *ACM Comput. Surv.*, 48(2):26:1–26:42, 2015.

[DR07] Joan Daemen and Vincent Rijmen. Probability distributions of correlation and differentials in block ciphers. *J. Math. Cryptol.*, 1(3):221–242, 2007.

[DR20] Joan Daemen and Vincent Rijmen. *The Design of Rijndael - The Advanced Encryption Standard (AES), Second Edition.* Information Security and Cryptography. Springer, 2020.

[Jae04] Jae W. Lee, Daihyun Lim, Blaise Gassend, G. Edward Suh, Marten van Dijk, and Srinivas Devadas. A technique to build a secret key in integrated circuits for identification and authentication applications. In *2004 Symposium on VLSI Circuits. Digest of Technical Papers (IEEE Cat. No.04CH37525)*, pages 176–179, 2004.

[JYO+17] Supreet Jeloka, Kaiyuan Yang, Michael Orshansky, Dennis Sylvester, and David Blaauw. A sequence dependent challenge-response PUF using 28nm SRAM 6T bit cell. In *2017 Symposium on VLSI Circuits*, pages C270–C271, 2017.

[KL07] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography.* Chapman and Hall/CRC Press, 2007.

[KL14] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography.* CRC Press, 2014.

[KR11] Lars Knudsen and Matthew Robshaw. *The Block Cipher Companion.* 01 2011.

[L] Lightweight Block Ciphers. `https://www.cryptolux.org/index.php/Lightweight_Block_Ciphers`.

[LFL+21] Kunyang Liu, Zihan Fu, Gen Li, Hongliang Pu, Zhibo Guan, Xingyu Want, Xinpeng Chen, and Hirofumi Shinohara. 36.3 A Modeling Attack Resilient Strong PUF with Feedback-SPN Structure Having $< 0.73\%$ Bit Error Rate Through In-Cell Hot-Carrier Injection Burn-In. In *2021 IEEE International Solid- State Circuits Conference (ISSCC)*, volume 64, pages 502–504, 2021.

[MvOV96] Alfred Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1996.

[MYIS15] Takanori Machida, Dai Yamamoto, Mitsugu Iwamoto, and Kazuo Sakiyama. A New Arbiter PUF for Enhancing Unpredictability on FPGA. *The Scientific World Journal*, 2015, 2015.

[RSS+10] Ulrich Rührmair, Frank Sehnke, Jan Sölter, Gideon Dror, Srinivas Devadas, and Jürgen Schmidhuber. Modeling Attacks on Physical Unclonable Functions. CCS '10, page 237–249, New York, NY, USA, 2010. Association for Computing Machinery.

[Rüh10] Ulrich Rührmair. Oblivious transfer based on physical unclonable functions. In Alessandro Acquisti, Sean W. Smith, and Ahmad-Reza Sadeghi, editors, *Trust and Trustworthy Computing*, pages 430–440, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[SD07] G. Edward Suh and Srinivas Devadas. Physical Unclonable Functions for Device Authentication and Secret Key Generation. In *2007 44th ACM/IEEE Design Automation Conference*, pages 9–14, 2007.

[SKA+20] Vikram B. Suresh, Raghavan Kumar, Mark Anders, Himanshu Kaul, Vivek De, and Sanu Mathew. A 0.26% BER, 10^{28} Challenge-Response Machine-Learning Resistant Strong-PUF in 14nm CMOS Featuring Stability-Aware Adversarial Challenge Selection. In *IEEE Symposium on VLSI Circuits, VLSI Circuits 2020, Honolulu, HI, USA, June 16-19, 2020*, pages 1–2. IEEE, 2020.

[VPPK16] Arunkumar Vijayakumar, Vinay C. Patil, Charles B. Prado, and Sandip Kundu. Machine learning resistant strong PUF: Possible or a pipe dream? In *2016 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pages 19–24, 2016.

[Wis21] Nils Wisiol. Towards attack resilient arbiter puf-based strong pufs. Cryptology ePrint Archive, Paper 2021/1004, 2021.

Chapter 8

Differential Cryptanalysis of K-Cipher

This paper was published at the 27th IEEE Symposium on Computers and Communications (IEEE ISCC 2022), where my co-author Mohammad Mahzoun presented the results.

I have contributed to all aspects of this paper. More particularly, I worked on the development of the attack, generation of differential characteristics and writing of the paper. Large part of the attack implementations were done by Mohammad Mahzoun.

Differential Cryptanalysis of K-Cipher ⋆

Mohammad Mahzoun[2], Liliya Kraleva[1], and Raluca Posteuca[1] and Tomer Ashur[1,2]

[1] imec-COSIC, KU Leuven, Leuven, Belgium,
[2] Eindhoven University of Technology, The Netherlands
m.mahzoun@tue.nl, liliya.kraleva@esat.kuleuven.be
raluca.posteuca@esat.kuleuven.be, tomer.ashur@esat.kuleuven.be

Abstract. K-Cipher is an ultra low latency block cipher with variable-length parameters designed by Intel Labs. In this work, we analyze the security of K-Cipher and propose a differential cryptanalysis attack with the complexity of $2^{29.7}$ for a variant of K-Cipher with state size $n = 24$ bits state and block size $m = 8$ bits. Our attack recovers the secret key and secret randomizer values with a total length of 240 bits in $\sim$ 30 minutes on a standard desktop machine. We show that it is possible to extend the same attack for an arbitrary set of parameters.

1 Introduction

K-Cipher [KDGD20a] is a low latency parametrizable block cipher proposed by researchers from Intel Labs at IEEE ISCC 2020. The motivation for designing K-Cipher is that current low latency ciphers such as PRINCE [BCG+12], PRINCEv2 [BEK+20], MANTIS [BJK+16], QARMA [Ava17] and SPEEDY [LMMR21], as well as the proposed ciphers at the NIST Lightweight competition, do not offer a wide range of values for the state size and block size or are not efficient enough. Despite the fact that it is published recently, K-Cipher attracted a lot of attention and is being used in other works because of its resistance to known ciphertext attacks, its small block size, and extremely low latency. An instance of such works is a new memory safety mechanism called C^3 [LRD+21] which is proposed in 2021 and uses K-Cipher as its underlying block cipher to encrypt the pointers to the memory blocks.

In this work, we analyze the security of K-Cipher against differential cryptanalysis - an attack that the authors claim the cipher is secure against. We propose a full key-recovery attack on the 24-bit input with the 8-bit SBox version and the 128-bit input and 16-bit SBox version, as well as an evaluation of some other possible parameters. Our attack is a chosen-plaintext attack and

⋆ This work was supported by CyberSecurity Research Flanders with reference number VR20192203. In addition it was supported by Intel through the Intel Project on Cryptographic Frontiers. Liliya Kraleva is a supported by a Ph.D. Fellowship from the Research Foundation — Flanders (FWO). Tomer Ashur is an FWO post-doctoral fellow under Grant Number 12ZH420N.

needs the corresponding encryption of 2^{14} pairs of chosen plaintexts. The complexity of our attack is $2^{29.7}$ encryptions/decryptions for all variants of K-Cipher with block size $m = 8$ and can fully recover all the secret values. We run our attack on a standard desktop machine and recover all 240 secret bits in ~ 30 minutes.

The paper is structured as follows. In Section 2, we introduce notations, differential cryptanalysis, and the description of K-Cipher. In Section 3 we describe the attack. We first explain in Section 3.1 a technique to construct characteristics independently of the random values added in the SBox layer. Further, in Section 3.2 we extend the characteristics to a key-recovery attack and recover the secret data. In Section 3.3 we analyze the complexity of our attack. Finally, we write our conclusion in Section 4.

2 Preliminaries

2.1 Notation

We denote a binary string B with length n by $B = b_{n-1}b_{n-2}\dots b_0$ where each $b_i \in \{0,1\}$. Binary strings are used to describe the states of the K-Cipher. Each state B is the concatenation of several consecutive blocks $B[i]_{1\leq i\leq \frac{n}{m}}$ of size m. In other words, $B = B[1]\|B[2]\|\dots\|B[m] = b_{n-1}b_{n-2}\dots b_0$. If the state size n is not the multiple of the block size m, the size of the last block is equal to the $n \mod m$. The SBoxes of K-Cipher are indexed from left to right as $S[1], S[2], \dots, S[m]$ and the input of $S[i]$ is the block $B[i]$. A binary sequence with zeros in all positions except for the i_{th} position is denoted by $e_i = [000\dots1\dots000]$.

In the rest of this paper, P is the input, C is the output, and $\mathcal{X}_i$'s are the intermediate states. The binary operations $\lll$ and $\ggg$ are used to show circular left shift and circular right shift. We define $+_m$ as block-wise modular addition over $GF(2^m)$. For two binary strings B_1 and B_2 with length km, $B_1 +_m B_2$ divides both B_1 and B_2 into k blocks of size m, then adds corresponding blocks modulo 2^m, and concatenates the resulting blocks in the same order. We define $-_m$ the same way as $+_m$ but for subtraction.

2.2 Differential cryptanalysis

Differential cryptanalysis is one of the most powerful and used cryptanalysis techniques against symmetric primitives. It is proposed by Biham and Shamir and is the first attack that fully breaks the DES algorithm [NIS79]. This technique has been largely analyzed and further developed, leading to attacks like truncated differentials [Knu94], the boomerang attack [Wag99] or impossible differentials [BBS99]. Differential cryptanalysis is a statistical chosen-plaintext attack that follows the difference $a = m \oplus m'$ between two plaintexts m and m', usually differing in only a few bits. Here we look only into XOR difference, however other group operations can also be analyzed. The corresponding difference

between the ciphertext can be exploited to reveal information about the secret key.

A differential characteristic over n rounds of an iterated cipher is defined as the sequence of intermediate differences $(a = a_1, a_2 \dots a_n = b)$, where each (a_i, a_{i+1}) represents the input and output difference of one round. In order to calculate the probability of a one-round transition (a_i, a_{i+1}), we look into the non-linear round operation, typically an SBox. A Difference Distribution Table (DDT) over a given SBox contains the information of how many times each input difference leads to each output difference and is calculated over all possible inputs. The more probable transformations from a DDT table are used to construct differential characteristics over more rounds. Then, assuming that the rounds are independent and the keys are uniformly distributed, the Expected Differential Probability (EDP) of a characteristic is computed by multiplying the probabilities of each individual round:

$$EDP(a_1, a_2, \dots a_n) = \prod_{i}^{n} DP(a_{i-1}, a_i) \tag{1}$$

where DP represents the probability differential characteristics for one round.

In practice, only the input and output differences can be observed without considering the intermediate differences. We define the set of all differential characteristics with input difference a and output difference b as a differential (a, b). The probability of a differential (a, b) is the sum of the probabilities of all differential characteristics having input difference a and output difference b.

Key-recovery To recover the secret values used in round r, we use an $r - 1$ round differential characteristic (distinguisher) with probability p. First, we select $N = \frac{4}{p}$ random pair plaintexts with the expected input difference and their corresponding ciphertexts. The constant 4 is chosen to guarantee the discarding of wrong candidate keys and ensure the success of the attack. The next step is to guess the secret values and verify the guess using the distinguisher. For each key guess, a counter is kept and incremented if the expected difference is satisfied after partial decryption using the candidate values. The set of keys with the largest counter is potential candidates for the secret values.

2.3 Rotational property

Word rotation is a commonly used operation for symmetric-key algorithms. The rotational property is linear with respect to cyclic rotation and XOR, but is not linear with respect to modular addition. While we have

$$\Pr\left[(x \lll l) \oplus (y \lll l) = (x \oplus y) \lll l\right] = 1 \tag{2}$$

and

$$\Pr\left[(x \lll l_1) \lll l_2 = (x \lll l_2) \lll l_1\right] = 1 \tag{3}$$

For modular addition we have the following probability [Dau05]:

$$\begin{aligned} Pr[(x +_m y) \lll l = (x \lll l) +_m (y \lll l)] \\ = \frac{1}{4}(1 + 2^{l-n} + 2^{-l} + 2^{-n}) \end{aligned} \tag{4}$$

where l is the rotation constant and n is the size of the word.

2.4 Specifications of K-Cipher

K-Cipher is a low latency parameterizable block cipher proposed by Intel Research Labs [KDGD20a, KDGD20b] in 2020. The original design of K-Cipher, called Flex flow, uses a deterministic SBox layer in its design. However, the authors suggested that Flex flow is insecure and proposed a new version, called CPA flow, with a nondeterministic SBox layer. We will omit the analysis of the Flex flow and focus on analyzing the CPA flow which is believed to be secure. We analyze the variant of K-Cipher with state size $n = 24$. A discussion on the different choices of parameters is presented in Section 3.5 after the key-recovery attack is explained.

Parameters selection. K-Cipher has a state size of $24 \leq n \leq 1024$ and computes other parameters such as length of the keys, randomizers, and SBoxes using the state size n. The function `GetSBoxLengths(n)` is used to compute the length of the SBox m which we also refer to as block size. For our case of analysis with $n = 24$, the block size is $m = 8$. The length of the secret values is computed using the function `GetKeyLength(n, _)`. The length of the secret for a state of length n is $l + 6n$ where l is a constant computed using the function `GetFlexKeyLength(n)`. For $n = 24$, we have $l = 96$ and the key length is $96 + 6 \cdot 24 = 240$ bits.

Key and Randomizers. The secret values used in K-Cipher are the master key $\mathcal{K}$, and the randomizers $\mathcal{R}$. The master key is used to generate round keys and the goal of the randomizers is to randomize the SBox layer so that each SBox behaves differently. The round keys are denoted by k_0, k_1, k_2, k_3 where k_0, k_1, k_2 are derived from a key schedule and k_3 is a known permutation of k_2. The randomizers are indexed by $r_0^j, r_1^j \;\; 0 \leq j \leq 2$. For the state size $n \leq 32$, k_0, k_1, k_2 are substrings of $\mathcal{K}$ and are independent random values while for $n > 32$, the key schedule is the Flex flow. The idea of the key schedule algorithm for states of size $n > 32$ is described in Section 3.5. For the more detailed description of the key schedule, we refer to [KDGD20a].

Algorithm. The encryption algorithm is depicted in Figure 1. The main component is called aggressive adder which is the modular addition of the round key over $GF(2^m)$ with the state followed by a permutation layer. The result of the aggressive adder function then goes to the SBox layer. The SBox layer of the jth round XORs the m-bit block with the randomizer $r_0[j]$, then the multiplicative inverse of each resulted block in $GF(2^m)$ is added to randomizer $r_1[j]$

using modular addition. Finally, each block is cyclically rotated by 2. In the case where n is not a multiple of m, the last z bits are treated as a block of size z, and all operations are performed in $GF(2^z)$.

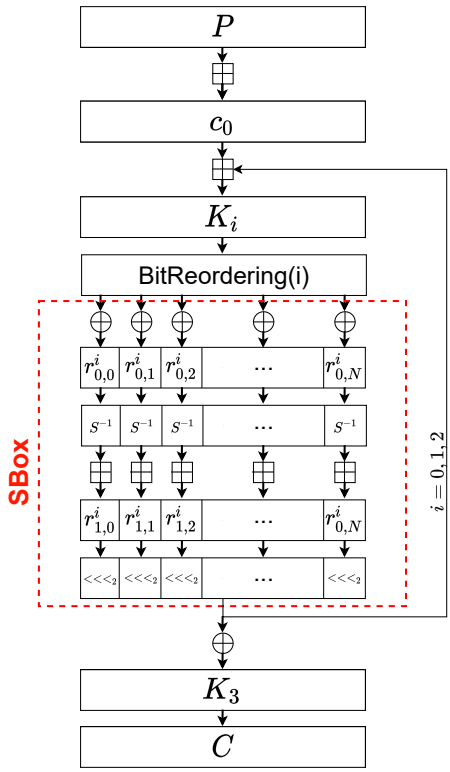

Fig. 1. The encryption algorithm of K-Cipher. First, the constant c is added to the input P using modular addition. Then 3 rounds are performed, each consisting of modular addition with the round key followed by a randomized SBox layer. Finally, another key value is XORed to the state and the ciphertext C is returned.

Security claims. The authors claim that the security is independent of the size of the input and perform experiments over 24-bit state. The security over all other input sizes is then bounded by these results. They report the existence of differentials with a maximum probability of $2^{-17.61}$ over 3 rounds and $2^{-20.54}$ over 4 rounds and suggested that $r \geq 2$ is enough to resist differential cryptanalysis attack.

3 Differential Cryptanalysis of K-Cipher

In this section, we analyze the security of K-Cipher against differential cryptanalysis. First, we explain how to find differential characteristics of secret SBoxes used in K-Cipher. Next, we describe how to recover the secret key and the secret randomizers. Finally, we analyze the complexity and the running time of the attack.

3.1 Generating differential characteristics

The SBox transformation used in K-Cipher is the following function mapping $3m$ bits to m bits.

$$S(x, r_0^i[j], r_1^i[j]) = \left[(x \oplus r_0^i[j]) +_m r_1^i[j]\right] \lll 2 \tag{5}$$

The use of so much randomness is motivated by improving the resistance of the SBox layer towards differential cryptanalysis by adding extra entropy. However, it is easy to see that for each $y \in GF(2^m)$, there are 2^{2m} tuples of (x, r_0, r_1) such that $S(x, r_0, r_1) = y$. Therefore, the construction of SBox leads to duplicate equivalent keys.

Our goal is to find differential characteristics such that the pair of inputs differ at exactly one bit, and their corresponding 2-round encryptions also differ at exactly one bit. Since xor with r_0 does not change the difference, we create DDT tables for $x^{-1} +_m r_1$. For each 8-bit possible value r_1, we create a 256 DDT table. Then all DDTs are combined into one General DDT, where the position (i, j) in GDDT will be non-zero if and only if it is non-zero for each of the 256 DDT tables. To ensure that our characteristic is valid for any choice of r_1^i we use the GDDT to create the needed characteristics. All 1-bit transitions of the GDDT are summarized in Table 1, where the minimum probability is 2^{-7}. The characteristics used in attacking each round of the key-recovery attack are shown in Table 2. The expected probability of a characteristic depends on the SBox and the modular addition and is calculated as the product of the one-round probabilities. The probability of 1-bit transition for modular addition over $GF(2^8)$ is $2^{-0.83}$. Therefore, A two-round characteristic has the probability $\Pr \leq 2^{-7} \cdot 2^{-0.83} \cdot 2^{-7} \cdot 2^{-0.83} = 2^{-15.66}$. All probabilities in the tables are verified empirically.

3.2 Description of the attack

We divide the attack into three phases for simplicity and in each phase, we recover additional bits of the unknown parameters. Each phase can also be seen as an individual attack over a similar primitive. The first phase targets the last round of the algorithm and obtains 2^3 indistinguishable candidates for k_3, k_2 and r_1^2. Phase two extends to the second round of the algorithm and recovers the randomizer values r_0^2 and r_1^1. Next phase repeats the same attack and recovers r_0^1 and $(r_1^0 \lll 2) +_m k_1$, where the exact values of r_1^0 and k_1 are not needed. Finally, the values of r_0^0 and k_0 are guessed.The details of each phase of the attack are described in Section 3.2.

Table 1. A part of the GDDT consisting only of the differences that satisfy our requirements for one active bit in the input and the output difference. The entry (i, j) gives the average number of times the input difference holds over all possible values of the randomizer for an input difference i and an output difference j. The probability is then the number divided by 256. For simplicity the unfilled values of the table are zeroes.

	1	2	4	8	16	32	64	128
1	2							
2			5					
4		3			3			
8	2			2				
32								2
64								2
128						3		

Phase 1:Recovering k_3, k_2 and r_1^2. We construct 3 characteristics, each having a different active output block after Bit-reordering(2). For each characteristics $i, i = 0, 1, 2$, we iterate over all possible 8-bit values of $k_3[i]$ and $r_1^2[i]$ and partially decrypt until the xor addition of r_0^2 of the SBox layer. The empirical difference is then verified with the difference of the corresponding characteristic in order to filter the correct values. If the partial decryption results in the expected difference, a counter associated with the current values of $k_3[i]$ and $r_1^2[i]$ is incremented. The counter with the highest value is the most likely assignment. The partial decryption is described in Algorithm 2 and the full recovery can be seen in Algorithm 1. The mathematical structure of K-Cipher implies that there are two pairs with the same highest counter value, and these are indistinguishable. The existence of such equivalent candidates is proven in Lemma 1. This results in 2^3 candidate pairs of k_3 and r_0^2 and the choice of any of them will result in the same ciphertext. This is possible only because for $n \leq 32$ the round keys are directly obtained from the master key and there is no key schedule, therefore the keys and randomizers are independent. In the case of $n > 32$, keys are derived from a key schedule, which is bijective and hence invertible. By deriving 2^m candidate last round keys leads to 2^m different candidate master keys, the correct value of which can be guessed with probability 2^{-m}. After recovering k_3, we can recover the value of k_2 using the inverse of the BitReordering$(k_2, 3)$ permutation.

Lemma 1. *The size of the set of candidates returned by the Algorithm 1 is always 2.*

Proof. The proof contains two parts. The first part is when $t = 3$ and the second part when $t \leq 2$. When $t = 3$, the set of candidates has the form $\{(k, r), (k +_m 2, r +_m 128)\}$ due the structure of the function. If $k \equiv 0 \mod 4$, then $k = k_7k_6k_5k_4k_300$ and $k +_m 2 = k_7k_6k_5k_4k_310$. The first step in the Algorithm 2 is xored with the input. Let us denote the input with $x = x_7x_6x_5x_4x_3x_2x_1x_0$. We have $y = x \oplus k = y_7y_6y_5y_4y_3y_2x_1x_0$ and $y' = x \oplus (k +_m 2) = y_7y_6y_5y_4y_3y_2(\neg x_1)x_0$.

Algorithm 1: Algorithm of the differential cryptanalysis attack, the return value for phase 1 is $k_3[j], r_1^2[j]$, for phase 2 is $r_0^2[j], (r_1^1 \ggg 2) +_m k_2$, and for phase 3 is $r_0^1[j], (r_1^0 \ggg 2) +_m k_1$

Input: Characteristic $\mathcal{C}$ and the target round $1 \leq \mathcal{R} \leq 3$.
Output: Set of all candidates for the secret of round $\mathcal{R}$.

$\mathcal{S} := \{[(P_{i,1}, C_{i,1}), (P_{i,2}, C_{i,2})] \mid i \leq 2^{16} \text{ and } P_{i,1} \oplus P_{i,2} = e_{\text{input difference}}\}$.

$t := \mathcal{R} - 1$

for $0 \leq i \leq |\mathcal{S}|$ **do**

  for *each* 8*-bit strings* k *and* r **do**

    if $\mathcal{R} \leq 2$ **then**

      $C_{i,1} :=$ Decryption of $C_{i,1}$ until r_0^t.

      $C_{i,2} :=$ Decryption of $C_{i,2}$ until r_0^t.

    end

    $\mathcal{X}_1 := \text{PartialDecryption}(k, r, C_{i,1}, t)$

    $\mathcal{X}_2 := \text{PartialDecryption}(k, r, C_{i,2}, t)$

    if $\mathcal{X}_1 \oplus \mathcal{X}_2 = e_{output\ difference}$ **then**

      $T_{k,r} := T_{k,r} + 1$

    end

  end

end

return $\{(k, r) \mid T_{k,r} \text{ is maximum}\}$

Algorithm 2: One round partial decryption using candidate values k and r for a block ct of length m. The t is the target round

PartialDecryption(k, r, ct, t)
$ct := ct \oplus k$
if $t \leq 2$ **then**
  $ct := ct -_m r$
  $ct := ct \ggg 2$
else
  if $t = 3$ **then**
    $ct := ct \ggg 2$
    $ct := ct -_m r$
  end
end
$ct := ct^{-1}$
return ct

Let $r = r_7r_6r_5r_4r_3r_2r_1r_0$ and $r +_m 128 = (\neg r_7)r_6r_5r_4r_3r_2r_1r_0$. We have the following:

$$\begin{aligned}&(y \ggg 2) -_m r\\ &= x_1x_0y_7y_6y_5y_4y_3y_2 -_m r_7r_6r_5r_4r_3r_2r_1r_0\end{aligned} \tag{6}$$

$$\begin{aligned}&(y' \ggg 2) -_m (r +_m 128)\\ &= (\neg x_1)x_0y_7y_6y_5y_4y_3y_2 -_m (\neg r_7)r_6r_5r_4r_3r_2r_1r_0\end{aligned} \tag{7}$$

Since the most significant bit of the both operands is complemented, the result of the modular subtraction will be the same. In the case of $k \equiv 1 \mod 4$ the same argument applies because only the value of x_0 is complemented after the xor and does not affect the rest of the computations. When $t \leq 2$, the set of candidates has the form $\{(k, r), (k +_m 128, r +_m 128)\}$ because of the structure of the function. Let us define $k = k_7k_6k_5k_4k_3k_2k_1k_0$ and $k +_m 128 = (\neg k_7)k_6k_5k_4k_3k_2k_1k_0$. Like the previous part of the proof, we denote the input with $x = x_7x_6x_5x_4x_3x_2x_1x_0$. Let $y = x \oplus k = y_7y_6y_5y_4y_3y_2y_1y_0$ and $y' = x \oplus (k +_m 128) = (\neg y_7)y_6y_5y_4y_3y_2y_1y_0$. Let $r = r_7r_6r_5r_4r_3r_2r_1r_0$ and $r +_m 128 = (\neg r_7)r_6r_5r_4r_3r_2r_1r_0$. We have the following:

$$\begin{aligned}&(y -_m r)\\ &= y_7y_6y_5y_4y_3y_2y_1y_0 -_m r_7r_6r_5r_4r_3r_2r_1r_0\end{aligned} \tag{8}$$

$$\begin{aligned}&(y' -_m (r +_m 128)\\ &= (\neg y_7)y_6y_5y_4y_3y_2y_1y_0 -_m (\neg r_7)r_6r_5r_4r_3r_2r_1r_0\end{aligned} \tag{9}$$

Since the most significant bit of both operands is complemented, the result of the modular subtraction will be the same. In both cases, the number of candidate pairs is 2.

Table 2. Differential characteristics used in the attack. Each column shows the position of the difference in the characteristic. Re(t) columns show the position of the difference after BitReordering(t). $\Pr_{Re(t)}$ shows the probability of the characterstic after BitReordering(t). The probabilities are in the base 2 logarithm of the characteristic probability. The elements x are not important because they are not used in the attack.

Input	Re(0)	Re(1)	Re(2)	$\Pr_{Re(0)}$	$\Pr_{Re(1)}$	$\Pr_{Re(2)}$
22	17	11	16	-1.1	-6.3	-11.1
15	8	18	13	-0.28	-6.5	-10.9
19	19	0	2	-0.96	-10	-11.8
4	0	x	x	-0.29	x	x

Phase 2: Recovering r_0^2 and r_1^1. Recovering r_0^2 and r_1^1 is simpler since we already recovered k_3, k_2, r_1^2 and we can partially decrypt any ciphertext for one round. Then for every $r_0^2[j]$ and every $k_2[j] + (r_1^1[j] \lll 2)$ we partially decrypt the second round until the XOR addition with r_0^1 and compare the intermediate difference with the expected difference from the corresponding characteristic. The attack starts from the SBox transformations from round 2 until the xor addition of r_0^2. We define the function $F_2 : \{0,1\}^{24} \to \{0,1\}^{24}$ as follow.

$$F_2(X) = P(((S(X) + r_1^1) \lll 2) +_m k_2) \oplus r_0^2 \tag{10}$$

Where P and S represent the BitReordering and SBox function respectively. Since xor and the permutation are both linear and commutative, we can swap their positions. Additionally, the permutation is known, so we can exclude it from the equation because the state after the permutation is known. Therefore, we have

$$F_2(X) = ((S(X) + r_1^1) \lll 2) +_m k_2 \oplus r_0^2. \tag{11}$$

We underline that aim at recovering the value of $k_2 + (r_1^1 \ggg 2)$ instead of the exact values of k_2 and r_1. In fact, knowing only the sum is enough to encrypt/decrypt messages. However, this value can not be recovered directly, because the rotation is applied over the state after the SBox transformations and the modular addition of the randomizer. To overcome this, we use the rotational property for modular addition and can say that

$$(S(X) + r_1^1) \lll 2 = (S(X) \lll 2) +_m (r_1^1 \lll 2) \tag{12}$$

holds with probability 0.4 for rotation constant $l = 2$ and size of the word $n = 8$ bits (see Equation 4). Then

$$F_2(X) = (S(X) \lll 2) +_m (r_1^1 \lll 2) +_m k_2 \oplus r_0^2 \tag{13}$$

holds with probability $p = 0.4 = 2^{-1.32}$. Therefore our characteristic will hold with probability $2^{-7} \times 2^{-1.32} = 2^{-8.32}$. The partial decryption function associated to Algorithm 2 is

$$F^{-1}(Y) = S^{-1}(((Y \oplus r_0^2 -_m k_2 -_m (r_1^1 \ggg 2)) \ggg 2) \tag{14}$$

where Y represents the ciphertext after 2 rounds, recovered by the previous phase. Then we guess the value of $k_2 +_m (r_1^1 \ggg 2)$ and as we already recovered k_2 in Section 3.2, we can recover r_1^1.

Phase 3: Recovering r_0^1, r_1^0 and k_1. We apply the same attack as in Section 3.2, but this time over the first round. The function we partially decrypt over is defined as follow.

$$F_3(X) = (S(X) \lll 2) + (r_1^0 \lll 2) +_m k_1 \oplus r_0^1 \tag{15}$$

and the inverse is

$$\begin{aligned} & F_3^{-1}(Y) \\ & = S^{-1}(Y \oplus r_0^1 -_m k_-(r_1^0 \ggg 2)) \end{aligned} \tag{16}$$

The algorithm of recovering r_0^1 and $r_1^0 \ggg 2 +_m k_1$ is described in the Algorithm 1.

Phase 4: Recovery of r_0^0 and k_0. Recovering r_0^0 and k_0 is straightforward since we can decrypt any ciphertext for three rounds. We can guess the values for $r_0^0[j]$ and $k_0[j]$ and verify the guess using 2^{16} different pairs. Let P be the first permutation BitReordering 0, for each pair 8-bit values (X, Y), there are 2^8 possible values of r_0^0 and k_0 such that $P(X +_m k_0) \oplus r_0^0 = Y$.

3.3 Complexity of the attack

The data complexity is computed as the number of times we call the encryption/decryption function. In the first phase of the attack, the probability of each characteristic is lower bounded by $p = 2^{-12}$ hence exploiting each differential characteristic requires encrypting at least $\frac{1}{p}$ pairs of chosen plaintexts. In our attack, we use $\frac{4}{p}$ pairs of chosen plaintext to guarantee a successful attack. The partial decryption in the last round involves one SBox out of the total 9 SBoxes that are computed in one encryption/decryption, which can be estimated to a total of $2^{15} \cdot \frac{1}{9}$ decryptions. Furthermore, we search the whole space of two 8-bit values to compute the respective key and randomizer values. Therefore the total complexity of the first phase is $3 \cdot 2^{16} \cdot 2^{15} \cdot \frac{1}{9} \approx 2^{29.4}$. The second phase has less complexity since the used characteristics are over 1 round and have a larger probability. The complexity of the second phase is $2^{27.4}$ and the complexity of the third phase is $2^{21.4}$. The total complexity of the attack is $2^{29.7}$.

The complexity of the attack depends on the probability of the characteristics, which depends on the GDDT of the SBoxes. Thus, we can say that all versions of the K-Cipher that have the same size SBoxes have the same theoretical complexity. However, in practice, the execution of the attack will have different measured times. This happens because the time required for one run of the algorithm is proportional to the number of SBoxes in one encryption. Hence, the smaller the state, the faster the attack. One way to compare the practical time is to look at the complexity as the number of SBox look-ups instead of the number of encryptions. For example, a 24-bit state with an 8-bit SBox would take $2^{29.4} \cdot 9$ table look-ups, while a 64-bit state with an 8-bit SBox will require $2^{29.4} \cdot 24$ table look-ups. We expect the second version to be around 2.5 times slower.

3.4 Experimental verification of the attack

We implemented our attack in C++ and run it on a 5.13.0-35-generic $x86_64$ Linux machine with 32 GiB of memory and Intel Core i7 processor. Our imple-

mentation terminates after 1812s and recovers all the secret data. The implementation of the attack is available on Github[3].

3.5 Other parameters for K-Cipher

The main reason for the choice of parameters in this paper is that K-Cipher is mostly intended to be used for small states [KDGD20a]. However, the same attack can be applied to any other choice of parameters. There are three main parameters that affect the overall security of the system: The state size, the block size, and the number of rounds. The block size determines the order of the underlying binary field and for larger orders, the probability of the differential characteristics is smaller and the DDTs are different. For the same block size, the larger state does not increase the complexity of the attack but increases the running time of the algorithm. The larger number of rounds results in characteristics with smaller probability and hence increases the complexity of the attack.

The other main difference for states of size $n > 32$, is that the round keys are derived from a key schedule algorithm and they are not independent. The dependence of round keys needs two adjustments in the attack we described in this paper. First, after recovering k_3, we can recover all other round keys using the Algorithm 3. Second, we can not choose any member of an equivalence class of the candidates and we need to guess the correct one. This can be done efficiently using exhaustive search and verification of each guess.

Algorithm 3: Recovering all the round keys. ReverseBitReordering functions are reversing the permutations specified in K-Cipher paper. The subtraction $k_i -_m c_j$ is a modular subtraction.

Input: k_3
$k_2 = \text{ReverseBitReordering}(k_3, 3)$
$k_1 = \text{ReverseBitReordering}(k_2, 7)$
$k_1 = S^{-1}(k_1)$
$k_1 = \text{ReverseBitReordering}(k_1, 6)$
$k_1 = k_1 -_m c_2$
$k_0 = \text{ReverseBitReordering}(k_1, 5)$
$k_0 = S^{-1}(k_0)$
$k_0 = \text{ReverseBitReordering}(k_0, 4)$
$k_0 = k_0 -_m c_1$
return k

[3] https://github.com/KULeuven-COSIC/K-Cipher.

4 Conclusion

The main motivation of designing K-Cipher was to have a low latency cipher that is efficient in different use cases. The original design with deterministic SBox layers was claimed to be vulnerable to differential cryptanalysis. Therefore, to improve its security, the designers suggested increasing the key length and using additional randomness in the SBox layer. The attack presented here showed that not only the increase of the key length does not result in a secure cipher but it also creates duplicate keys which is not a desirable property for a block cipher. Apart from the existence of the GDDTs, the attack exploited the low number of rounds of the cipher. Although the number of rounds improves the latency, it jeopardizes security.

References

Ava17. Roberto Avanzi. The QARMA block cipher family. almost MDS matrices over rings with zero divisors, nearly symmetric even-mansour constructions with non-involutory central rounds, and search heuristics for low-latency s-boxes. *IACR Trans. Symmetric Cryptol.*, 2017(1):4–44, 2017.

BBS99. Eli Biham, Alex Biryukov, and Adi Shamir. Cryptanalysis of Skipjack Reduced to 31 Rounds Using Impossible Differentials. In *EUROCRYPT*, volume 1592 of *Lecture Notes in Computer Science*, pages 12–23. Springer, 1999.

BCG+12. Julia Borghoff, Anne Canteaut, Tim Güneysu, Elif Bilge Kavun, Miroslav Knezevic, Lars R. Knudsen, Gregor Leander, Ventzislav Nikov, Christof Paar, Christian Rechberger, Peter Rombouts, Søren S. Thomsen, and Tolga Yalçin. PRINCE - A low-latency block cipher for pervasive computing applications - extended abstract. In Xiaoyun Wang and Kazue Sako, editors, *Advances in Cryptology - ASIACRYPT 2012 - 18th International Conference on the Theory and Application of Cryptology and Information Security, Beijing, China, December 2-6, 2012. Proceedings*, volume 7658 of *Lecture Notes in Computer Science*, pages 208–225. Springer, 2012.

BEK+20. Dusan Bozilov, Maria Eichlseder, Miroslav Knezevic, Baptiste Lambin, Gregor Leander, Thorben Moos, Ventzislav Nikov, Shahram Rasoolzadeh, Yosuke Todo, and Friedrich Wiemer. Princev2 - more security for (almost) no overhead. In Orr Dunkelman, Michael J. Jacobson Jr., and Colin O'Flynn, editors, *Selected Areas in Cryptography - SAC 2020 - 27th International Conference, Halifax, NS, Canada (Virtual Event), October 21-23, 2020, Revised Selected Papers*, volume 12804 of *Lecture Notes in Computer Science*, pages 483–511. Springer, 2020.

BJK+16. Christof Beierle, Jérémy Jean, Stefan Kölbl, Gregor Leander, Amir Moradi, Thomas Peyrin, Yu Sasaki, Pascal Sasdrich, and Siang Meng Sim. The SKINNY family of block ciphers and its low-latency variant MANTIS. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part II*, volume 9815 of *Lecture Notes in Computer Science*, pages 123–153. Springer, 2016.

Dau05. Magnus Daum. Cryptanalysis of hash functions of the md4-family. 2005.

KDGD20a. Michael Kounavis, Sergej Deutsch, Santosh Ghosh, and David Durham. K-cipher: A low latency, bit length parameterizable cipher. In *2020 IEEE Symposium on Computers and Communications (ISCC)*, pages 1–7, 2020.

KDGD20b. Michael E. Kounavis, Sergej Deutsch, Santosh Ghosh, and David Durham. K-cipher: A low latency, bit length parameterizable cipher. *IACR Cryptol. ePrint Arch.*, page 30, 2020.

Knu94. Lars R. Knudsen. Truncated and Higher Order Differentials. In *FSE*, volume 1008 of *Lecture Notes in Computer Science*, pages 196–211. Springer, 1994.

LMMR21. Gregor Leander, Thorben Moos, Amir Moradi, and Shahram Rasoolzadeh. The SPEEDY family of block ciphers engineering an ultra low-latency cipher from gate level for secure processor architectures. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2021(4):510–545, 2021.

LRD+21. Michael LeMay, Joydeep Rakshit, Sergej Deutsch, David M. Durham, Santosh Ghosh, Anant Nori, Jayesh Gaur, Andrew Weiler, Salmin Sultana, Karanvir Grewal, and Sreenivas Subramoney. Cryptographic capability computing. In *MICRO '21: 54th Annual IEEE/ACM International Symposium on Microarchitecture, Virtual Event, Greece, October 18-22, 2021*, pages 253–267. ACM, 2021.

NIS79. NIST. FIPS-46: Data Encryption Standard (DES). 1979. http://csrc.nist.gov/publications/fips/fips46-3/fips46-3.pdf.

Wag99. David A. Wagner. The Boomerang Attack. In *FSE*, volume 1636 of *Lecture Notes in Computer Science*, pages 156–170. Springer, 1999.

Chapter 9

On the SNOW family of stream ciphers

This work was presented as a poster to the 41st Symposium on Information Theory in the Benelux.

I have carried the research in this paper under the supervision of my co-author. Both the paper and the poster were written and presented by me.

On the SNOW Stream Ciphers

Liliya Kraleva Chaoyun Li
imec-COSIC
Dept. Electrical Engineering (ESAT), KU Leuven
Leuven, Belgium
liliya.kraleva@esat.kuleuven.be chaoyun.li@esat.kuleuven.be

Abstract

SNOW 3G is the core of the UEA2 & UIA2 algorithms in the 3GPP standard. Our work revisits previous results on chosen IV differential cryptanalysis and adjusts some results within the original steps of the SNOW algorithm. Further, we extend the analysis with 2 rounds and compare the algorithms of SNOW 2/ $2^{\oplus}$/ 3G / 3G$^{\oplus}$ with respect to the number of active Sboxes.

1 Introduction

SNOW 3G is a word-oriented synchronous stream cipher. It is the core of the integrity and confidentiality algorithms UEA2 and UIA2 of the 3GPP UMTS system, which are identical to the algorithms 128-EEA1 and 128-EIA1 in LTE. The previous version SNOW 2.0 is developed by Ekdahl and Johansson [4]. Recently, a new version, namely SNOW-V [5] has been proposed for 5G.

We briefly recall some cryptanalysis of SNOW 2.0 and SNOW 3G. Linear distinguishing attacks and fast correlation attacks against SNOW 2.0 and have been investigated in [6, 11, 8, 13]. To strengthen the resistance against algebraic attacks [1], the SNOW 3G has been proposed. Recently, Yang *et al.* propose distinguishing and key recovery attacks on SNOW 3G based on vectorized linear approximations [12].

With respect to differential attacks there are few results on SNOW 3G, assuming chosen IV scenarios. In [2] Biryukov *et al.* show a 13-round multiset distinguisher with low complexity. Further, they extend similar distinguisher on SNOW 3G$^{\oplus}$ and recover the key for up to 18 initialization rounds. Recently, a differential attack improving the results from [2] is proposed in [7] .

In this paper we revisit the attack in [7] and present adjusted results according to the original details of the algorithm. We show that the adjusted attacks can in fact cover one round less than the authors presented. Further, we consider the composition of Sboxes in order to extend the analysis with 2 more rounds. Additionally, we compare the SNOW 2.0 and SNOW 3G versions with respect to differential cryptanalysis and use the metric of active Sboxes to measure the diffusion of the FSM. We show that for up to 12 initialization rounds the differences behave identically for the two versions and produce keystream words with the same difference. Finally, we give a lower bound on the number of active Sboxes over 2 rounds.

Section 2 introduces the algorithm details. The adjusted results compared to [7] are given in Section 3. We further give observations on how to extend the analysis in Section 4. The branch number of a transformation is introduced in section 5, together with our motivation about the minimal number of active Sboxes for 2 rounds of the algorithm. Then we compare

the SNOW versions and provide tables for easier visualization. Finally, some conclusions and future work are presented in Section 6.

2 The algorithms of SNOW 2 and SNOW 3G

SNOW 3G consist of a linear feedback shift register (LFSR) with 16 32-bit registers denoted s_i and a finite state machine(FSM) as a nonlinear component with three 32-bit registers denoted R_1, R_2, R_3. The feedback word of the LFSR is recursively computed as

$$s_{15}^t = \alpha^{-1} \cdot s_{11}^{t-1} + s_2^{t-1} + \alpha \cdot s_0^{t-1},$$

where α is the root of $x^4 + \beta^{23}x^3 + \beta^{245}x^2 + \beta^{48}x + \beta^{239}$ over $GF(2^8)[x]$ and β is the root of $x^8 + x^7 + x^5 + x^3 + 1$ over $GF(2)[x]$.

At time t, the FSM outputs

$$F^t = (s_{15}^{t-1} + R_2^{t-1}) \oplus R_2^{t-1},$$

after which the registers are updated as

$$R_1^t = R_2^{t-1} + (R_3^{t-1} \oplus s_5^{t-1}), \quad R_2^t = S_1(R_1^{t-1}), \quad R_3^t = S_2(R_2^{t-1}).$$

The 32-bit Sboxes S_1 and S_2 are composed of a MixColumn operation and four 8-bit Sboxes S_R and S_Q applied to each byte, respectively. Note that S_R is the AES S-box. Let $w = (w_0, w_1, w_2, w_3)$ be a 32-bit word and let $r = (r_0 r_1, r_2, r_3)$ be the output of $S_1(w)$. Then

$$\begin{pmatrix} r_0 \\ r_1 \\ r_2 \\ r_3 \end{pmatrix} = \begin{pmatrix} x & x+1 & 1 & 1 \\ 1 & x & x+1 & 1 \\ 1 & 1 & x & x+1 \\ x+1 & 1 & 1 & x \end{pmatrix} \cdot \begin{pmatrix} S_R(w_0) \\ S_R(w_1) \\ S_R(w_2) \\ S_R(w_3) \end{pmatrix},$$

where x is the root of $x^8 + x^4 + x^3 + x^1 + 1$ over $GF(2^8)$. Correspondingly, for $r = S_2(w)$ we have

$$\begin{pmatrix} r_0 \\ r_1 \\ r_2 \\ r_3 \end{pmatrix} = \begin{pmatrix} y & y+1 & 1 & 1 \\ 1 & y & y+1 & 1 \\ 1 & 1 & y & y+1 \\ y+1 & 1 & 1 & y \end{pmatrix} \cdot \begin{pmatrix} S_Q(w_0) \\ S_Q(w_1) \\ S_Q(w_2) \\ S_Q(w_3) \end{pmatrix},$$

where y is the root of $y^8 + y^6 + y^5 + y^3 + 1$ over $GF(2^8)$.

The cipher works in two modes - an initialization mode and a key-stream mode. In the initialization mode, first the state is initialized by the 128 bit key and the 128 bit IV as:

$$\begin{array}{llll} s_{15} = k_3 + IV_0, & s_{14} = k_2, & s_{13} = k_1, & s_{12} = k_0 + IV_1 \\ s_{11} = k_3 \oplus 1, & s_{10} = k_2 \oplus 1 \oplus IV_2, & s_9 = k_1 \oplus 1 \oplus IV_3, & s_8 = k_0 \oplus 1 \\ s_7 = k_3, & s_6 = k_2, & s_5 = k_1, & s_4 = k_0 \\ s_3 = k_3 \oplus 1, & s_2 = k_2 \oplus 1, & s_1 = k_1 \oplus 1, & s_0 = k_0 \oplus 1 \end{array}$$

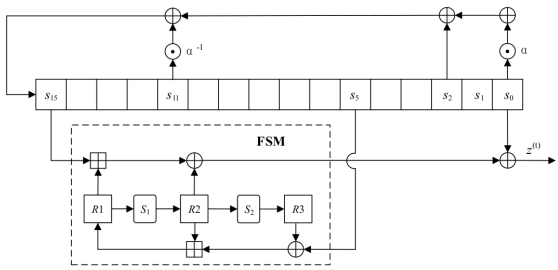

Figure 1: Diagram of the key-stream mode of SNOW3G.

During the initialization mode the output of the FSM is fed to the LFSR as $s_{15}^{t+1} = \alpha^{-1} \cdot s_{11}^t \oplus s_2^t \oplus \alpha \cdot s_0^t \oplus F$. After the initialization, the FSM and LFSR are clocked once more in keystream mode and the output of the FSM is discarded. Finally, the first keystream is produced and for each next keystream the stream cipher works in a keystream mode as shown in Figure 1.

SNOW 2 has the same LFSR, initialization and key generation as SNOW 3G, however the FSM has one less register. Therefore, we have

$$S_{15} = \alpha^{-1} S_{11} \oplus S_2 \oplus \alpha S_0 \oplus F,$$
$$F = (S_{15} + R_1) \oplus R2$$
$$R_1 = R_2^{t-1} + S_5^{t-1}, \quad R_2 = S_1(R_1^{t-1}),$$

where S_1 is the same as in SNOW 3G. Finaly, the keystream

$$z = F \oplus S_0.$$

The algorithms of SNOW $2^{\oplus}$/3G$^{\oplus}$ are the same, but with all modular additions substituted with XORs.

3 Differential attack revisited

The authors of [7] merge the initialization rounds and the step performed before producing the keystream. This is not necessary here since the analysis is fully feasible when keeping the original structure of the algorithm. In this section we show the difference between the two initialization algorithms and propose adjusted results.

Algorithms 1 and 2 describe the two implementations by taking t initialization rounds. According to Algorithm 1, the value of the register s_{15}, used in the first keystream word is calculated as $s_{15}^t = \alpha^{-1} s_{11}^{t-1} \oplus s_2^{t-1} \oplus \alpha s_0^{t-1} \oplus F^t$, leading to a keystream

$$z^0 = F^{t+1} \oplus s_0^t = (\alpha s_{11}^{t-1} \oplus s_2^{t-1} \oplus \alpha s_0^{t-1} \oplus F^t) + R1^t) \oplus R2^t \oplus s_0^t.$$

Alternatively, using Algorithm 2 leads to the value $s_{15}^{*(t+1)} = \alpha^{-1} s_{11}^t \oplus s_2^t \oplus \alpha s_0^t$. Then the keystream is

$$z^{*(0)} = F^{t+2} \oplus s_0^{t+1} = (\alpha s_{11}^t \oplus s_2^t \oplus \alpha s_0^t) + R1^{t+1}) \oplus R2^{t+1} \oplus s_0^{t+1}$$

Algorithm 1 Key generation as in [7]

1: **for** t=1,2,...t **do**
2: $F^t \leftarrow clockFSM$
3: $ClockLFSR_InitMode$
4: **for** $i = 0,1,\ldots n$ **do**
5: $F^{t+1+i} \leftarrow clockFSM$
6: $z^i \leftarrow F^{t+1+i} \oplus s_0^{t+i}$
7: $clockLFSR_KeystreamMode$

Algorithm 2 Key generation as in the specification [10] and this paper

1: **for** round=1,2,...t **do**
2: $F^t \leftarrow clockFSM$
3: $ClockLFSR_InitMode$
4: $F^{t+1} \leftarrow clockFSM$
5: $ClockLFSR_KeystreamMode$
6: **for** $i = 0,1,\ldots n$ **do**
7: $F^{t+2+i} \leftarrow clockFSM$
8: $z^i \leftarrow F^{t+2+i} \oplus s_0^{t+1+i}$
9: $clockLFSR_KeystreamMode$

Now let us consider IV and IV', such that $IV'_0 = IV_0 \oplus a$, where $a = 0x80000000$. Then if we load the state using the two IVs with the same key, we can follow how the difference a propagates for different number of initialization rounds. Table 1 shows how this difference propagates through the state for SNOW 3G, including the differences for s_{15} and s^*_{15}. Instead of computing the keystream difference

$$\Delta(0x????0000 + a) \oplus \Delta S_1(a) = 0x????||A||A$$

over 12 initialization rounds with Algorithm 1, by using Algorithm 2 we would have

$$\Delta(0x00??8000 + a) \oplus \Delta S_1(a) = 0x????||(A \oplus 0x80)||A$$

over 11 initialization rounds, where $A = \Delta S_R(0x80)$ is one of the 127 possible output differences of the S_R Sbox when the input difference is $0x80$, as shown in table 6 in [7].

In all further analysis in this paper we refer only to values obtained by Algorithm 2.

4 Observations on more rounds

As the registers of the FSM are updated by applying the Sboxes S_1 and S_2, it is natural to look into their compositions. In this section we analyse such compositions for SNOW 3G$^{\oplus}$.

The difference in register R_3 after 13 rounds is $R_3^{13} = \Delta S_2 S_1(a)$, so we look into the possible output difference of $S_2S_1(a)$, when the input difference of S_1 is a. From the specifications of the two Sboxes we have the following difference after S_1:

$\Delta(r_0^1, r_1^1, r_2^1, r_3^1) = M_1 \cdot \Delta(0,0,0,0x80)^T = (\Delta S_R(0x80), \Delta S_R(0x80), 3\Delta S_R(0x80), 2\Delta S_R(0x80))$.
Further, when we apply S_2 we have

$$\Delta(r_0^2, r_1^2, r_2^2, r_3^2) = M_2 \cdot (\Delta S_R(0x80), \Delta S_R(0x80), 3\Delta S_R(0x80), 2\Delta S_R(0x80)),$$

where $\Delta r_0^2 = 2S_Q(A) \oplus 3S_Q(A) \oplus S_Q(2A) \oplus S_Q(3A)$ and $A = \Delta S_R(0x80)$. Corresponding equations are derived for $\Delta r_1^2, \Delta r_2^2$ and Δr_3^2. We generated difference tables for the output bytes of the compositions $\Delta S_2 S_1(a)$ and $\Delta S_1(a \oplus S_1(a))$. Those are the compositions involved in the FSM registers at round 13 and 14, and involved in the keystream after 13 initialization rounds.

Table 1: The propagation of the difference a for SNOW 3G/2.0, where (s_{15}, z) and $(s_{15}*, z^*)$ are based on Algorithms 1 and 2 respectively, and $a = 0x80000000; b = 80800000; c =$??000000$; d =$??800000$; e =$??000000$; f =$??800000$; g =$??008000$; h =$????8000$; i =$????0000$; j =$????$AA; k, l =$ unknown.

SNOW 3G

r	s0	s1	s2	s3	s4	s5	s6	s7	s8	s9	s10	s11	s12	s13	s14	s15	s15*	R1	R2	R3	z	z^*
11	0	0	0	0	a	a	a	a	a	b	c	d	e	f	g	h	00??8000	a	0	0	????8000	????$(A \oplus 0x80)A$
12	0	0	0	a	a	a	a	a	b	c	d	e	f	g	h	i	00??0000	a	$\Delta S_1(a)$	0	????AA	?
13	0	0	a	a	a	a	a	b	c	d	e	f	g	h	i	j	8???8000	$\Delta(a+\Delta S_1(a))$	$\Delta S_1'(a)$	$\Delta S_2 S_1(a)$	?	?
14	0	a	a	a	a	a	b	c	d	e	f	g	h	i	j	k	80????80	$\Delta((a \oplus S_2 S_1(a)) + R_2^{13})$	$\Delta S_1(a+S_1(a))$	$\Delta S_2 S_1'(a)$	?	?
15	a	a	a	a	a	b	c	d	e	f	g	h	i	j	k	l	d0????17	$\Delta((a \oplus S_2 S_1'(a)) + R_2^{14})$	$\Delta S_1(R_1^{14})$	$\Delta S_2(R_2^{14})$	?	?

SNOW 2.0

11	0	0	0	0	a	a	a	a	a	b	c	d	e	f	g	h	00??8000	a	0	-	????8000	????$(A \oplus 0x80)A$
12	0	0	0	a	a	a	a	a	b	c	d	e	f	g	h	i	00??0000	a	$\Delta S_1(a)$	-	????AA	?
13	0	0	a	a	a	a	a	b	c	d	e	f	g	h	i	j	8???8000	$\Delta(a+\Delta S_1(a))$	$\Delta S_1'(a)$	-	?	?
14	0	a	a	a	a	a	b	c	d	e	f	g	h	i	j	k	80????80	$\Delta(a+\Delta S_1'(a))$	$\Delta S_1(\Delta(a+\Delta S_1(a)))$	-	?	?
15	a	a	a	a	a	b	c	d	e	f	g	h	i	j	k	l	d0????17	$\Delta(b+\Delta S_1(a+S_1(a)))$	$\Delta S_1(\Delta(a+\Delta S_1'(a)))$	-	?	?

As we can see from Table 2, for SNOW 3G$^{\oplus}$ the keystream difference after 12 rounds is $0x00800000 \oplus a \oplus S_1(a) \oplus S_1'(a) =$????$||X||X$, where X is the XOR of the outputs of two S_R Sboxes with input difference a. According to the difference table of S_R for input difference a (Table 6 in [7]), there are between 28 and 35 possible combinations, or candidate values, for the input values of S_R. Therefore, there are in average 31 possible values for the last byte of the keystream word, compared to 256 values if we have to guess.

The keystreams after 13 initialization rounds is

$$z = S_{15}^* \oplus R_1 \oplus R_2 \oplus S_0 = 0x80808000 \oplus \Delta S_1(a \oplus \Delta S_1(a)) \oplus a \oplus \Delta S_1'(a) \oplus \Delta S_2 S_1(a).$$

Since from the tables we know the possible values for $\Delta S_1(a \oplus S_1(a))$, $\Delta S_1'(a)$ and $\Delta S_2 S_1(a)$, combining them we have N possible values for z. Following the key-recovery approach of [7], by running the algorithm again with $IV'' \neq IV$ we make a new candidate set of values. Finally, we intersect them in order to obtain the actual value.

5 Active Sboxes

When applying the chosen IV differential attack on the versions of SNOW 3G and SNOW 2.0, we notice that the differences propagate identically for the first 13 rounds. This is not surprising since their algorithms are so similar, however it brings up the question how much more secure is SNOW 3G in terms of differential cryptanalysis? To answer this question we looked into the diffusion of the state, which can be measured by active Sboxes in the FSM.

We consider the branch number of the matrixes M_1 and M_2 used in S_1 and S_2 in order to evaluate the minimum number of active 8-bit Sboxes over 2 rounds. This metric is typically used in the security analysis of block ciphers, for example the AES [3].

The branch number provides a lower bound on the minimum byte weight of any two-round trail. For a matrix multiplication, such as $M \cdot (x_0, x_1, x_2, x_3)^T = (y_0, y_1, y_2, y_3)^T$, the branch number β is the minimum number of nonzero values among $x_0, x_1, x_2, x_3, y_0, y_1, y_2, y_3$.

Table 2: The propagation of the difference a for SNOW 3G$^{\oplus}$/2.0$^{\oplus}$, where (s_{15}, z) and $(s_{15}*, z^*)$ are based on Algorithms 1 and 2 respectively, and $a = 80000000; b = 80800000; c = 80008000; d = 80808000; e =$????$AA$; $f =$????$(A \oplus 0x80)A$.

SNOW 3G$^{\oplus}$

r	s0	s1	s2	s3	s4	s5	s6	s7	s8	s9	s10	s11	s12	s13	s14	s15	s15*	R1	R2	R3	ks	ks*
11	0	0	0	0	a	a	a	a	a	b	a	b	a	b	c	d	808000	a	0	0	00808000	????$(A \oplus 0x80)A$
12	0	0	0	a	a	a	a	a	b	a	b	a	b	c	d	0	800000	a	$\Delta S_1(a)$	0	????AA	????XX
13	0	0	a	a	a	a	a	b	a	b	a	b	c	d	0	e	80808000	$a \oplus \Delta S_1(a)$	$\Delta S_1'(a)$	$\Delta S_2 S_1(a)$	????$A'A'$	?
14	0	a	a	a	a	a	b	a	b	a	b	c	d	0	e	f	80800080	$a \oplus \Delta S_1'(a) \oplus \Delta S_2 S_1(a)$	$\Delta S_1(a \oplus S_1(a))$	$\Delta S_2 S_1'(a)$	?	?
15	a	a	a	a	a	b	a	b	a	b	c	d	0	e	f	?	d0????17	$b \oplus R_2^{14} \oplus R_3^{14}$	$\Delta S_1(R_1^{14})$	$\Delta S_2(R_2^{14})$	?	?

SNOW 2.0$^{\oplus}$

11	0	0	0	0	a	a	a	a	a	b	a	b	a	b	c	d	808000	a	0	-	00808000	????$(A \oplus 0x80)A$
12	0	0	0	a	a	a	a	a	b	a	b	a	b	c	d	0	800000	a	$\Delta S(a)$	-	????AA	????XX
13	0	0	a	a	a	a	a	b	a	b	a	b	c	d	0	e	80808000	$a \oplus \Delta S_1(a)$	$\Delta S_1'(a)$	-	????$A'A'$	?
14	0	a	a	a	a	a	b	a	b	a	b	c	d	0	e	f	80800080	$a \oplus \Delta S_1'(a)$	$\Delta S_1(a \oplus S_1(a))$	-	?	?
15	a	a	a	a	a	b	a	b	a	b	c	d	0	e	f	?	d0????17	$b \oplus \Delta S_1(a \oplus S_1(a))$	$\Delta S_1'(a \oplus S_1'(a))$	-	?	?

The branch number ranges from 2 (i.e., there is no diffusion) to the total number of bytes in the state plus 1. Formally described, having a difference between two inputs $a' = a \oplus b$ and w_t being the weight, we have:

Definition 5.1 *[3] The* differential branch number *of a linear transformation* $\lambda : \lambda(a) \oplus \lambda(b) = \lambda(a \oplus b)$ *is given by*

$$\mathfrak{B}_d(\lambda) = \min_{a' \neq 0} \{w_t(a') + w_t(\lambda(a'))\}$$

Let us consider each update of the LFSR and the FSM as one round of the algorithm. The matrices M_1 and M_2 are used in AES and are proven to have branch number 5, implying that if one byte is nonzero, after the matrix multiplication all 4 bytes will be nonzero. Since an 8-bit Sbox is applied on each byte before the matrix multiplication, then we have one active Sbox before and 4 active Sboxes after, giving a minimum of 5 active Sboxes for 2 rounds. Let us now look into the FSM and how the two Sboxes are applied. For the second register of SNOW 3G we have $R_2^{t+2} = S_1(R_2^t + (R_3^t \oplus s_5^t)$. This means that the bound derived from the branch number can be reached starting from two rounds after the difference enters the FSM. This is the case for SNOW 2.0 as well, since then we have $R_2^{t+2} = S_1(R_2^t + s_5^t)$. Figure 2 illustrates how the registers are connected and how the Sboxes are activated for SNOW 3G. Theoretically, it is possible the difference of R_2^t to be canceled by the addition of $R_3^t \oplus s_5^t$, leading to difference 0 for the register R_1^{t+1}. In this case the number of active Sboxes will be 0 in the following round.

Observation 5.1 *Over two rounds of SNOW 2.0 we have at least 5 active Sboxes starting from 2 rounds after the difference enters the FSM, unless the difference is canceled locally.*

Note that for the chosen IV difference here, the first round with nonzero difference of R_1 is round 11, however the active Sboxes are still 0. In round 12 and 13 we have one active Sbox each and in round 14 we have 4, since the input is the output of R_2^{12} added to s_5^{12}. We

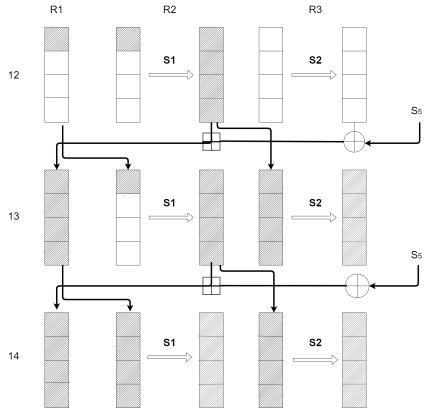

Figure 2: Three consecutive updates of the registers of SNOW 3G according to the chosen difference $a = 0x80000000$. The dark gray blocks indicate active Sboxes, while the light grey blocks indicate that at least one of them will be activated.

start counting the active Sboxes over two rounds from round 13, when the difference has reached all registers and one round has passed. Then over round 13 and 14 we have 5 active Sboxes for the chosen difference a. For any other 2 rounds we have at least 5 active Sboxes, unless we have 0 active Sboxes for one of the transitions.

Observation 5.2 *For two rounds of SNOW 3G we have at least 10 active Sboxes starting from 2 rounds after the difference enters the FSM, unless the difference is canceled locally.*

Again, the chosen difference a enters the first register at round 11, so over rounds 13 and 14 we have at least 10 active Sboxes, 5 from S_1 and at least 5 from S_2. This observation is valid for any other 2 consecutive rounds after round 13, unless we have 0 active Sboxes for one of the transitions.

The same observations are valid for the versions of SNOW $2.0^{\oplus}$ and SNOW $3G^{\oplus}$.

6 Conclusion and future work

In this work we adjusted the results of previous study without considering the tweak for easier analysis and extended the analysis by considering the compositions of Sboxes. Further, we looked into the diffusion of the FSM for several SNOW versions in terms of active Sboxes per 2 rounds. Even though they have the same security level until certain number of rounds, after that the diffusion of SNOW 3G is twice as fast as SNOW 2.0.

As a future work we will extend the analysis of composition of Sboxes in order to predict the keystream difference and use the results to perform key-recovery attacks over more rounds. Another research direction is to use the kyestream difference of several consecutive keystream words in order to gain information on the UEA2, since this algorithm uses SNOW 3G as one-time pad.

Acknowledgements. This work was supported by CyberSecurity Research Flanders with reference number VR20192203.In addition, this work was partially supported by the Research Council KU Leuven, C16/18/004, through the IF/C1 on New Block Cipher Structures and by the Flemish Government through FWO Project Locklock G0D3819N. The authors are funded by an FWO PhD and an FWO postdoc fellowship, respectively.

References

[1] O. Billet, H. Gilbert: "Resistance of SNOW 2.0 Against Algebraic Attacks", CT-RSA 2005, 19-28, 2005.

[2] A. Biryukov, D. Priemuth-Schmid, B. Zhang: "Multiset Collision Attacks on Reduced-Round SNOW 3G and SNOW 3G (+)", ACNS 2010: 139-153, 2010

[3] J. Daemen, V. Rijmen: "The Design of Rijndael: AES - The Advanced Encryption Standard", Information Security and Cryptography, Springer 2002

[4] P. Ekdahl, T. Johansson: "A new version of the stream cipher SNOW", SAC 2002, pp. 47-61, 2003

[5] P. Ekdahl, T. Johansson, A. Maximov, J. Yang: "A new SNOW stream cipher called SNOW-V", IACR Trans. Symmetric Cryptol. 2019(3): 1-42, 2019

[6] J.-K. Lee, D.-H. Lee, S. Park: "Cryptanalysis of SOSEMANUK and SNOW 2.0 using linear masks", ASIACRYPT 2008, pp. 524-538, 2008

[7] S. Ma, J. Guan: "Differential attacks on reduced-round SNOW 3G and SNOW 3G(+)", IET Inf. Secur., 14(5):587–594, 2020.

[8] K. Nyberg, J. Wall en: "Improved linear distinguishers for SNOW 2.0", FSE 2006, pp. 144-162, 2006

[9] Specification of the 3GPP Confidentiality and Integrity Algorithms UEA2 & UIA2. Document 5: Design and Evaluation Report, 2006

[10] Specification of the 3GPP Confidentiality and Integrity Algorithms UEA2 & UIA2. Document 2: SNOW3G Specification, 2006. https://www.gsma.com/aboutus/wp-content/uploads/2014/12/snow3gspec.pdf

[11] D. Watanabe, A. Biryukov, C. De Cannière: "A Distinguishing Attack of SNOW 2.0 with Linear Masking Method", SAC 2003, pp. 222-233, 2003

[12] J. Yang, T. Johansson, A. Maximov: "Vectorized linear approximations for attacks on SNOW 3G", IACR Trans. Symmetric Cryptol. 2019(4): 249-271, 2019

[13] B. Zhang, C. Xu, W. Meier: "Fast Correlation Attacks over Extension Fields, Large-Unit Linear Approximation and Cryptanalysis of SNOW 2.0", CRYPTO (1) 2015, pp. 643-662, 2015

Chapter 10

Cryptanalysis of the permutation based algorithm SpoC

This paper was published at INDOCRYPT 2021. Additionally, the results were presented at Central European Conference on Cryptology (CECC) 2020 and at the NIST Lightweight Cryptography Workshop 2020. All presentations were given online by me.

I have contributed equally with Raluca Posteuca to the development of the attack ideas, implementations and writing of the paper. All aspects of this research were supervised by prof. Vincent Rijmen, who helped with fruitful discussions.

Cryptanalysis of the permutation based algorithm SpoC

Liliya Kraleva[1], Raluca Posteuca[1], and Vincent Rijmen[1,2]

[1] imec-COSIC, KU Leuven, Leuven, Belgium,
[2] Department of Informations, University of Bergen, Norway
[liliya.kraleva, raluca.posteuca, vincent.rijmen]@esat.kuleuven.be

Abstract. In this paper we present an analysis of the SpoC cipher, a second round candidate of the NIST Lightweight Crypto Standardization process. First we present a differential analysis on the sLiSCP-light permutation, a core element of SpoC. Then we propose a series of attacks on both versions of SpoC, namely round-reduced differential tag forgery and message recovery attacks in the related-key, related-nonce scenario, as well as a time-memory trade-off key-recovery attack on the full round version of Spoc-64. Finally, we present an observation regarding the constants used in the sLiSCP-light permutation.

Keywords: SpoC · sLiSCP permutation · lightweight · differential cryptanalysis · TMTO attack · NIST lightweight competition · lwc

1 Introduction

The majority of the current standards in symmetric cryptography were initially designed for desktop and server environments. The increasing development of technology in the area of constrained environments (RFID tags, industrial controllers, sensor nodes, smart cards) requires the design of new, lightweight primitives. For this reason, NIST organised a competition aiming at standardizing a portfolio of lightweight algorithms, targeting authenticated encryption with associated data (AEAD) ciphers and hash functions. Currently the competition is at the second round with 32 out of 56 candidates left. The candidates of the NIST Lightweight Competition [NIS19] need to satisfy certain criteria for performance and have a level of security of at least 112 bits.

In order to contribute to the public research efforts in analysing the candidates of the on-going second round, we focused on the SpoC cipher, a permutation based AEAD. In this paper we present the results of a security research on both versions of SpoC, namely SpoC-64 and SpoC-128. We analyse the sLiSCP-light permutation used in the algorithm, as well as the structural behaviours of SpoC-64.

1.1 Related work

In [LSSW18] the authors describe a differential analysis of reduced-step sLiSCP permutation, a previous version of sLiSCP-light, in the AE and hashing modes.

This paper presents forgery attacks on 6 (out of 18) steps of the permutation. Their approach is similar to the one presented in this paper, even though we aim at analysing the newer version of the permutation, namely the sLiSCP-light permutation and its application in the SpoC cipher.

In parallel with our research, an open source paper analyzing the sLiSCP and sLiSCP-light permutations was published. In [HNS20] the authors use an approach based on Limited birthday distinguishers, improving the attacks against the sLiSCP permutation and proving the lower bound of the complexity to solve LBDs for a random permutation.

1.2 Our contribution

In this paper we present differential characteristics for round reduced versions of both sLiSCP-light-[256] and sLiSCP-light-[192]. For the former a 6 out of 18 rounds characteristic is given with probability $2^{-106.14}$ and for the latter a 7 round and 9 (out of 18) round characteristics with probabilities $2^{-108.2}$ and $2^{-110.84}$ respectively are presented. Based on the described characteristic, we introduce selective tag forgery attacks for both versions of round-reduced SpoC and a message recovery attack on 9-round SpoC-64. Additionally, a key-recovery attack is introduced for the full round version of SpoC-64, based on a time-memory trade-off approach. Table 1 summarizes our attacks and their complexities. To the best of our knowledge, this is the first research that analyses the security of the sLiSCP-light permutation and the first published results on SpoC.

Table 1: All attacks on SpoC. We underline that the data complexity of the key-recovery attack corresponds to the success probability 2^{-15}

Attack	steps of π	data	time	memory	section
Tag forgery on SpoC-128	6	$2^{106.14}$	$2^{107.14^*}$	-	4.2
Tag forgery on SpoC-64	7	$2^{108.2}$	$2^{109.2^*}$	-	4.3
Message recovery on SpoC-64	9	$2^{110.84}$	$2^{109.84^{**}}$	-	4.4
Key recovery on SpoC-64	all	2^{67}	2^{110}	$2^{110^{***}}$	5

* SBox computations, ** table look ups, *** table entries

1.3 Structure

The paper is structured as follows: In section 2 we briefly introduce the tools used in our research, the SpoC cipher and the sLiSCP-light permutation. In

section 3 we present round-reduced differential characteristics of both versions of the sLiSCP-light permutation. Section 4 introduces tag-forgery and message recovery attacks on the SpoC cipher parameterized with round-reduced sLiSCP-light permutations, while in section 5 is introduced a TMTO key-recovery attack on full Spoc-64. Section 6 presents an observation regarding the generation algorithm of the sLiSCP-light constants. Finally, the last section concludes the paper.

2 Preliminaries

This section gives some essential aspects of differential cryptanalysis and how to estimate the differential probability. Additionally, the algorithms of SpoC and sLiSCP-light are presented.

2.1 Differential cryptanalysis

Differential cryptanalysis is one of the most powerful and used cryptanalysis techniques against symmetric primitives. Proposed by Biham and Shamir [BS90], this approach was introduced as the first attack that fully breaks the DES algorithm [NIS79]. This attack has been largely analysed and further developed, leading to attacks like Truncated differentials [Knu94], the Boomerang attack [Wag99] or Impossible differentials [BBS99].

A differential characteristic over n rounds of an iterated cipher is defined as the sequence of intermediate differences $(a = a_1, a_2 \dots a_n = b)$, where each (a_i, a_{i+1}) represents the input and output difference of one round. Assuming that the rounds are independent and the keys are uniformly distributed, then the Expected Differential Probability (EDP) of a characteristic is computed by multiplying the single rounds' probabilities.

$$EDP(a_1, a_2, \dots a_n) = \prod_{i}^{n} DP(a_{i-1}, a_i),$$

where DP represents the differential probability for one round. For simplicity, in this paper we also use the notion of weight of a differential characteristic, instead of probability, computed as the absolute value of the $\log_2$ of the probability.

In practice, only the input and output differences can be observed without considering the intermediate differences. We define the set of all differential characteristics with input difference a and output difference b as a differential (a, b). The probability of a differential (a, b) is computed as the sum of the probabilities of all differential characteristics contained by it and is thus a hard problem. Therefore, the probability of the optimal characteristic serves as a lower bound for the probability of the differential.

2.2 SAT solvers

Nowadays, the security against differential cryptanalysis of ARX ciphers is analysed using automated tools such as SAT (Booolean Satisfability) or MILP (Mixed -Integer Linear Programming) solvers.

A SAT solver determines whether a boolean formula is satisfiable. Many operations like modular addition, rotation and XOR can be written as simple equations and easily translated to boolean expressions. For ARX ciphers several automatic search tools have been developed, see for example [Ste14, MP13, Ran17]. We chose to use the ARXpy tool [Ran17, RAS+20], because of its easy to use implementation, complete documentation and open-source code.

2.3 Specifications of SpoC

SpoC, or Sponge with masked Capacity [AGH+19], is a permutation based mode of operation for authenticated encryption. Since it is a sponge construction, its b-bit state is divided into rate and capacity bits with r the size of the rate and $c = b - r$ the size of the capacity. The authors introduced a masked capacity part with size r, representing the blocks in which the message and AD are added.

Two main versions are defined, namely SpoC-64_sLiSCP-light-[192] and SpoC-128_sLiSCP-light-[256], where 64 and 128 represent the corresponding size of the rate bits and sLiSCP-light-[] is the permutation used. Throughout this paper we refer to each of the SpoC versions as SpoC-r, while the permutation is also denoted by π. The key and nonce sizes of both versions are 128 bits. The following table describes the bit size of the parameters of both versions:

Instance	state b	rate r	tag t	SB size	rounds u	steps s
SpoC-64_sLiSCP-light-[192]	192	64	64	48	6	18
SpoC-128_sLiSCP-light-[256]	256	128	128	64	8	18

The state of SpoC is divided into 4 subblocks $(S_0||S_1||S_2||S_3)$ with equal length, where $||$ defines the concatenation. The encryption process of SpoC contains the *Initialization*, *Associated Data Processing*, *Plaintext Processing* and *Tag Processing* phases and is shown in Figure 1.

Initialization. This is the only step that is different for the two versions of SpoC. For SpoC-128, this process consists only of loading the key $K = K_0||K_1$ and nonce $N = N_0||N_1$ to the state into the odd and even numbered subblocks respectively. That is, $S_1[j] \leftarrow K_0[j], S_3[j] \leftarrow K_1[j], S_0[j] \leftarrow N_0[j], S_2[j] \leftarrow N_1[j]$, where $0 < j < 7$ and $X[j]$ represents the jth byte of the block X.

All capacity bits are considered as masked.

In SpoC-64 the state is smaller, while the sizes of the key and nonce are unchanged. First, K and N_0 are loaded to the state and then the permutation sliSCP-light-[192] is applied. Finally, N_1 is added to the masked capacity bits. The rate part is represented by the first 4 bytes of subblocks S_0 and S_2 and is where N_0 is loaded, while the key is loaded to the rest of the state.

AD and Message processing. This two phases are very similar. The blocks of AD, respectively message, are absorbed into the state and after each block

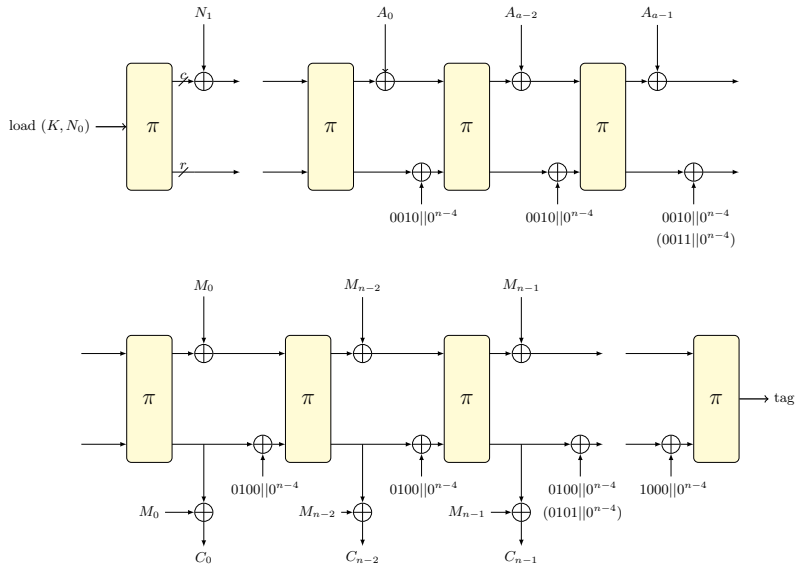

Fig. 1: Schematic diagram of the different phases used in the SpoC cipher. The shown initialization is for SpoC-64. Note that the state is divided in two parts, the capacity (the upper part) and the rate (the lower part).

a control signal (also referred to as a constant) is added. Finally, π is applied to the state. The added constant depends on both the phase and the length of the block. For the AD processing phase, the added constant is 0010 for a full block and 0011 for a partial (last) block. Respectively, for the message processing phase, 0100 is added for a full block and 0101 for a partial block.

In both AD and message processing phases, in the case of an incomplete last block, a padding function is applied. We denote this by $padding(M) = M||1||0^{m_c-n-1}$ for a block M of length $n < m_c$, where m_c is the length of a full block (which is, in fact, the length of the masked capacity part).

Tag processing. The control signal in this phase is set to 1000 and is added right after the control signal of the previous phase. Then π is applied and the tag is extracted from S_1 and S_3 of the output state.

Note, that if there is null AD (respectively, null message), the corresponding phase is entirely omitted. We denote with (AD, M) an associated data and message pair and with "" an empty instance of either of them. For example, $("", M)$ denotes an input pair having null associated data.

2.4 Specifications of sLiSCP-light

sLiSCP-light-[b] is a permutation that operates on a b-bit state, where b equals 192 or 256, and is defined by repeating a step function s times. The state is

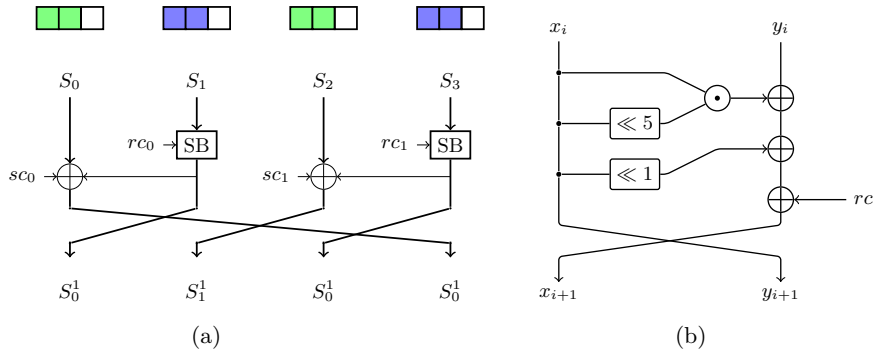

Fig. 2: The sLiSCP-light step function is shown in (a). On top, the blue blocks represent masked capacity bits and the green represent the rate bits for SpoC-64. (b) represents one round of the Simeck cipher, used as SB.

divided into four $2m$-bit subblocks $(S_0^i||S_1^i||S_2^i||S_3^i)$, with $0 < i < s-1$ the step number and m being equal to 24 or 32. We denote a zero subblock as 0^{2m} or simply 0, where the number of zero bits is clear from the context. The following 3 transformations are performed: *SubstituteSubblocks(SSb)*, *AddStepconstants(ASc)*, and *MixSubblocks(MSb)*. An ilustration is shown in Figure 2a.

The SSb transformation is a partial substitution layer, in which a non-linear operation is applied to half of the state - in subblocks S_1 and S_3. The non-linear operation, or the SBox, is represented by an u-round iterated unkeyed Simeck-$2m$ block cipher [YZS+15].

The Simeck SBox used in the description of the permutation is depicted in Figure 2b. For an input (x_i, y_i) of the ith round, the output is

$$R(x_{i+1}, y_{i+1}) = (y_i \oplus f(x_i) \oplus rc, x_i),$$

where $f(x) = (x \odot (x \ll 5) \oplus (x \ll 1))$ and rc represents a constant computed using the sLiSCP-light's round constants rc_i.

The ASc layer is also applied to half of the state as the step constants sc_1^i and sc_2^i are applied to subblocks S_0^i and S_2^i respectively. Table 2 lists all round and step constants used in sLiSCP-light.

The reader can refer to [AGH+19] or [ARH+18] for more details regarding the description of SpoC and the sLiSCP-light permutation.

2.5 Security claims and the impact of our attacks

In the original paper, the authors of SpoC introduced the security claims in different manners. In this paper, we refer to the security claims described in Table 3.1 from [AGH+19], since we consider this to be the most restrictive. Our interpretation of this table is that the best attack on either version of SpoC uses at most 2^{50} data encrypted under the same key and has a time complexity of at

Table 2
Round and step constants for sLiSCP-light-[192]:

step i	(rc_0^i, rc_1^i)	(sc_0^i, sc_1^i)
0-5	(7,27), (4,34), (6,2e), (25,19), (17,35), (1c,f)	(8, 29), (c, 1d), (a, 33), (2f, 2a), (38, 1f), (24, 10)
6-11	(12, 8), (3b, c), (26, a), (15, 2f), (3f, 38), (20, 24)	(36, 18), (d, 14), (2b, 1e), (3e, 31), (1, 9), (21, 2d)
12-17	(30, 36), (28, d), (3c, 2b), (22, 3e), (13, 1), (1a, 21)	(11, 1b), (39, 16), (5, 3d), (27, 3), (34, 2), (2e, 23)

Round and step constants for sLiSCP-light-[256]:

step i	(rc_0^i, rc_1^i)	(sc_0^i, sc_1^i)
0-5	(f, 47), (4, b2), (43, b5), (f1, 37), (44, 96), (73, ee)	(8, 64), (86, 6b), (e2, 6f), (89, 2c), (e6, dd), (ca, 99)
6-11	(e5, 4c), (b, f5), (47, 7), (b2, 82), (b5, a1), (37, 78)	(17, ea), (8e, 0f), (64, 04), (6b, 43), (6f, f1), (2c, 44)
12-17	(96, a2), (ee, b9), (4c, f2), (f5, 85), (7, 23), (82, d9)	(dd, 73), (99, e5), (ea, 0b), (0f, 47), (04, b2), (43, b5)

most 2^{112}. Furthermore, the attack, aiming at either breaking the confidentiality or the integrity of the cipher, has a success probability of at least 2^{-16}.

In all of our differential-based attacks, each encryption is performed under different key-nonce pairs, whereas in the online phase of the key-recovery attack the adversary intercepts messages encrypted by an honest user.

The impact of our attacks. Since all of our differential-based attacks are applied to round-reduced versions of SpoC, we consider that they do not have an impact on the security of the SpoC cipher. Nonetheless, we consider this work relevant because it presents an analysis on the SpoC cipher and on the sLiSCP-light permutation that improves the knowledge about the security of both.

Regarding our key-recovery attack, since we were able to apply a generic Time-Memory Trade-Off on the full SpoC-64, we consider that there might be some undesirable properties inducing vulnerabilities in the mode of operation of SpoC-64. Since the initialization function of SpoC-64 is not bijective, different key-nonce pairs might lead to a collision on the internal state after initialization. This fact can be compared to the nonce misuse scenario since, in both cases, the same initialized state is used more than once. Therefore, there is no distinction, with respect to the ciphertext and tag, between encrypting twice with the same (key, nonce) pair and encrypting with two (key,nonce) pairs that collide after the initialization phase.

3 Differential cryptanalysis on sLiSCP-light

In this section we present several characteristics for both versions of the sLiSCP-light permutation. Our characteristics are constructed by imposing specific constraints on the input and output differences in order to be further used in a series of attacks on SpoC. First, the details for our 6 round characteristic on sLiSCP-light-[256] are presented, then 2 different characteristics over 7 and 9 rounds of sLiSCP-light-[192] are shown.

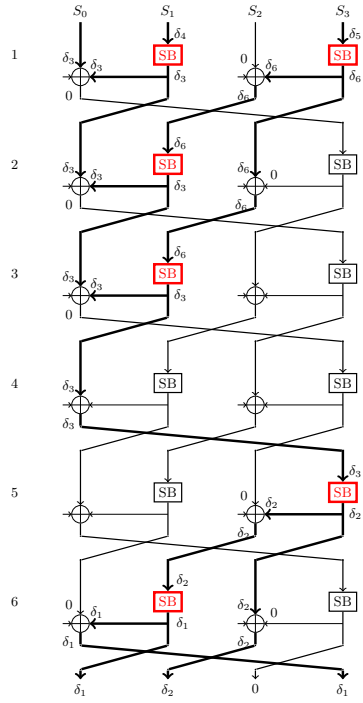

Fig. 3: The active SBoxes for 6 rounds of sLiSCP-[256] with the difference propagation

3.1 Characteristics on sLiSCP-light-[256]

In order to construct this 6 rounds characteristic of sLiSCP-light-[256] we impose a constraint only on the output difference and thus we construct it backwards. We fix the output difference in the first and third subblocks to $\Delta S_0^6 = \delta_1$ and $\Delta S_2^6 = 0$. In fact, those are the positions of the rate bits when applied to SpoC. The purpose and value of δ_1 is discussed in section 4. The output difference in subblocks S_1^6 and S_3^6 is irrelevant for our attacks, thus can take any value. However, in order to decrease the number of active Sboxes, we choose $\Delta S_1^6 = \delta_2$ and $\Delta S_3^6 = \delta_1$, where δ_2 is a possible input difference for the Sbox that leads to δ_1. In this case, the characteristic has only one active Sbox in each of the last two rounds and none in the 4th one.

As it can be seen in Figure 3, for 6 rounds of the permutation we have 6 active Sboxes with the following transitions:

$$\delta_5 \xrightarrow{SB} \delta_6 \xrightarrow{SB} \delta_3 \xrightarrow{SB} \delta_2 \xrightarrow{SB} \delta_1, \qquad \delta_4 \xrightarrow{SB} \delta_3.$$

The resulted characteristic has the input difference $\delta_3||\delta_4||0^{64}||\delta_5$ and the output difference $\delta_1||\delta_2||0^{64}||\delta_1$, therefore, by fixing the input and the output

of the characteristic we actually fix the differences from δ_1 to δ_5. However, the difference δ_6 can take multiple values, as long as it is a valid output difference for the input difference δ_5 and is a valid input difference for the output difference δ_3. In the case of our characteristic we take $\delta_6 = \delta_4$.

To choose the differences δ_i we used the automatic search tool ArxPy [Ran17]. By fixing δ_1 to one of the possible values described in section 4, we constructed a tree of differences. The nodes of this tree represent all possible input differences returned by ArxPy, having weight less than the optimal one plus 3.

After choosing the appropriate differences, the weight of each transition was empirically verified using 2^{30} data. In order to obtain the final weight of the characteristic all weights of the SBox transitions were added up. Our best differential characteristics and their corresponding weights are listed in Table 3. The best weight that we found for 6 rounds of the permutation is 106.14.

Note that the optimal characteristic over one SBox (8 rounds of Simeck) has weight 18. This is proven in [LLW17] and verified by us with the ArxPy tool. The empirical differential probability of 8 rounds of Simeck has slightly lower or higher weight than the optimal characteristic, which is expected due to the differential effect and the independency assumptions.

3.2 sLiSCP-light[192]

In this section we present 2 characteristics, one over 7 rounds and one over 9 rounds of sLiSCP-light-[192]. They are constructed in a similar way as the characteristic described in the previous subsection. The requirements of the desired differences in the input and output bit positions are different, since the constraints imposed by our attack scenarios are different.

7-round characteristic. The constraints of our 7-round characteristic fix the nonzero input difference to S_1^0 and S_3^0, while the output difference is $S_0^7 = \delta_1$ and $S_2^7 = 0^{48}$. The difference in S_1^7 and S_3^7 is chosen for convenience, to reduce the number of active SBoxes. Therefore, our characteristic has 7 active Sboxes, as shown in Figure 4a. The described characteristic has input and output differences

Table 3: The best characteristics we found for sLiSCP-[256] are shown in this table. For simplicity the 64-bit differences are presented in hexadecimal and separated in two halves and the ".." denote 3 zero bytes. The values above the arrows represent the weight of each transition's differential.

δ_5		$\delta_4 = \delta_6$		δ_3		δ_2		δ_1	w
1..0, 1..0	$\xrightarrow{17.69}$	1..1, 0..0	$\xrightarrow{17.69}$	1..0, 0..0	$\xrightarrow{17.69}$	1..1, 0..0	$\xrightarrow{17.69}$	1..0, 0..0	106.14
1..0, 0..0	$\xrightarrow{17.69}$	1..1, 0..0	$\xrightarrow{17.69}$	1..0, 0..0	$\xrightarrow{17.69}$	1..1, 0..0	$\xrightarrow{17.69}$	1..0, 0..0	106.14
1..0, 0..2	$\xrightarrow{18.69}$	1..1, 0..0	$\xrightarrow{17.69}$	1..0, 0..0	$\xrightarrow{17.69}$	1..1, 0..0	$\xrightarrow{17.69}$	1..0, 0..0	107.14
0..0, 2..0	$\xrightarrow{18.30}$	0..0, 2..2	$\xrightarrow{18.30}$	0..0, 2..0	$\xrightarrow{18.30}$	0..0, 2..2	$\xrightarrow{18.90}$	6..0, 0..0	110.4

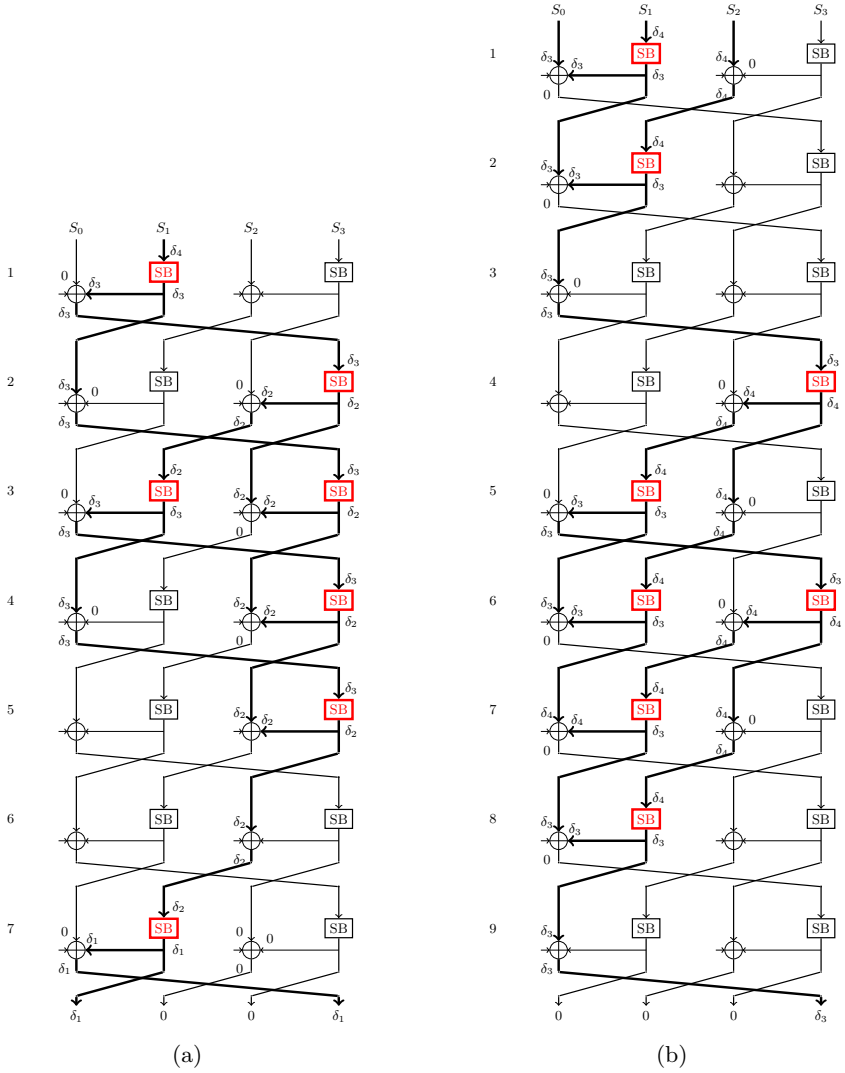

Fig. 4: Differential characteristics of a) 7 rounds and b) 9 rounds of sLiSCP-light-[192]

$(0^{48}||\delta_4||0^{48}||0^{48} \rightarrow \delta_1||0^{48}||0^{48}||\delta_1)$, with

$$\delta_4 \xrightarrow{SB} \delta_3 \xrightarrow{SB} \delta_2 \xrightarrow{SB} \delta_1, \quad \delta_2 \xrightarrow{SB} \delta_3.$$

From Figure 4a we can see that the iterative transition $\delta_3 \xrightarrow{SB} \delta_2$ happens 4 times, while $\delta_2 \xrightarrow{SB} \delta_3$ appears only once. The exact values are chosen to minimize the weight of $\delta_3 \xrightarrow{SB} \delta_2$. Our best probability of $2^{-108.2}$ happens for

$$\delta_1 = 0x700000000000, \delta_2 = 0x500001800000,$$
$$\delta_3 = 0x100000200000, \delta_4 = 0x100001100000.$$

with weights

$$\delta_4 \xrightarrow{17} \delta_3 \xrightarrow{11,3} \delta_2 \xrightarrow{23.1} \delta_1, \quad \delta_2 \xrightarrow{22.9} \delta_3.$$

Note that the optimal characteristic over one SBox (6 rounds of Simeck) has weight 12, as verified with the ArxPy tool.

9 round characteristic. For this characteristic we fix the non-zero output difference to S_1^9 and S_3^9. More precisely, in the first 4 bytes of the subblocks, which correspond to the masked capacity bits in SpoC.

In order to design this characteristic we used an iterative transition $\delta_3 \xrightarrow{SB} \delta_4 \xrightarrow{SB} \delta_3$. The input and output differences of our characteristic are $(\delta_3||\delta_4||\delta_4||0^{48})$ and $(0^{48}||0^{48}||0^{48}||\delta_3)$, respectively.

As seen in Figure 4b, the characteristic has 8 active SBoxes with $\delta_3 \xrightarrow{SB} \delta_4$ appearing 6 times and $\delta_4 \xrightarrow{SB} \delta_3$ two times. Our best probability of $2^{-108.5}$ holds for the differences $\delta_3 = 0x000a00000400$ and $\delta_4 = 0x000a00001000$ with weights

$$\delta_3 \xrightarrow{22.34} \delta_4 \xrightarrow{10.65} \delta_3.$$

4 Differential attacks on SpoC-128 and SpoC-64

In this section we present a series of attacks based on the differential characteristics introduced in the previous section. More precisely, we design tag-forgery attacks based on the 6-round characteristic of sLiSCP-light-[256] and the 7-round characteristic of sLiSCP-light-[192]. The 9-round characteristic of sLiSCP-light[192] is used to design a message-recovery attack.

4.1 Tag forgery attacks

As stated in Section 2.3, in order to distinguish between different phases of the encryption process, the authors of SpoC used a 4-bit control signal. The values of these 4 bits depend on the current phase, but also on whether the inputs (associated data or plaintext) are padded or not. Moreover, if the associated data or the plaintext is null, the corresponding phase is disregarded.

Our approach is based on identifying and exploiting scenarios in which different types of inputs lead to similar internal states.

Take, for example, the scenario where one uses SpoC-64 or SpoC-128 to encrypt two one-block plaintexts: an incomplete block M and a complete block $M^* = padding(M)$, using, in both cases, the same associated data and the same (key, nonce) pair. Take into account that, at the end of the plaintext addition phase, just before generating the tag, the difference between the corresponding internal states is given by the difference between the corresponding 4-bit control signals, i.e. $0001||0^{n-4}$. In the plaintext processing phase of the first case, the 0101 control signal is used, while in the second case 0100 is used. The difference between the used constants we denote by δ_1 and it represents the convenient difference that we can cancel locally.

The difference δ_1. Depending on the scenario, we identified three possible values for the control signals' difference δ_1, as follows:

1. $\delta_1 = 0001||0^{n-4} = 0100||0^{n-4} \oplus 0101||0^{n-4}$
   This value can be obtained in the case when we encrypt the plaintexts M and M^* described above, using the same (key, nonce) pair and the same AD.
2. $\delta_1 = 0110||0^{n-4} = 0100||0^{n-4} \oplus 0010||0^{n-4}$
   This value can be obtained when we encrypt $(\text{""}, M)$ and $(M, \text{""})$. More precisely, in the first case we use a null AD, while in the second case we use a null plaintext. The former encryption consist of initialization, message processing phase and finalization, whereas the latter has AD processing phase instead of message processing. It will produce no ciphertext, however the tags of the two would be the same. Hence we can forge the verification of associated data.
3. $\delta_1 = 0111|0^{n-4} = 0101||0^{n-4} \oplus 0010||0^{n-4}$
   This value can be obtained when we encrypt the pairs $(\text{""}, M)$ and $(AD, \text{""})$, where the length of M is less than the length of a full block and $AD = padding(M)$.

In order to achieve a tag forgery, we designed differential characteristics such that, after the plaintext processing phase, the difference between the corresponding control signals is cancelled by the output difference of the characteristic. Since this results in the same internal states, the corresponding tags will collide.

We underline the fact that the control signal bits are influencing the difference on the rate part of the internal state. The target characteristic might also have active bits in the capacity part, these being canceled through a difference between the two plaintexts. Therefore, we aim at finding characteristics having the output difference of the form $(\delta, \lambda, 0, \gamma)$, where δ is the difference between the constants, while the differences λ and γ can be cancelled through the plaintext block difference. In our experiments, in order to optimize the number of active Sboxes, we imposed the additional constraint that $\delta = \gamma$. In section 3 we presented the best characteristics that we found, suitable for our approach, on 6-round sLiSCP-256 and on 7-round sLiSCP-192. Using these characteristics and the approach presented above, we designed tag forgery attack on reduced versions of both Spoc-64 and Spoc-128.

Since the complexity of our round-reduced characteristics are close to the security bound, we chose the input parameters such that the difference propagates through only one permutation.

4.2 Tag forgery attack on SpoC-128

After we have fixed our characteristic, we can proceed to the attack. Note that, in the case of SpoC-128, the initialization phase is represented only by the loading of the (key, nonce) pair into the internal state. Since there is only one sLiSCP-light application before the ciphertext generation, our tag forgery attack on SpoC-128 follows the related-key related-nonce scenario. According to our 6-round characteristic, we use inputs such that the key difference is $(\delta_4||\delta_5)$, while the nonce difference is $(\delta_3||0)$. Moreover, since our best characteristic uses $\delta_1 = 0x1||0^{124}$, the setup of this attack assumes the use of null associated data, a plaintext $M = M_1||M_2$ having the size less than 128 bits and a plaintext $M^* = padding(M) \oplus \delta_2||\delta_1 = M_1^*||M_2^*$ encrypted under related-key related-nonce pairs. As we mentioned before, by injecting a difference in the plaintexts we cancel the capacity difference after the permutation. The encryption processes are described in Figure 5.

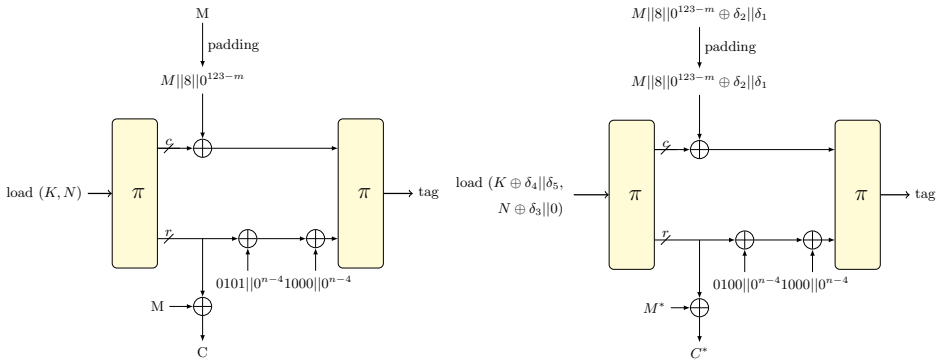

Fig. 5: The encryption of an incomplete and a full block of plaintexts, using related-key related-nonce inputs and null associated data

If our differential characteristic holds, then the following equations also hold with probability 1:

$$M_1^* = M_1 \oplus \delta_2, \quad S_0^{6*} = S_0^6 \oplus \delta_1, \quad S_2^{6*} = S_2^6, \; C^* = M_1^* \oplus S_0^{6*}||M_2^* \oplus S_2^{6*}.$$

Therefore, $C^* = (C_1||C_2) \oplus (\delta_2 \oplus \delta_1||\delta_1)$.

More precisely, the ciphertext-tag pair (C^*, τ) would then be valid under $(K \oplus \Delta_K, N \oplus \Delta_N) = (K \oplus \delta_4||\delta_5, N \oplus \delta_3||0)$ with the probability of the characteristic. The pseudocode of this attack is presented in Algorithm 1. Note that it is not

Algorithm 1: The tag forgery attack on Spoc-128

Encryption Obtain $(C = C_1C_2, \tau)$, the encryption of $("", M)$ under arbitrary (K, N);
Compute $S_0^6 S_2^6 = padding(M) \oplus C$;
if $SB^{-1}(S_0^6) \oplus SB^{-1}(S_0^6 \oplus \delta_1) == \delta_2$ **then**
  Decryption Ask for P, the decryption of $(C_1 \oplus \delta_2 \oplus \delta_1 || C_2 \oplus \delta_1, \tau)$ under $(K \oplus \delta_4 || \delta_5, N \oplus \delta_3 || 0)$;
  if $P \neq \perp$ **then**
    $P = ("", M^*)$;
  else
    go to Encryption;
  end
else
  go to Encryption;
end

needed to use the same key for each Encryption step of the algorithm. Since the rate part of the internal state is used for the encryption, by knowing both the plaintext and the ciphertext we can recover the rate part of each internal state used in the plaintext processing phase. In the case of our approach we use this observation to decrease the data complexity of our attack, by filtering the ciphertexts obtained in the first step.

The data complexity of this attack, computed as the number of encryptions and decryptions required, is $(P_{R_6} + 1) \cdot P_{R_1 \to R_5} = (2^{17.69} + 1) \cdot 2^{88.45} = 2^{106.14} + 2^{88.45}$, where R_i represents the ith sLiSCP step. The time complexity, computed as the number of offline SBox computations, is $2 \cdot P_{R_6} \cdot P_{R_1 \to R_5} = 2^{107.14}$.

Improved attack. We can improve the complexity of the attack by using multiple differential characteristics that have the same output difference, i.e. equal δ_1s and δ_2s. Suppose that we have d differential characteristics such that

$$\begin{matrix} \delta_3^i \\ \delta_4^i \\ 0 \\ \delta_5^i \end{matrix} \to \begin{matrix} \delta_3^i \\ \delta_4^i \\ \delta_4^i \\ 0 \end{matrix} \to \begin{matrix} \delta_3^i \\ \delta_4^i \\ 0 \\ 0 \end{matrix} \to \begin{matrix} \delta_3^i \\ 0 \\ 0 \\ 0 \end{matrix} \to \begin{matrix} 0 \\ 0 \\ 0 \\ \delta_3^i \end{matrix} \to \begin{matrix} 0 \\ \delta_2 \\ \delta_2 \\ 0 \end{matrix} \to \begin{matrix} \delta_1 \\ \delta_2 \\ 0 \\ \delta_1 \end{matrix},$$

where each characteristic has the probability p_i, $i = 1, \ldots, d$.

The attack follows the lines of Algorithm 1, where instead of asking for the decryption under a fixed (key, nonce) pair, we use every $(K \oplus \Delta_K^i, N \oplus \Delta_N^i)$ pair. For our improved attack the time and data complexities can improve with at most a factor of $log_2 d$, when all the characteristics have the same probability.

By using 10 different characteristics that we found that have $\delta_1 = 1..0, 0..0$ and $\delta_2 = 1..1, 0..0$, the complexity is improved by a factor of $2^{1.82}$, the time complexity being around $2^{105.32}$, while the data complexity decreases to approximately $2^{104.32} + 2^{86.63}$.

Time-Memory trade-off. The time complexity of our attack can be improved by using a time-memory trade-off approach. In this case the attack will

also imply an offline phase, as follows: for all possible values of S_0^6 we verify if $SB^{-1}(S_0^6) \oplus SB^{-1}(S_0^6 \oplus \delta_1) == \delta_2$. If the condition holds, we store the corresponding value of S_0^6 in the sorted list $list_{S_0^6}$.

The complexity of this phase is 2^{64} SBox computations. In this case, instead of verifying the specified condition, it will be verified if $S_0^6 \in list_{S_0^6}$. The time complexity of each query will be $log_2(\#list_{S_0^6})$ operations, while the memory complexity will be less than 2^{64} (negligible compared to the data complexity).

4.3 Tag forgery attack on SpoC-64

The main idea of the attack is similar to the one presented in subsection 4.2, some modifications being imposed due to the different loading phase of SpoC-64.

This attack is based on the 7-round characteristic presented in subsection 3.2. Since our characteristic covers only one permutation, in this scenario we have one more constraint on the input differences. More precisely, the setup of our attack assumes the use of related N_1s, while the key and the nonce N_0 are equal. The input difference is given by the difference between the corresponding N_1s, while the output difference respects the constraints from subsection 4.2. Since N_1 is added to the masked capacity bits, note that the difference needs to have active bits only in the masked capacity part.

Moreover, for the 7-round characteristic on sLiSCP-light-192 we used $\delta_1 = 0111||0^{n-4}$, therefore the setup of our attack is the one presented in Figure 6.

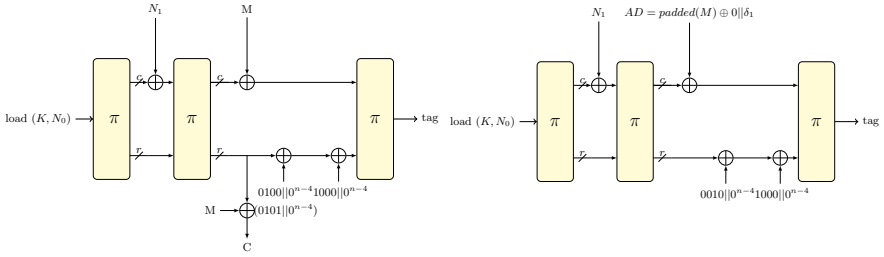

Fig. 6: The two processes of SpoC-64 used by our approach. Note that the second XORed constant is imposed by the beginning of the tag generation phase.

We state that, while the encryption of the message $(\text{""}, M)$ will return a (ciphertext, tag) pair, the encryption of the pair $(AD, \text{""})$ results in a null ciphertext and a tag. Therefore, assuming the 7-round characteristic holds, the ciphertext-tag pair $(\text{""}, \tau)$ is valid under $(K, N_0||(N_1 \oplus \Delta_N)) = (K, N_0||(N_1 \oplus \delta_4||0))$ with probability 1.

The tag forgery attack on SpoC-64 is very similar to the one described in Algorithm 1. The distinction is given by the input difference of the characteristic which impacts the decryption (key, nonce) pair. More precisely, only the nonce N_1 is different.

The data complexity of this attack, computed as the number of encryptions and decryptions required, is $(P_{R_7}+1)\cdot P_{R_1\to R_6} = (2^{23.1}+1)\cdot 2^{85.1} = 2^{108.2}+2^{85.1}$. The time complexity, computed as the number of offline Sbox computations, is $2 \cdot P_{R_7} \cdot P_{R_1\to R_6} = 2^{109.2}$.

Even though the required amount of data is higher than the size of the tag space, we consider that our attack is meaningful since the authors of SpoC claim security of 112 bits for both confidentiality and integrity.

Time-Memory trade-off. By following the same time-memory trade-off approach presented in subsection 4.2, we can improve the time complexity of our attack. The complexity of the offline phase is also 2^{64} SBox computations. Thus, the time complexity of each query will be $log_2(\#list_{S_0^7})$ operations, while the memory complexity will be less than 2^{64} (considerably smaller compared to the data complexity).

4.4 Message recovery attack on SpoC-64

In this section we present a message recovery attack on Spoc-64 based on a differential cryptanalysis approach. This attack exploits the fact that the initialization phase is not a bijective function, since the input is 256 bits and the internal state is 192 bits. The analysis aims at constructing (key, nonce) pairs that lead to the same internal state after the initialization. Thus, we designed the 9-round differential characteristic on sLiSCP-light-[192] presented in section 3. More precisely, the constraint of our characteristic is that the output difference only affects the capacity part, this difference being canceled by a difference between the corresponding N_1's. Therefore our approach uses a key-related nonce-related scenario.

By using our 9-round characteristic on a round-reduced scenario, the internal states after the initialization collide. Therefore, the encryption of the same plaintext under different (key, nonce) pairs lead to identical ciphertexts and tags. Moreover, if we encrypt two messages with the same l first blocks, the corresponding l ciphertext blocks will also be the same.

We used this approach to design a related-key related-nonce attack on SpoC-64. The attack works as follows:

1. With a key-nonce pair (K, N) we ask for the encryption of an arbitrary, unknown plaintext M, using the associated data AD; we obtain the ciphertext-tag pair (C, τ);
2. We ask for the decryption of (C, τ) under $(K \oplus \Delta_K, N \oplus \Delta_N)$ and using the initial AD;
3. If the tag verification holds, we obtain the plaintext M'. If M' is a readable text, then $M' = M$ and the message is recovered.

We specify M' being readable, since there is always a probability that tags collide. As stated in section 3, the probability of our 9-round characteristic is $2^{-109.84}$. Since the data complexity defines the number of encryptions and decryptions, in our case the data complexity is $2^{110.84}$, while the time complexity is bounded by the data complexity.

5 Key-recovery attack on SpoC-64

In this section we generalise the approach described in subsection 4.4, by defining the notion of class-equivalence over the space of all (key, nonce) pairs. We then present a time-memory trade-off attack based on the class-equivalence that leads to the recovery of the secret key K.

Equivalence in the set of (key, nonce) pairs.

Definition 1. *The (key, nonce) pairs* (K^1, N^1) *and* (K^2, N^2) *are said to be in the same equivalence class (or simply equivalent) if the corresponding internal states, after the initialization phase, are equal.*

The number of equivalence classes is given by the number of all possible internal states of SpoC-64, namely 2^{192}. For each fixed internal state, one can consider all values of N_1 and can compute the associated (K, N_0) pairs by applying the inverse of the permutation. Therefore, each equivalence class is formed by 2^{64} (key, nonce) pairs.

Note that the encryption of the same message under equivalent (key, nonce) pairs results in equal ciphertexts and tags. Moreover, the decryption and tag verification of a (ciphertext, tag) pair can successfully be performed under any (key, nonce) pair belonging to the same equivalence class.

The key-recovery attack. Our attack consists of two phases: an offline and an online phase. In the offline phase, the adversary generates a table containing 2^{110} entries. Each entry contains a $(K, N_0||N_1)$ pair and the ciphertexts and tag obtained by applying SpoC-64 on a well chosen plaintext M, under the $(K, N_0||N_1)$ pair and a null AD.

The (key, nonce) pairs are generated such that they belong to different equivalence classes. More precisely, 2^{110} different internal states are generated by the adversary. For each state an arbitrary N_1 is chosen and, by XORing it to the internal state and by applying the inverse of the permutation, the (K, N_0) pair is computed. Using each $(K, N_0||N_1)$ pair, the adversary encrypts, using SpoC-64, a common short message M. Note that, in practice, depending on the nature of a correspondence, messages usually start with the same words or letters. For example, e-mails normally start with "Dear (*name*)," or "Hello (*name*),". By making this assumption, we choose a plaintext M to be a regularly used word or phrase of length l blocks. In our research we make the assumption that the full 18-round sLiSCP-light behaves as a random permutation, thus no particular properties can be observed. Therefore, we claim that $l = 3$ is the number of blocks of ciphertext that uniquely defines the equivalence class of the $(K, N_0||N_1)$ pairs.

We consider the encryption function of SpoC-64, defined using a fixed plaintext and considering as the input state the result of the initialization phase. On one hand, in order to have uniqueness, this function has to be injective. Therefore, since the length of the internal state is 192 bits and the length of one block of ciphertext is 64 bits, the minimum value of l is 3. On the other hand, by writing the system of bit-level equations of the targeted function, for l blocks of ciphertext we obtain $64 \times l$ equations using 192 variables. If this system does

not have an unique solution for $l = 3$, it means that the resulted equations are not independent, thus there are some particular properties of the sLiSCP-light permutations that could be further extended to an attack.

The pseudocode of the offline phase is presented in Algorithm 2.

Algorithm 2: Offline phase

```
list = null ;
choose M;
while list.length < 2^110 do
    sample internal_state;
    sample N_1;
    compute (K, N_0) = π^{-1}(internal_state ⊕ N_1) ;
    encrypt (C, τ) = SpoC-64(K, N_0||N_1, "", M) ;
    list.Add(K, N_0||N_1, C)
end
Result: list populated with 2^110 entries
```

Note that π^{-1} denotes the inverse of the full sLiSCP-light-[192] permutation. The resulted list is sorted with respect to the ciphertexts, using a hash table. The memory complexity of this phase is 2^{110} table entries, while the time complexity is 2^{110} SpoC-64 encryptions. Note that the steps of an encryption are not performed sequentially. Since the first step is to sample the internal state, the encryption can be performed without the initialization phase, while the initialization phase is performed backwards, by computing a (key, nonce) pair corresponding to a fixed internal state. Thus, by assuming that the permutation function and the inverse of the permutation function are equivalent time-wise, the time complexity of our offline phase is 2^{110} encryptions.

In the online phase the adversary intercepts the (ciphertext, tag) pairs encrypted by a valid user. For simplicity, we assume that the valid user used null associated data. We discuss in a paragraph below the case where the associated data is not null. For every intercepted ciphertext, the adversary verifies if the first l blocks belong to the table computed in the offline phase. Since a string of l blocks uniquely defines the equivalence class, a match means that the valid user encrypted the plaintext under a (key, nonce) pair that is in the same equivalence class with the pair $(K, N_0||N_1)$ extracted from the precomputed table. Moreover, the adversary can easily compute the internal state obtained after the initialization phase, using the $(K, N_0||N_1)$. Since the nonce N_1 is public, it can XOR N_1 to the internal state and, by applying the reverse of the permutation, the adversary can compute the user's key.

In the case where the valid user chooses a non-empty value for the associated data, the key-recovery works as follows:

1. The adversary verifies if the first l blocks of the ciphertext belongs to the table;

2. When a match is found, the adversary reverse the associated data addition phase; this action is allowed, since AD is a public value;
3. On the obtained internal state, the adversary XOR the N_1 and apply the inverse of the permutation.

Therefore, the adversary gains full control over the encryption of the valid user, being able to decrypt all the past and future communication in which the valid user used the recovered key. Moreover, the adversary gains the ability of impersonating the valid user, being able to generate (ciphertext, tag) pairs using the secret key of the valid user.

Since the adversary can control 2^{110} equivalence classes, through the precomputed table, the probability that an intercepted message belongs to the precomputed table is $2^{110-192} = 2^{-82}$. Thus, if the adversary intercepts 2^{67} (ciphertext, tag) pairs, the success probability of this attack is 2^{-15}, twice the probability claimed by the authors of SpoC. By increasing the amount of intercepted data, the success probability of the attack also increases.

The data complexity of the online phase is represented by the number of required online encryptions. So, for a success probability of 2^{-15}, the data complexity of the online phase is 2^{67}. Since the precomputed table is a sorted hash table, the search of a ciphertext has a time complexity of $O(1)$. Thus, the time complexity of the online phase is 2^{67} table lookups. For comparison, an exhaustive search attack with the same success probability of 2^{-15} would require 2^{113} data. Note that even though the online phase of the attack can be performed many times (e.g. the attack targets two or more valid users), the offline phase of the attack is only performed once. Thus, the time complexity of the offline phase can be overlooked in any application of the attack, except for the first one. Therefore, every other instance of the attack has a total time complexity of 2^{67}.

Note that our attack recovers only one of the secret keys used by the valid user.

6 Other observations

While analyzing the sLiSCP-light-256 permutation we noticed a particular property of both the round and the step constants. More precisely, using the notations introduced in [AGH+19], we noticed that

$$rc_0^i = rc_1^{i+8}, \forall i \in \{0, ...10\}$$
$$sc_0^i = sc_1^{i+8}, \forall i \in \{0, ...10\}$$

The design rationale behind the generation of these constants is described in [ARH+17]. The constants are computed using an LFSR with length 7 and the primitive polynomial $x^7 + x + 1$. The initial state of the LFSR is filled with seven bits of 1.

For the computation of the round constants, the LFSR runs continuously for $18 \times 2 \times 8$ steps. The first 16 bits of the returning string are: 1111111000000100. The constants rc_0^0 and rc_1^0 are computed by 2-decimation. More precisely, the

bits of rc_0^0 are the bits in odd positions of the string above while the bits of rc_1^0 are the bits from the even positions, both of them being read in an little-endian manner. Thus, $rc_0^0 = 00001111 = 0xF$ and $rc_1^0 = 01000111 = 0x47$.

Since the primitive polynomial has degree 7, it's period is $2^7 - 1 = 127$. Therefore, the $127 + n^{th}$ bit will be equal to the nth generated bit. In particular, the bits of rc_1^{8+n} are equal to the bits of rc_0^n. A similar approach is used for the computation of the step constants. In this case, after loading the initial state of the LFSR with seven bits of 1, 14 steps are performed (discarding the outputed bits). Then the same procedure is applied, thus, the same observation is also valid for the step constants.

The round and step constants of the sLiSCP-light-192 permutation are computed by a similar manner. But since both the constants and the LFSR length is 6, the 2-decimation does not influence the distribution of the bits through the constants. Note that the authors of the sLiSCP permutation claim that each 8-bit constant is different.

7 Conclusion and future work

Our work analyzes the SpoC cipher, a second round candidate of NIST Lightweight competition, and the permutation sLiSCP-light which represents one core component of the SpoC cipher. For both versions of SpoC, namely SpoC-128 and SpoC-64, we propose characteristics covering round-reduced versions of the permutation. We then use these characteristics to design tag-forgery and message-recovery attacks on SpoC parameterized with round-reduced versions of the sLiSCP-light permutation. Furthermore, by using an TMTO approach, we designed a key-recovery attack on SpoC-64. A summary of our results is depicted in Table 1.

The work we presented can be extended in several directions. For example, it would be interesting to analyse both the SpoC cipher and the sLiSCP-light permutation using other techniques. It also remains to be investigated if or how our observations regarding the round and step constants can be exploited. Further research should also consider investigating the impact of our characteristics to other ciphers based on the sLiSCP-light permutation.

8 Acknowledgements

The authors would like to thank Adrián Ranea for all the fruitful discussions regarding the ARXpy tool. This work was supported by CyberSecurity Research Flanders with reference number VR20192203 and partially supported by the Research Council KU Leuven, C16/18/004, through the EIT Health RAMSES project, through the IF/C1 on New Block Cipher Structures and by the Flemish Government through FWO Project Locklock G0D3819N.

Bibliography

[AGH+19] Riham AlTawy, Guang Gong, Morgan He, Ashwin Jha, Kalikinkar Mandal, Mridul Nandi, and Raghvendra Rohit. SpoC:An Authenticated Cipher Submission to the NIST LWC Competition. 2019. `https://csrc.nist.gov/CSRC/media/Projects/lightweight-cryptography/documents/round-2/spec-doc-rnd2/spoc-spec-round2.pdf`.

[ARH+17] Riham AlTawy, Raghvendra Rohit, Morgan He, Kalikinkar Mandal, Gangqiang Yang, and Guang Gong. sLiSCP: Simeck-Based Permutations for Lightweight Sponge Cryptographic Primitives. In *Selected Areas in Cryptography - SAC 2017 - 24th International Conference, Ottawa, ON, Canada, August 16-18, 2017, Revised Selected Papers*, volume 10719 of *Lecture Notes in Computer Science*, pages 129–150. Springer, 2017.

[ARH+18] Riham AlTawy, Raghvendra Rohit, Morgan He, Kalikinkar Mandal, Gangqiang Yang, and Guang Gong. SLISCP-light: Towards Hardware Optimized Sponge-specific Cryptographic Permutations. *ACM Trans. Embedded Comput. Syst.*, 17(4):81:1–81:26, 2018.

[BBS99] Eli Biham, Alex Biryukov, and Adi Shamir. Cryptanalysis of Skipjack Reduced to 31 Rounds Using Impossible Differentials. In *EUROCRYPT*, volume 1592 of *Lecture Notes in Computer Science*, pages 12–23. Springer, 1999.

[BS90] Eli Biham and Adi Shamir. Differential Cryptanalysis of DES-like Cryptosystems. In *Advances in Cryptology - CRYPTO '90, 10th Annual International Cryptology Conference, Santa Barbara, California, USA, August 11-15, 1990, Proceedings*, volume 537 of *Lecture Notes in Computer Science*, pages 2–21. Springer, 1990.

[HNS20] Akinori Hosoyamada, María Naya-Plasencia, and Yu Sasaki. Improved attacks on sliscp permutation and tight bound of limited birthday distinguishers. *IACR Cryptol. ePrint Arch.*, 2020:1089, 2020.

[Knu94] Lars R. Knudsen. Truncated and Higher Order Differentials. In *FSE*, volume 1008 of *Lecture Notes in Computer Science*, pages 196–211. Springer, 1994.

[LLW17] Zhengbin Liu, Yongqiang Li, and Mingsheng Wang. Optimal Differential Trails in SIMON-like Ciphers. *IACR Trans. Symmetric Cryptol.*, 2017(1):358–379, 2017.

[LSSW18] Yunwen Liu, Yu Sasaki, Ling Song, and Gaoli Wang. Cryptanalysis of reduced sliscp permutation in sponge-hash and duplex-ae modes. In *SAC*, volume 11349 of *Lecture Notes in Computer Science*, pages 92–114. Springer, 2018.

[MP13] Nicky Mouha and Bart Preneel. Towards finding optimal differential characteristics for ARX: Application to Salsa20. Cryptology ePrint

Archive, Report 2013/328, 2013. https://eprint.iacr.org/2013/328.

[NIS79] NIST. FIPS-46: Data Encryption Standard (DES). 1979. http://csrc.nist.gov/publications/fips/fips46-3/fips46-3.pdf.

[NIS19] NIST. Lightweight Cryptography Competition, 2019. https://csrc.nist.gov/projects/lightweight-cryptography.

[Ran17] Adrián Ranea. An easy to use tool for Rotational-XOR Cryptanalysis of ARX Block Ciphers, 2017. https://github.com/ranea/ArxPy.

[RAS+20] Adrián Ranea, Seyyed Arash Azimi, Mahmoud Salmasizadeh, Javad Mohajeri, Mohammad Reza Aref, and Vincent Rijmen. A bit-vector differential model for the modular addition by a constant. 2020. https://eprint.iacr.org/2020/1025.

[Ste14] Stefan Kölbl. CryptoSMT: An easy to use tool for cryptanalysis of symmetric primitives, 2014. https://github.com/kste/cryptosmt.

[Wag99] David A. Wagner. The Boomerang Attack. In *FSE*, volume 1636 of *Lecture Notes in Computer Science*, pages 156–170. Springer, 1999.

[YZS+15] Gangqiang Yang, Bo Zhu, Valentin Suder, Mark D. Aagaard, and Guang Gong. The Simeck Family of Lightweight Block Ciphers. *IACR Cryptology ePrint Archive*, 2015:612, 2015.

Chapter 11

Rotational cryptanalysis on MAc algorithm Chaskey

This paper is published at ACNS 2020 and presented online by me. All implementations and writing are done by me, with the kind supervision and discussions with prof. Vincent Rijmen and prof. Tomer Ashur.

Rotational Cryptanalysis on MAC Algorithm Chaskey

Liliya Kraleva[1], Tomer Ashur[1,2], and Vincent Rijmen[1]

[1] imec-COSIC, KU Leuven, Leuven, Belgium
liliya.kraleva,tomer.ashur,vincent.rijmen@esat.kuleuven.be
[2] TU Eindhoven, Eindhoven, the Netherland

Abstract. In this paper we analyse the algorithm Chaskey - a lightweight MAC algorithm for 32-bit micro controllers - with respect to rotational cryptanalysis. We perform a related-key attack over Chaskey and find a distinguisher by using rotational probabilities. Having a message m we can forge and present a valid tag for some message under a related key with probability 2^{-57} for 8 rounds and 2^{-86} for all 12 rounds of the permutation for keys in a defined weak-key class. This attack can be extended to full key recovery with complexity 2^{120} for the full number of rounds. To our knowledge this is the first published attack targeting all 12 rounds of the algorithm. Additionally, we generalize the Markov theory with respect to a relation between two plaintexts and not their difference and apply it for rotational pairs.

Keywords: Rotational Cryptanalysis · Lightweight · ARX · Chaskey · Markov Theory

1 Introduction

When constructing a cryprographic system, one of the main building blocks is the Message Authentication Code (MAC). It is accompanying most symmetric cryptosystems used in online communication and every application where authenticity is needed. When given a message m, the MAC algorithm ensures it is authentic and that no third party has tampered with the message by providing a tag τ, computed after processing the message and a secret key k. It is usually sent together with the message as a combination (m, τ). It should be hard for the attacker to forge a valid tag for some message without knowing the key. Furthermore, the size of the tag should be large enough to prevent a guessing attack. The encryption function F used in the MAC's mode of operation can be based on various primitives like hash functions, permutations, block ciphers, pseudo-random functions, etc.

Microcontrollers are used for various applications from home devices such as ovens, refrigerators, etc., to life important applications such as providing critical functions for medical devices, vehicles and robots. They are small chips used on embedded systems and consist of a processor, memory and input/output (I/O) peripherals. Commonly used MAC algorithms for microcontrollers are

UMAC [5], CMAC [9], HMAC [21]. It is said that MACs based on a hash function or a block cipher might perform slow because of the computational cost of the underlying operations. The algorithm Chaskey [20] is said by its authors to overcome the implementation issues of a MAC on a microcontroller. It is lightweight and performs well both in software and hardware. We present it with more details in Section 4.

Rotational cryptanalysis [13] is a probabilistic technique mainly used over ARX cryptographic structures as is the permutation layer of Chaskey. It takes a rotational pair of plaintexts such that all words of one are rotations of the corresponding words of the other. After encryption, if the outputs also form a rotational pair with probability more than for a random permutation we can use that as a distinguisher. This attack has been successfully applied to ciphers like Threefish [12], Skein [11,13] and Keccak [18]. We apply it to Chaskey in a weak-key related-key scenario with size of the weak-key class 2^{120} and forge a tag over a related-key with probability 2^{-86} over all 12 rounds of the permutation. This is extended to a key-recovery attack with complexity 2^{120} for the full cipher. To our knowledge this is the first published attack targeting all 12 rounds of the algorithm. Our attack is theoretical but since Chaskey is considered for standardization we believe every input regarding its security is important.

This paper is structured as follows: In Section 2 we present some theory needed for a better understanding of the subject. In Section 3 we discuss the Markov theory and generalize it with respect to a relation between two plaintexts. The rotational attack is shown in the same section. Further in Section 4 the MAC algorithm Chaskey is presented together with a short analysis and previously published cryptanalysis techniques on it. Our results and the attack performed on Chaskey are shown in Section 5. Finally, some comments and future problems are discussed in Section 6.

2 Preliminaries and related work

2.1 Even-Mansour ciphers

The Even-Mansour cipher, first introduced in [10] is a minimalistic construction, i.e. if we eliminate any of its components the security will be compromised. This is the simplest cipher with provable security against a polynomialy bounded adversary. It consist of a key divided into two subkeys K_1 and K_2 and a random permutation F. The ciphertext is then obtained by $C = K_2 \oplus F(M \oplus K_1)$, see Fig 1. The authors prove that the construction is secure under the assumption that the permutation is (pseudo)randomly chosen and give a lower bound for the probability of success of an adversary. In [6] Daemen shows that the security claims do not hold against known plaintext and known ciphertext attacks and particularly against differential attack which reduces the security from 2^{2n} for a brute force attack over the keyspace of size $2n$ to 2^n security. He also discusses that the security proof should be based on diffusion and confusion properties and not on complexity theory. Later in [8] it is shown that the construction have the same security level even when the two keys K_1 and K_2 are equal. Attacks of

the Even-Mansour construction include differential cryptanalysis and the sliding attack [4,22]. Thanks to its simplicity this construction is widely used, including in Chaskey.

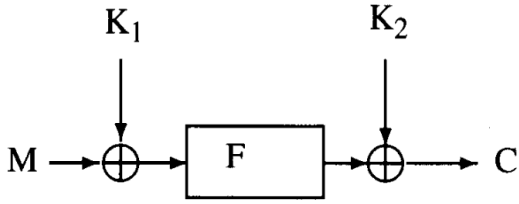

Fig. 1. Schematic model of the encryption scheme of Even-Mansour

2.2 Markov ciphers and differential cryptanalysis

In differential cryptanalysis we choose two plaintexts X and X^* with fixed difference ΔX and follow how it propagates throughout the rounds. We call a differential (α, β) an input difference α that yields an output difference β, no matter what the intermediate round differences are. The differential probability (DP) of (α, β) is the number of pairs for which $\Delta X = \alpha$ and $\Delta Y = \beta$ over the total number of pairs with input difference α. The DP is difficult to compute in practise, so what is normally done instead is to use the Expected Differential Probability (EDP), estimated by computing the product of all intermediate round probabilities.

We say that two random variables X_1 and X_2, defined on a common probability space are called stochastically equivalent if $\Pr(X_1 = X_2) = 1$. In most reasonings about the security of a cipher against differential cryptanalysis, we use the hypothesis of stochastic equivalence, defined as follows:

Definition 1 (Hypothesis of stochastic equivalence [15]). *For an* $(r-1)$*-round differential* (α, β)*,*

$$\Pr(\Delta Y(r-1) = \beta | \Delta X = \alpha) \approx \Pr(\Delta Y(r-1) = \beta | \Delta X = \alpha, \\ Z^{(1)} = \omega_1, Z^{(2)} = \omega_2, \ldots, Z = \omega_{(r-1)})$$

for almost all subkeys values $(\omega_1, \ldots, \omega_{r-1})$*, where* $Z^{(i)}$ *denote the* $i-th$ *subkey.*

It means that the probability of a differential does not depend on the choice of subkeys. We compute or bound the Expected Differential probability (EDP) of a differential and assume that $DP[k](\alpha, \beta) \approx EDP(\alpha, \beta)$ holds for almost all keys.

Definition 2 (Markov cipher [15]). *An iterated cipher with round function $Y = f(X, Z)$ is a Markov cipher if there is a group operation $\otimes$ for defining differences such that, for all choices of α and β, $(\alpha, \beta \neq e)$,*

$$\Pr(\Delta Y = \beta | \Delta X = \alpha, X = \gamma)$$

is independent of γ, when the subkey is uniformly random.

If we have a Markov cipher, then the probability of a differential does not depend on the choice of input text, $EDP(\alpha_r, \alpha_{r+1}) = EDP(\alpha_r, \alpha_{r+1}|X(r))$. A Markov chain is a sequence $v_0, v_1, \ldots$ of random variables satisfying the rule of conditional independence, or with other words variables for which the output of the rth iteration does not depend on the previous $r-1$ iterations. Mathematically formulated, a sequence of discrete random variables $(v_0, \ldots, v_r)$ is a Markov chain if, for $0 < i < r$ (where $r = \infty$ is allowed)

$$P(v_{i+1} = \beta_{i+1} | v_i = \beta_i, v_{i-1} = \beta_{i-1}, \ldots, v_0 = \beta_0) = P(v_{i+1} = \beta_{i+1} | v_i = \beta_i).$$

Finally, the following theorem is formulated:

Theorem 1 ([15]). *If an r-round iterated cipher is a Markov cipher and the r round keys are independent and uniformly random, then the sequence of differences $\Delta X = \Delta Y(0), \Delta Y(1), \ldots, \Delta Y(r)$ is a homogeneous Markov chain.*

If we have a Markov cipher and the round keys are independent and uniformly random we can use the Chapman-Kolmogorov equation for a Markov chain to compute the $EDP(\alpha, \beta)$ by multiplying $EDP(\alpha_r, \alpha_{r+1})$ over all the rounds, which is easy to compute in comparison to the real DP. In general, for alternating ARX ciphers the Markov theory holds with respect to differential cryptanalysis.

2.3 Attack settings

There are different scenarios in which we can attack a MAC. The single-user setting suggests that Alice and Bob share the same key so Bob can authenticate that the messages he receives from Alice are not changed in any way. In the existential forgery problem (EFP) the adversary has access to the encryption and decryption oracles. If the adversary can present a new message with a valid tag then this is a forgery and the adversary wins the game. Another scenario is the multi-user setting in which we have multiple users typically with their own secret keys. The adversary then wins if it can present a triplet (i, m', τ') for some user i and new message m'. If the number of users is large enough the adversary can find a collision in the keys due to the birthday paradox. Having at least two users using the same key or related in some way keys can then enhance any further attack in matter of data, memory and time. Some environments allow to tamper with the key and change it in a certain way, like adding a constant for example. We can then observe the ciphertext under the related keys and draw conclusions over the real key. This is called a related-key attack [14] and is quite a powerful setting. As shown in [2, 3] even the AES is theoretically vulnerable

under related-key attacks, but not if you can only add a constant. Stronger attacks are such that reveal some bits of the key and in the multi-user scenario one or more of the individual keys are exposed. They are called key-recovery attacks.

3 Rotational Cryptanalysis and generalized Markov ciphers

In this section we discuss the Markov theory with respect to a relation between chosen plaintexts and not to their difference. We extend the definition for more general cases and more specifically, we concentrate on plaintexts forming a rotational pair. Further in Section 3.2 we recall the idea of rotational cryptanalysis and how to compute the rotational probability for ARX ciphers.

3.1 Markov theory and rotational cryptanalysis

In [13] the authors mention the term rotational difference and argue that modular additions do not form a Markov chain with respect to the rotational property. In fact, we cannot consider a rotational difference in the sense it is defined in [15]:

Definition 3. *[15] The* difference ΔX *between two plaintexts (or two ciphertexts)* X *and* X^* *is*

$$\Delta X = X \otimes X^{*-1},$$

where $\otimes$ *denotes a specified group operation on the set of plaintexts (= set of ciphertexts) and* X^{*-1} *denotes the inverse of the element* X^* *in the group.*

According to this formulation the rotation has to be a group operation with the integers, which it is not. It is a group action $\mathbb{F}_2^n \times \mathbb{N} \to \mathbb{F}_2^n$, whereas a group operation is defined in $\mathbb{F}_2^n \times \mathbb{F}_2^n \to \mathbb{F}_2^n$.

In order to use the Markov theory we need to slightly extend it. We generalize the concept of "two plaintexts X and X^* have a certain difference ΔX" to "X and X^* have a certain relation". The definition of a Markov cipher can easily be generalized to accommodate this. Let us have two related plaintexts X and X^*, such that for a relation $R \subseteq \mathbb{F}_2^n \times \mathbb{F}_2^n$ we have $(X, X^*) \in R$ if X^* has a relation R with X.

Definition 4 (Generalized Markov cipher). *An iterated cipher with round function* $Y = f(X, Z)$ *is a* generalized Markov cipher *if there are two relations, different from the identity,* R^α *and* R^β*, such that for all choices of* α *and* β*,*

$$\Pr((Y, Y^*) \in R^\beta | (X, X^*) \in R^\alpha, X = \gamma) \tag{1}$$

is independent of the choice of γ *when the subkeys* Z *are uniformly random.*

Note that for differences we have for $\alpha \neq 0$, $R^{\alpha} = \{(X, X^*)|X^* = X \otimes \alpha^{-1}, X \in \mathbb{F}_2^n\}$. For our purposes X^* is a rotation of X over l positions to the left, $R^l = \{(X, X^*)|X^* = X^{\lll l}, X \in \mathbb{F}_2^n\}$ and we want the same relation to hold between the plaintexts and the ciphertexts. Then the condition for Markov cipher is that for any $l \neq 0$ the probability $Pr(Y^* = Y^{\lll l}|X^* = X^{\lll l}, X = \gamma)$ is independent of γ, when the subkeys are uniformly random.

Without generalization, the hypothesis of stochastic equivalence does not hold for a rotational pair: XOR-ing with a fixed key will maintain the rotational property only if the key is rotation-symmetric. The hypothesis can be generalized to some related-key scenario where the relation between the keys is that the second key is a rotation of the first key. Since for the rotational property to hold in the last state we need it to hold in every intermediate state as well, in general we do not have a Markov chain. Once the property is broken, it cannot come back by chance. Therefore, EDP cannot be calculated as a product of the round probabilities as in differential cryptanalysis.

3.2 Rotational attack

Let us consider a pair of plaintexts $(m, m \lll l)$, where $m \lll l$ is a rotation of m to the left with l positions. We call this a rotational pair. When after some operations the outputs also form a rotational pair we say that the rotational property holds. It is preserved by all bit-wise operations like XOR or another rotation, but not always by modular addition. The attack relies on the fact that the probability after modular addition can be computed (proven in [7]) and is

$$\Pr((x + y) \lll l = x \lll l + y \lll l) = \frac{1}{4}(1 + 2^{l-n} + 2^{-l} + 2^{-n}) \qquad (2)$$

for n-bit long words, while

$$\Pr((x \lll l \oplus y \lll l = (x \oplus y) \lll l) = 1.$$

$$\Pr((x \lll l_1) \lll l_2 = (x \lll l_2) \lll l_1) = 1$$

That makes it applicable to ARX structures, in which the only operations are modular addition, rotation, and XOR. More precisely, we start the attack from a rotational pair of two states $(X, \overleftarrow{X})$ of size n and divided into s words typically of 32 or 64 bits. With $\overleftarrow{X}$ we denote the word-wise rotation of X : $X = (x_1, x_2, \ldots, x_s), \overleftarrow{X} = (x_1 \lll l, x_2 \lll l, \ldots, x_s \lll l)$, where $x_i, i = 1, \ldots s$ are the state words.

If the corresponding output states also form a rotational pair with probability higher than for a random permutation, we can use this property as a distinguisher.

When the attack was first formalized as a rotational cryptanalysis in [12] the authors claimed that the rotational probability of an ARX cipher depends only on the number of modular additions in the algorithm and can be easily computed as shown in the following theorem:

Theorem 2. *[12] Let q be the number of additions in an ARX primitive. Then the rotational probability of the primitive is p_+^q, where p_+ is the rotational probability of modular addition as calculated in (2).*

This is only valid under the assumption of stochastic equivalence and Markov chain, both in fact do not hold with respect to the rotational property.

In [13] the authors introduce chained modular additions, namely additions for which the output of one is the input to the other. The output of modular addition is biased when the input is a rotational pair. Namely, if $(x+y) \lll l = x \lll l + y \lll l$ and $r > 0$, then the value $z = x + y$ is biased. More precisely, for $l = 1$, the most significant bit of z is biased towards 1. The second modular addition has smaller probability and therefore Theorem 2 fails to give the correct probability. Due to this bias the variables are not random and independent, so we can say they do **not** form a Markov chain. Therefore, the probability does not depend only on the number of additions but on their positions as well. The authors also introduce the following formula, that very precisely calculates the rotational probability of $k-1$ consecutive modular additions:

Lemma 1. *[13, Lemma 2] Let $a_1, \ldots, a_k$ be n-bit words chosen at random and let l be a positive integer such that $0 < l < n$. Then*

$$\begin{aligned} &Pr([(a_1 + a_2) \lll l = a_1 \lll l + a_2 \lll l] \wedge \\ &\wedge [(a_1 + a_2 + a_3) \lll l = a_1 \lll l + a_2 \lll l + a_3 \lll l] \wedge \\ &\ldots \\ &[(a_1 + \ldots a_k) \lll l = a_1 \lll l + \ldots a_k \lll l]) = \\ &= \frac{1}{2^{nk}} \binom{k + 2^l - 1}{2^l - 1} \binom{k + 2^{n-l} - 1}{2^{n-l} - 1}. \end{aligned}$$

The more chained additions we have, the lower the probability. In Table 1 we can see a comparison between the rotational probabilities calculated with the independence assumption and with the formula from Lemma 1 for rotational amount $l = 1$. We can see that for a larger number of chained additions the difference is quite big and suggests that chained additions are a better design choice with respect to rotational cryptanalysis.

Table 1. log_2 values for the rotational probabilities calculated with the formula of Theorem 2 and Lemma 1 for rotational amount $r = 1$.

# of additions	1	2	3	4	5	10	20	30
Theorem 2	−1.4	−2.8	−4.2	−5.7	−7.1	−14.1	−28.3	−42.4
Lemma 1	−1.4	−3.6	−6.3	−9.3	−12.7	−32.7	−82.0	−138.7

4 The MAC algorithm Chaskey

In this chapter we introduce the MAC algorithm Chaskey and the previously performed attacks on it.

4.1 Chaskey

Chaskey [20] is a lightweight Message Authentication Code (MAC) algorithm that is dedicated to 32-bit microcontrollers. It is claimed to have better performance and efficiency than previously used algorithms and it is provably secure based on the Even-Mansour structure.

The algorithm is as follows: a 128-bit key K is used with 128-bit blocks of messages m_i in a permutation π, designed only with XOR, rotation and modular addition operations. These simple operations are very efficient in software and in hardware.

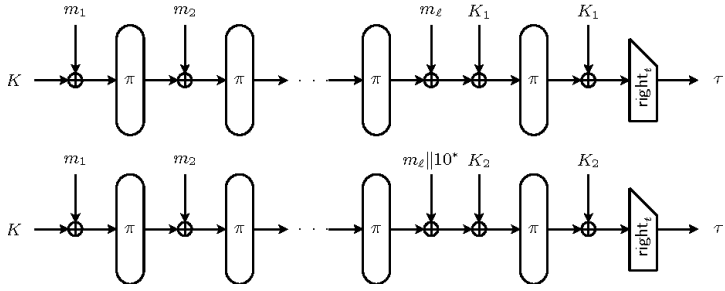

Fig. 2. The Chaskey mode of operation

The mode of operation can be seen in Fig. 2. The text is broken into 128-bit blocks m_i which are consecutively XORed and passed through a permutation. There is a key addition before the first and after the last block. If the last block is less than 128 bits, a 1 is appended and as many 0 bits as necessary (the second mode in Fig. 2). Finally, the last t bits of the output are used as a tag. In the paper in which Chaskey was first proposed [20] the authors suggested that 8 or 16 rounds should be used on the permutation π, although 8 provide enough security. Later in [19] the rounds were set to 12. One round of the permutation is shown in Fig. 3. The algorithm can also be considered as an Even-Mansour cipher with keys K and K_1, respectively K_2 when the last message block is less than 128 bits. Here K_1, K_2 are generated from K by simple polynomial multiplication by x, respectively x^2 over the finite field $F_{2^{128}}$ with generating polynomial $g(x) = x^{128} + x^7 + x^2 + x + 1$. For $K_1 = xK$ this means we shift K with one position to the left if the first bit(the leftmost bit) is equal to zero or shift and then XORed with $0^{120}10000111$ if the first bit is one. If the bit is 0 then K_1 can be considered as a state-rotation of K. For $K_2 = x^2K$ the same operation is valid and applied twice.

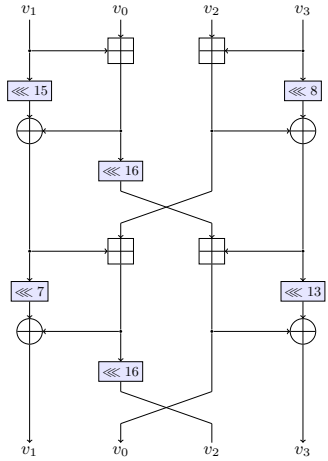

Fig. 3. A round of the Chaskey permutation

4.2 Markov theory and Chaskey

Chaskey is Even-Mansour cipher, hence there are no round keys. The generalized hypothesis of stochastic equivalence holds. This means if we are in a related-key scenario, the probability that the rotational property holds is independent of the choice of keys as long as the second key is a rotation of the first one. This can be proven easily for any Even-Mansour construction. Even with the generalized definition (Def. 4) Chaskey is not a Markov cipher. Since there are no round keys, $EDP(a_r, a_{r+1}|X_r)$ is either 0 or 1 (there is no randomness when the input is fixed). $EDP(a_r, a_{r+1})$ is the average over all inputs, which will often be a value between 0 and 1. One can still hope to estimate $EDP(\alpha)$ with respect to rotational operation as the product over all rounds. Khovratovich et al. [13] show that this leads to wrong results and propose improved formula - Lemma 1 (Lemma 2 in [13]). Our experiments confirm that the formula gives trustable results.

4.3 Previous attacks on Chaskey

In [17] a collision-based attack both in the single and multi-user scenarios is executed. That is, we define two functions $f_s(m) = K_s \oplus \pi(m \oplus (K_s \oplus K))$ and $F_{f_s}(M) = f_s(M) \oplus f_s(M \oplus \delta) \oplus M$ and search for collisions between the chains constructed from this two functions. It can be seen from Fig. 4 that $f_s(M)$ describes the Chaskey mode for one block of text. As a result, the attack has complexity 2^{64} in the single-user scenario and to recover all keys in the multi-user scenario needs 2^{43} users and 2^{43} queries per user.

In [16] a differential-linear attack is performed over Chaskey, improved with the partitioning technique proposed in [1]. Their best result is over 6 and 7

Table 2. Review of the existing attacks over Chaskey

Rounds	Data	Time	Attack	Reference
6	2^{25}	2^{29}	linear-differential attack with partitioning, gains 6 bits of the key	[16]
7	2^{48}	2^{67}	linear-differential attack with partitioning, gains 6 bits of the key	[16]
8	2^{64}		collision attack in single user mode, full key recovery	[17]
8	2^{43} per user for 2^{43} users		collision attack in multi-user mode, full key recovery	[17]
6	2^{42}		weak-key related-key rotational distinguishing attack	here
12	2^{86}		weak-key related-key rotational attack, forge a valid tag	here
12	2^{120}		weak-key related-key rotational attack, full key recovery	here

rounds with data complexity 2^{25} and 2^{48} respectively and time complexity 2^{29} and 2^{67} respectively. The attack builds differential-linear distinguisher which is extended to key-bits recovery in the last round phase.

A comparison between those attacks and our contribution can be seen in Table 2.

5 Application to Chaskey

In this section we will show how we apply the rotational property in several attack scenarios. We first show how to calculate the rotational probabilities and then how to use them as a distinguisher, to forge a message or for key-recovery.

5.1 Calculating the rotational probability

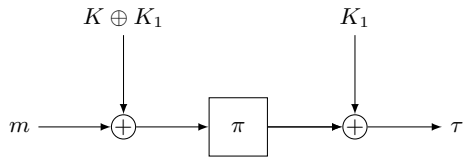

Fig. 4. Chaskey mode of operation for messages with single block of 128 bits

We consider the case where we want to tag a message m that has only one block of 128 bits. Then the tag would be $\tau = \pi(K \oplus K_1 \oplus m) \oplus K_1$, as shown in Fig. 4.

To apply rotational cryptanalysis to Chaskey we first need to calculate the rotational probability of the permutation π, i.e. the probability for a rotational pair of input texts $(m, \overleftarrow{m})$ to produce output pair $(\pi(m), \overleftarrow{\pi(m)})$. It depends only on the number of modular additions and their positions. Note that in one round of π (see Fig. 3) we have 4 modular additions - two single additions and one chain of two. Further note that when we continue the permutation to a second round, the addition of $v_0 + v_3$ makes a chain with the addition $v_0 + v_1$ of the second round which bring us to 2 singles and 3 chains of two modular additions for 2 rounds, and so on for any further round. These chains are depicted in bold in Fig. 5. We then calculate the probability using Lemma 1 [13]. More precisely, we take the parameter k to be the number of chained additions that we have plus 1 and set the rotation r to 1, because then the probability is maximized. The size of the words is $n = 32$. Let a_1, a_2 and a_3 be the words that we are adding. Then the probability of a chain with two additions $(a_1 + a_2) + a_3$ is

$$\begin{aligned}&\Pr([(a_1 + a_2) \lll 1 = a_1 \lll 1 + a_2 \lll 1] \wedge \\ &\wedge [(a_1 + a_2 + a_3) \lll 1 = a_1 \lll 1 + a_2 \lll 1 + a_3 \lll 1]) = \\ &= \frac{1}{2^{32.3}} \binom{3+2-1}{2-1} \binom{3+2^{32-1}-1}{2^{32-1}} = 2^{-3.6}.\end{aligned}$$

Table 3. Table with the theoretical and experimentally calculated rotational probabilities for any number of rounds of the permutation π

rounds	Modular additions		Theoretical probability	Experimental probability
	singles	chains of two		
1	2	1	-6.436	-6.421
2	2	3	-13.636	-13.639
3	2	5	-20.836	-20.844
4	2	7	-28.036	-28.142
5	2	9	-35.236	-36
6	2	11	-42.436	
7	2	13	-49.636	
8	2	15	-56.836	
9	2	17	-64.036	
10	2	19	-71.236	
11	2	21	-78.436	
12	2	23	-85.636	

Further, for 8 rounds we have 15 chains of two additions and 2 single additions, which corresponds to a probability $p = (2^{-3.6})^{15}.(2^{-1.4})^2 = 2^{-56.836}$. Table 3 presents how many single and chained additions we have for any number of rounds up to 12 and what is the evaluated rotational probability calculated with Lemma 1 and the experimental probability we get after running simulations. We performed our experiments on a computer with an Intel(R) Core(M)

i5-4590 CPU running at 3.30 GHz. We did not perform experiments beyond 5 rounds due to the time complexity. Our experimental results are very close to the expected ones and based on that observation we anticipate that the probability calculated with this formula is correct and further take it as verified and refer to it when considering larger number of rounds.

We show an example how certain words change through the operations of the Chaskey permutation in Fig. 5. The rotational property after the modular additions can be observed for message m and its rotation $\overleftarrow{m}$. We can see that for words v_2 = `0xA1008E9C` = 01000010000000010001110100111001 and v_3 = `0x45EAA81C` = 10001011110101010101000000111000 the rotational property holds and the pair $(v2 + v3, \overleftarrow{v2} + \overleftarrow{v3})$ is rotational.

What we have to consider next is the rotational probability of the whole Chaskey function $\Pi = \pi(K \oplus K_1 \oplus m) \oplus K_1$, that is with the key addition before and after the permutation π. In fact $\pi(K \oplus K_1 \oplus \overleftarrow{m}) \oplus K_1$ cannot be a rotation of $\pi(K \oplus K_1 \oplus m) \oplus K_1$, since $a \oplus b \neq a \oplus \overleftarrow{b}$. Therefore we need to consider also a rotated key. The pair $\pi(K \oplus K_1 \oplus m) \oplus K_1$ and $\pi(\overleftarrow{K} \oplus K_1' \oplus \overleftarrow{m}) \oplus K_1'$, where K_1' is the key generated from $\overleftarrow{K}$, i.e. $K_1' = x\overleftarrow{K} \bmod g(x)$, is rotational for a large set of keys, but not all keys. To ensure K_1' is word rotation of K_1, i.e. $K_1' = \overleftarrow{K_1}$, we need the first 2 bits of every word to be equal to zero. This becomes clear with the following example: denote with a, b, f, g, k, l, p and q the first 2 bits of the 4 words of K and with $*$ any bundle of 30 bits that we are not interested in. Then the word and state rotations of K and K_1 are as follows: .

$$K = ab* \quad fg* \quad kl* \quad pq*; \quad K_1 = b*f \quad g*k \quad l*p \quad q*a$$

$$\overleftarrow{K} = b*a \quad g*f \quad l*k \quad q*p; \quad K_1' = *ag \quad *fl \quad *kq \quad *pb$$

If $a = 1$ (resp. $b = 1$) then K_1 (resp. K_1') will not be a rotation of K (resp. $\overleftarrow{K}$) but a rotation and XOR with 135 in decimal. Therefore we set $a = b = 0$. Furthermore, to have $K_1' = \overleftarrow{K_1} = *fb \quad *kg \quad *pl \quad *aq$ we need to set $a = b = f = g = k = l = p = q$ which means we set the first 2 bits of every word of K to be zero.

The keys satisfying this property we call a weak-key class of keys and there are 2^{120} weak-keys for which we can apply the attack. For the rest of the keys the rotational property will definitely not hold. This means if a random key K is chosen, the probability that we will have a rotational pair after Π is $2^{-8}p_1$,where p_1 is the rotational probability of π.

5.2 Attack scenarios

Distinguisher. Using the rotational property we can distinguish whether a key is in the weak-key class of keys that we defined earlier, namely one of the 2^{120} keys with the first two bits of every word being zero.

In this setting we can use the authentication oracles for the related keys K and $\overleftarrow{K}$, denoted respectively as O_K and $O_{\overleftarrow{K}}$. The properties of the algorithm

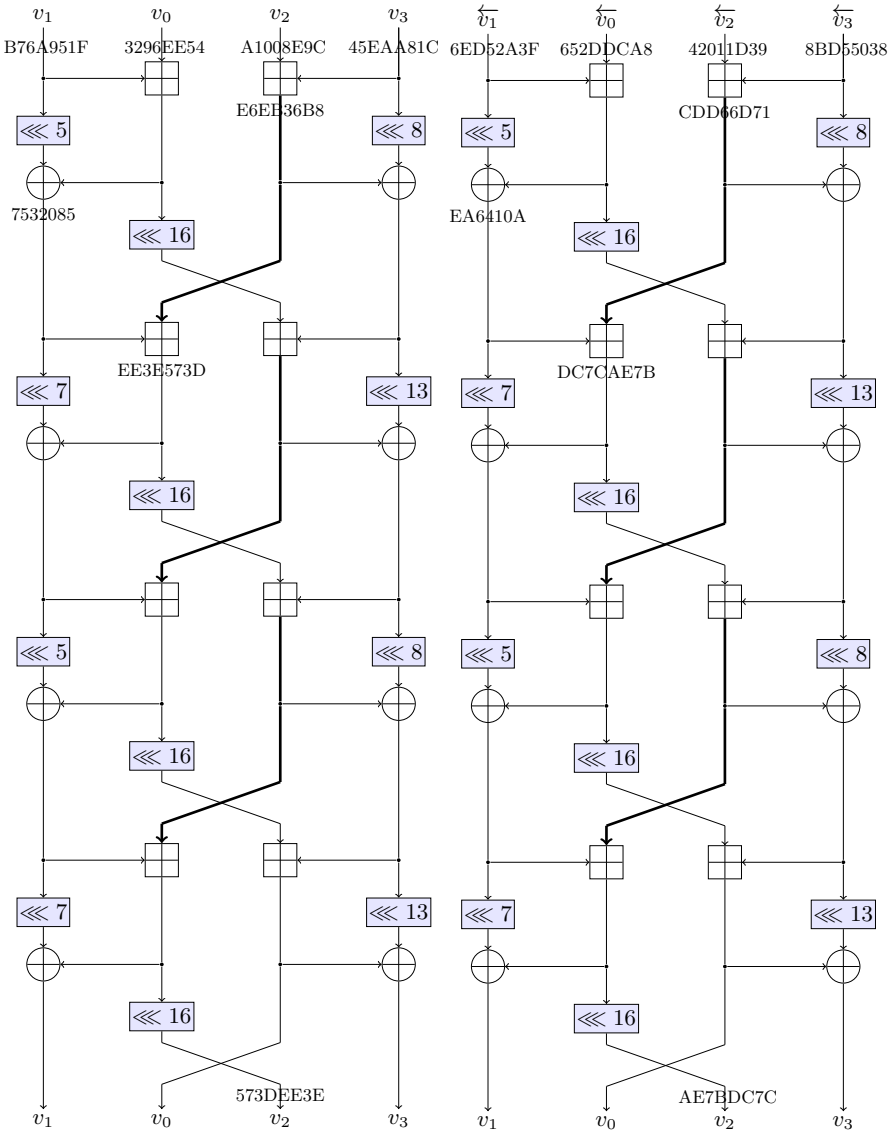

Fig. 5. 2 rounds of Chaskey's permutation for chosen input(left) and its word-wise rotation (right). The chained modular additions are connected with ticker line. The words are represented in hexadecimal.

Chaskey allow us to tag 2^{48} messages under the same key, therefore we will take a set of that many messages $D = \{m \in F_2^{128}\}, |D| = 2^{48}$. We send all messages $m_i \in D, i = 1 \dots 2^{48}$ to O_K and their word-wise rotations $\overleftarrow{m_i}$ to $O_{\overleftarrow{K}}$ and the oracles give back the corresponding tags τ_i and τ_i'. If the key K is one of the weak-key class keys, then the expected number of messages for which $\overleftarrow{\tau} = \tau'$ will be $2^{48}p$, where p is the rotational probability according to Table 3. For example, over 6 rounds we will have $2^{48} \cdot 2^{-42} = 2^8 = 64$ messages for which that happens. In the single user mode for the full 12 rounds the probability of success is $p = 2^{48} \cdot 2^{-86} = 2^{-38}$ with data complexity 2^{49} encryptions, time complexity 2^{48} look ups in a table and memory of storing a table with $128 \cdot 2^{48}$ bits which is 2^{47} bytes. In order to have success probability one we repeat the experiment 2^{38} times which gives us data complexity of 2^{86}.

Tag forgery. We can find users with related keys and exploit this fact to forge a tag. One way of doing this is by querying each of the n users with the same message m and then store the corresponding tags τ_i. By doing the same thing but this time with message $\overleftarrow{m}$, we can look for a collision between the stored tags and the new tags τ_j'. When $\tau_i = \tau_j'$ for some i, j, then users i and j have related keys and we can perform a forgery attack that goes as follows:

1. We collect data of pairs of a message and its corresponding tag (m, τ) from user u_i.
2. We rotate them word-wise and send $(\overleftarrow{m}, \overleftarrow{\tau})$ for verification.

If we are attacking 8 rounds and we have 2^{57} pairs we expect at least 1 pair to be accepted as authentic which means we will posses a message and a valid tag. That is considered a forgery and a success to our attack. As a result from user with key K we can generate a valid tag for some message $\overleftarrow{m}$ under the related key $\overleftarrow{K}$ with complexity of the attack 2^{57} under the assumption that we are in the weak-key class. For 12 rounds of the permutation the probability of success is 2^{-86}.

Key - recovery. After distinguishing that we are in the weak-key class we know 8 bits of the key. The rest of the bits can be recovered by guessing which gives us a key - recovery with complexity 2^{120}.

6 Conclusions and future work

In this work we showed how by using the property of rotational probabilities we can forge a valid tag using the MAC algorithm Chaskey. Our results are not compromising the security of the algorithm in practice and do not violate the security claims of the authors, however they show a vulnerability in the underlying permutation. Our best result is a distinguishing attack over the full number of rounds of the algorithm with complexity 2^{86}.

The Chaskey algorithm suggests that only the last t bits of the output can be used as a tag. Our attack is over all 128 bits. However, for shorter tags the

results could be enhanced by only following the rotational property to 2 or 3 words of the output. This is to be further analysed.

Acknowledgements. We thank the anonymous reviewers for their comments. This research is supported by a Ph.D. Fellowship from the Research Foundation - Flanders (FWO), the Research Council KU Leuven - grant C16/18/004 and FWO-BAS grant VS.077.18. Tomer Ashur is an FWO post-doctoral fellow under grant number 12ZH420N.

References

1. Biham, E., Carmeli, Y.: An improvement of linear cryptanalysis with addition operations with applications to FEAL-8X. In: Selected Areas in Cryptography. Lecture Notes in Computer Science, vol. 8781, pp. 59–76. Springer (2014)
2. Biryukov, A., Khovratovich, D.: Related-key cryptanalysis of the full AES-192 and AES-256. In: ASIACRYPT. Lecture Notes in Computer Science, vol. 5912, pp. 1–18. Springer (2009)
3. Biryukov, A., Khovratovich, D., Nikolic, I.: Distinguisher and related-key attack on the full AES-256. In: CRYPTO. Lecture Notes in Computer Science, vol. 5677, pp. 231–249. Springer (2009)
4. Biryukov, A., Wagner, D.A.: Advanced slide attacks. In: EUROCRYPT. Lecture Notes in Computer Science, vol. 1807, pp. 589–606. Springer (2000)
5. Black, J., Halevi, S., Krawczyk, H., Krovetz, T., Rogaway, P.: UMAC: fast and secure message authentication. In: CRYPTO. Lecture Notes in Computer Science, vol. 1666, pp. 216–233. Springer (1999)
6. Daemen, J.: Limitations of the Even-Mansour Construction. In: ASIACRYPT. Lecture Notes in Computer Science, vol. 739, pp. 495–498. Springer (1991)
7. Daum, M.: Cryptanalysis of Hash functions of the MD4-family. Ph.D. thesis, Ruhr University Bochum (2005), http://www-brs.ub.ruhr-uni-bochum.de/netahtml/HSS/Diss/DaumMagnus/
8. Dunkelman, O., Keller, N., Shamir, A.: Minimalism in Cryptography: The Even-Mansour Scheme Revisited. In: EUROCRYPT. Lecture Notes in Computer Science, vol. 7237, pp. 336–354. Springer (2012)
9. Dworkin, M.: Recommendation for block cipher modes of operation: The CMAC mode for authentication. NIST special publication 800-38b, National Institute of Standards and Technology (NIST) (May 2005) (2005)
10. Even, S., Mansour, Y.: A Construction of a Cipher From a Single Pseudorandom Permutation. In: ASIACRYPT. Lecture Notes in Computer Science, vol. 739, pp. 210–224. Springer (1991)
11. Ferguson, N., Lucks, S., Schneier, B., Whiting, D., Bellare, M., Kohno, T., Callas, J., Walker, J.: The skein hash function family. Submission to SHA-3 Nist Competition (2008) (2008)
12. Khovratovich, D., Nikolic, I.: Rotational Cryptanalysis of ARX. In: FSE. Lecture Notes in Computer Science, vol. 6147, pp. 333–346. Springer (2010)
13. Khovratovich, D., Nikolic, I., Pieprzyk, J., Sokolowski, P., Steinfeld, R.: Rotational Cryptanalysis of ARX Revisited. In: FSE. Lecture Notes in Computer Science, vol. 9054, pp. 519–536. Springer (2015)

14. Knudsen, L.R., Rijmen, V.: Known-Key Distinguishers for Some Block Ciphers. In: ASIACRYPT. Lecture Notes in Computer Science, vol. 4833, pp. 315–324. Springer (2007)
15. Lai, X., Massey, J.L., Murphy, S.: Markov ciphers and differential cryptanalysis. In: EUROCRYPT. Lecture Notes in Computer Science, vol. 547, pp. 17–38. Springer (1991)
16. Leurent, G.: Improved differential-linear cryptanalysis of 7-round Chaskey with partitioning. In: EUROCRYPT (1). Lecture Notes in Computer Science, vol. 9665, pp. 344–371. Springer (2016)
17. Mavromati, C.: Key-Recovery Attacks Against the MAC Algorithm Chaskey. In: SAC. Lecture Notes in Computer Science, vol. 9566, pp. 205–216. Springer (2015)
18. Morawiecki, P., Pieprzyk, J., Srebrny, M.: Rotational Cryptanalysis of Round-Reduced Keccak. In: FSE. Lecture Notes in Computer Science, vol. 8424, pp. 241–262. Springer (2013)
19. Mouha, N.: Chaskey: a MAC Algorithm for Microcontrollers - Status Update and Proposal of Chaskey-12 -. IACR Cryptology ePrint Archive **2015**, 1182 (2015), http://eprint.iacr.org/2015/1182
20. Mouha, N., Mennink, B., Herrewege, A.V., Watanabe, D., Preneel, B., Verbauwhede, I.: Chaskey: An Efficient MAC Algorithm for 32-bit Microcontrollers. In: Selected Areas in Cryptography. Lecture Notes in Computer Science, vol. 8781, pp. 306–323. Springer (2014)
21. Turner, J.: The keyed-hash message authentication code (HMAC). FIPS PUB 198-1, National Institute of Standards and Technology (NIST) (July 2008) (2008), http://csrc.nist.gov/publications/fips/fips198-1/FIPS-198-1_final.pdf
22. Yang, G., Zhang, P., Ding, J., Hu, H.: Advanced slide attacks on the Even-Mansour scheme. In: DSC. pp. 615–621. IEEE (2018)

Addendum

In this paper we present a key-recovery attack in a weak-key model. Note, that an attack that guesses the unknown bits of the key after fixing some bits in order to be in a weak class does not make sense as it is no different than brute force. Instead, we first distinguish whether we are in the weak key class and only after we have established we are indeed in that class, can we apply the key-recovery in the form of guessing the remaining bits.

The designers of Chaskey provide a security proof based on the Even-Mansour construction [58]. However, as pointed out by Daemen in [45] an Even-Mansour cipher with n-bit block size can be broken with $2^{n/2}$ chosen plaintexts. What is considered as provably-secure due to the Even-Mansour construction after this paper then becomes the bound of $2^{n/2}$ data complexity. Additionally, the Even-Mansour construction assumes an ideal permutation, and for Chaskey we show that this is not the case, as the permutation is vulnerable to rotational pairs. Hence the security proof for Chaskey is not in contradiction with the comment by Daemen [45], nor with our results presented in this chapter.

Chapter 12

Correlation distribution analysis of a two-round key-alternating block cipher

This paper is published at Tatra Mauntains Mathematical Publications 73(1). Additionally, it was presented at CECC 2018 and at Colloquium Coding Theory and Cryptography 2018, Gent. Both presentations were given by me.

Most of the results in this paper follow from my master thesis, obtained at Sofia University and partly supervised by prof. Vincent Rijmen during my research stay at COSIC. All results concerning the case of $f = x^3$ are developed and implemented by me, under the supervision of prof. Vincent Rijmen and partly by prof. Nikolai Manev. I have little contribution to the generalized cases of $f = x^{(2^t+1)2^s}$ for s, t-positive integers, which are mainly developed by prof. Nikolai Manev. I have contributed to the writing of the paper.

Correlation distribution analysis of a two-round key-alternating block cipher

Liliya Kraleva[1], Vincent Rijmen[1], and Nikolai L. Manev[2]

[1] imec-COSIC, KU Leuven, Leuven, Belgium
liliya.kraleva,vincent.rijmen@esat.kuleuven.be
[2] Bulgarian Academy of Sciences, Sofia, Bulgaria
nlmanev@math.bas.bg

Abstract. In this paper we study two-round key-alternating block ciphers with round function $f(x) = x^{(2^t+1)2^s}$, where t, s are positive integers. An algorithm to compute the distribution weight with respect to input and output masks is described. In the case $t = 1$ the correlation distributions in dependence on input and output masks are completely determined for arbitrary pairs of masks. We investigate with more details the case $f(x) = x^3$ and fully derive and classify the distributions, proving that there are only 5 possible values for the correlation for any pair of masks.

Keywords: correlation distribution, linear cryptanalysis, key-alternating ciphers, cube function

1 Introduction

Linear cryptanalysis is one of the most powerful attacks on symmetric-key block ciphers. It investigates the correlation between chosen bits of the input and the output in order to make conclusions about some bits of the key. Nowadays ciphers are designed to be resistant against linear cryptanalysis by analyzing some statistical properties of the cipher.

The study presented herein is inspired by a paper by Abdelraheem, Agren, Beelen and Leander [1]. In the paper the authors give an example of two-round key-alternating block cipher (see Figure 1) with correlation distribution for masks $(1, 1)$ that is behaving differently from what is expected. It takes only five different values whereas previously it was assumed that the distribution would be normal [7]. The cryptanalysis of two-round key-alternating block ciphers is interesting because it is the basic step of the cryptanalysis of any multi-round block cipher.

In this paper we consider the same cipher but with a round function of more general form and we determine the correlation distribution in dependence of the key for arbitrary input and output masks (σ, ω), where $\sigma, \omega \in \mathbb{F}_2^n$, n odd.

Our methods are based on finite-field theory, see e.g. [12, 14, 11, 3] and some general results on correlation analysis and linear cryptanalysis [5, 8, 13].

In the next section we give the necessary notions and notations. In Section 3 we prove several general facts about correlation distribution and introduce the notion distribution weight. In Section 4 we give some theoretical results and describe how to compute the distribution weight for round function of the form $f(x) = x^{(2^t+1)2^s}$. How to compute the correlation distribution in the case $t = 1$ is described in Section 5. Computational results are presented in Section 6. For the sake of reader's convenience the proofs of the theorems in Section 2 are given in Appendix A.

2 Preliminaries

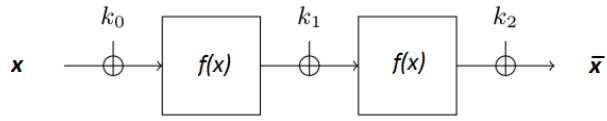

Fig. 1. Two round key-alternating cipher.

Let $f : \mathbb{F}_2^n \to \mathbb{F}_2^n$ be a bijection and $\langle \mathbf{u}, \mathbf{v} \rangle \stackrel{def}{=} \mathbf{u}\mathbf{v}^\tau = u_1v_1 + \cdots + u_nv_n$ be the inner product of $\mathbf{u}$ and $\mathbf{v}$. Note that $\langle \mathbf{u}, \mathbf{v} \rangle \in \mathbb{F}_2$, thus, takes values 0 or 1.

Definition 1. The **correlation** of the **linear approximation** $(\mathbf{u}, \mathbf{v})$ of f is referred to be the difference

$$c_f(\mathbf{u}, \mathbf{v}) \stackrel{def}{=} \frac{1}{2^{n-1}} |\{\mathbf{x} \in \mathbb{F}_2^n \mid \langle \mathbf{u}, \mathbf{x} \rangle + \langle \mathbf{v}, f(\mathbf{x}) \rangle = 0\}| - 1.$$

The vectors $\mathbf{u}$ and $\mathbf{v}$ are commonly called *input and output masks* of the approximation.

Let $f, g : \mathbb{F}_2^n \to \mathbb{F}_2^n$ be bijections and let $f_k : \mathbb{F}_2^n \to \mathbb{F}_2^n$ be defined by $f_k(\mathbf{x}) = f(\mathbf{x}) + \mathbf{k}$, $\mathbf{k} \in \mathbb{F}_2^n$. Let us denote by F_k the composition $F_k(\mathbf{x}) = g \circ f_k(\mathbf{x}) = g(f(\mathbf{x}) + \mathbf{k})$.

Theorem 2.

$$c_{f_k}(\mathbf{u}, \mathbf{v}) = (-1)^{\langle \mathbf{v}, \mathbf{k} \rangle} c_f(\mathbf{u}, \mathbf{v}).$$

Theorem 3.

$$c_{F_k}(\mathbf{u}, \mathbf{v}) = \sum_{\mathbf{w} \in \mathbb{F}_2^n} c_{f_k}(\mathbf{u}, \mathbf{w}) c_g(\mathbf{w}, \mathbf{v}) = \sum_{\mathbf{w} \in \mathbb{F}_2^n} (-1)^{\langle \mathbf{w}, \mathbf{k} \rangle} c_f(\mathbf{u}, \mathbf{w}) c_g(\mathbf{w}, \mathbf{v}).$$

Remark. Theorem 3 can be found in [5] formulated and proved in the terms of correlation matrices. A different proof in manner that follows the style of this paper is given in [10]. The proofs of Theorem 2, Theorem 3, and Theorem 4 are also given in Appendix A.

Theorem 4.

$$\sum_{\mathbf{k}\in\mathbb{F}_2^n} c^2_{F_k}(\mathbf{u},\mathbf{v}) = 2^n \sum_{\mathbf{w}\in\mathbb{F}_2^n} c^2_f(\mathbf{u},\mathbf{w}) c^2_g(\mathbf{w},\mathbf{v}).$$

Theorems 2.1–2.3 concern the composition of two functions. They can be generalized to composition of several functions:

$$f = f_r \circ \cdots f_2 \circ f_1, \qquad f_i : \mathbb{F}_2^n \to \mathbb{F}_2^n.$$

Definition 5. A **linear trail** of length r is an ordered set of intermediate masks

$$\theta = (\theta_0 = u, \theta_1, \ldots \theta_r = w).$$

A **linear hull** is the set of all trails starting with the same input mask and ending with the same output mask.

In this paper we deal only with trails of length two and refer the reader to [6, Chapter 7] for more information about the general case.

Another measure which is used for evaluating how well the round function f is approximated by $(\mathbf{u},\mathbf{v})$ is the following

Definition 6. Fourier transformation (or Walsh transform) of f in regard to $(\mathbf{u},\mathbf{v})$ is a function $\hat{f} : \mathbb{F}_2^n \times \mathbb{F}_2^n \to \mathbb{Z}$ defined by

$$\hat{f}(\mathbf{u},\mathbf{v}) = \sum_{\mathbf{x}\in F_2^n} (-1)^{\langle \mathbf{u},\mathbf{x}\rangle + \langle \mathbf{v}, f(\mathbf{x})\rangle} \quad \in [-2^n, 2^n].$$

The Fourier transformation of f gives some advantages when algebraic methods are involved. It is connected with correlation by the following equality.

Proposition 1.

$$c_f(\mathbf{u},\mathbf{v}) = \frac{1}{2^n}\hat{f}(\mathbf{u},\mathbf{v}).$$

Hence, whenever we say correlation distribution in this paper, we understand the distribution of $\hat{f}(\mathbf{u},\mathbf{v})$ in dependence of the masks.

By fixing a basis of $\mathbb{F}_{2^n}$ over $\mathbb{F}_2$ we define a bijection between $\mathbb{F}_{2^n}$ and $\mathbb{F}_2^n$ and can identify f with a permutation polynomial over $\mathbb{F}_{2^n}$. When this basis is self-dual we have $\langle \mathbf{u},\mathbf{v}\rangle = \mathrm{Tr}(uv)$. Such a basis exists for any finite field with characteristic 2, see [14, 5.1.18]. We can obtain some advantages in computer computation if the basis is also normal. Self-dual normal basis exists in $\mathbb{F}_{2^n}$ if $n \not\equiv 0 \pmod 4$, see [14, 5.2.23] and [11].

In this paper we assume that the chosen basis is **self-dual** and **normal**, namely, $\alpha, \alpha^2, \ldots, \alpha^{2^{n-1}}$ with $\mathrm{Tr}(\alpha^{2^i}\alpha^{2^j}) = 0$, where α is a primitive element of $\mathbb{F}_{2^n}$. The fact that the basis is normal has no effect on the theoretical results but is useful for computations.

The nonlinear function f studied here is the polynomial $f(x) = x^a$ with $a = 3.2^s$, s integer, but we will prove some properties in the more general case $a = (2^t + 1).2^s$, s, t positive integers. In order to assure that f is a permutation polynomial we **assume that** n **is odd**.

Since the keys $\mathbf{k}_0$ and $\mathbf{k}_2$ do not change the distribution, for simplicity we do not consider them and denote $\mathbf{k}_1 = \mathbf{k}$. Let $x \in \mathbb{F}_{2^n}$ correspond to the input n-tuple $\mathbf{x}$. Then the output $\bar{\mathbf{x}}$ of the block cipher corresponds to

$$F_k(x) = f(f(x) + k) = (x^a + k)^a.$$

Let $\chi : \mathbb{F}_{2^n} \to \{\pm 1\}$ be the additive character of $\mathbb{F}_{2^n}$ defined by

$$\chi(x) \stackrel{\mathrm{def}}{=} (-1)^{\mathrm{Tr}(x)}.$$

Obviously we have $\chi(x + y) = \chi(x)\chi(y)$ and $\chi(x^2) = \chi(x)$, $x, y \in \mathbb{F}_{2^n}$. Recall also the following well-known property

Proposition 2.

$$\sum_{x \in \mathbb{F}_{2^n}} \chi(x) = 0.$$

Now we can write that

$$\hat{F}_k(\sigma, \omega) = \sum_{x \in \mathbb{F}_{2^n}} (-1)^{\mathrm{Tr}(\sigma x + \omega F_k(x))} = \sum_{x \in \mathbb{F}_{2^n}} \chi(\sigma x + \omega F_k(x)).$$

3 Several general results

Our goal is to determine the distribution of $\hat{F}_k(\sigma, \omega)$ (instead of $c_F(\sigma, \omega)$) depending on the pair of masks (σ, ω) and the key k. To achieve this goal we evaluate $\hat{F}_k^2(\sigma, \omega)$.

Lemma 1. *For any σ and ω of $\mathbb{F}_{2^n}$, and $f : \mathbb{F}_{2^n} \to \mathbb{F}_{2^n}$*

$$\hat{f}^2(\sigma, \omega) = \sum_{x \in \mathbb{F}_{2^n}} \Big[\chi(\sigma x + \omega f(x)) \sum_{y \in \mathbb{F}_{2^n}} \chi(\omega h(x, y)) \Big],$$

where $h(x, y) = f(x + y) - f(x) - f(y)$.

Proof.

$$\begin{aligned}
\hat{f}^2(\sigma, \omega) &= \left(\sum_{y \in \mathbb{F}_{2^n}} \chi(\sigma y + \omega f(y)) \right) \left(\sum_{z \in \mathbb{F}_{2^n}} \chi(\sigma z + \omega f(z)) \right) \\
&= \sum_{y \in \mathbb{F}_{2^n}} \sum_{z \in \mathbb{F}_{2^n}} \chi(\sigma y + \omega f(y)) \chi(\sigma z + \omega f(z)) \\
&= \sum_{y \in \mathbb{F}_{2^n}} \sum_{z \in \mathbb{F}_{2^n}} \chi(\sigma y + \omega f(y) + \sigma z + \omega f(z))
\end{aligned}$$

Substituting $z = x + y$ we obtained

$$\begin{aligned}\hat{f}^2(\sigma,\omega) &= \sum_{x\in\mathbb{F}_{2^n}}\sum_{y\in\mathbb{F}_{2^n}} \chi(\sigma y + \omega f(y) + \sigma(x+y) + \omega f(x+y)) \\ &= \sum_{x\in\mathbb{F}_{2^n}}\sum_{y\in\mathbb{F}_{2^n}} \chi(\sigma x + \omega f(y) + \omega f(x+y)) \\ &= \sum_{x\in\mathbb{F}_{2^n}}\sum_{y\in\mathbb{F}_{2^n}} \chi(\sigma x + \omega f(x) + \omega h(x,y)) \\ &= \sum_{x\in\mathbb{F}_{2^n}}\sum_{y\in\mathbb{F}_{2^n}} \chi(\sigma x + \omega f(x))\chi(\omega h(x,y)),\end{aligned}$$

where $h(x,y) = f(x+y) - f(x) - f(y)$. □

Let us consider the set $\mathcal{H}(x) = \{h(x,y) \mid y \in \mathbb{F}_{2^n}\} \subseteq \mathbb{F}_{2^n}$. For many polynomials $f(x)$ the set $\mathcal{H}(x) = \mathbb{F}_{2^n}$ and thus $\omega\mathcal{H}(x) = \mathbb{F}_{2^n}$ for any $\omega \neq 0$. Then, according to Proposition 2 we have

$$\sum_{y\in\mathbb{F}_{2^n}} \chi(\omega h(x,y)) = 0.$$

Hence $\hat{f}^2(\sigma,\omega)$ (as we will see below) is a sum of a relatively small number of addends and can be evaluated.

Lemma 2. *Let $f : \mathbb{F}_{2^n} \to \mathbb{F}_{2^n}$ and $g(x) = f(x)^{2^t}$. For any σ and ω of $\mathbb{F}_{2^n}$*

$$\hat{g}^2(\sigma,\omega) = \hat{f}^2(\sigma,\mu),$$

where $\mu = \omega^{2^{n-t}}$.

Proof. According to Lemma 1

$$\hat{g}^2(\sigma,\omega) = \sum_{x\in\mathbb{F}_{2^n}} \Big[\chi(\sigma x + \omega g(x)) \sum_{y\in\mathbb{F}_{2^n}} \chi(\omega H(x,y))\Big],$$

where $H(x,y) = g(x+y) + g(x) + g(y) = (f(x+y) + f(x) + f(y))^{2^t} = h(x,y)^{2^t}$.

$$\begin{aligned}\hat{g}^2(\sigma,\omega) &= \sum_{x\in\mathbb{F}_{2^n}} \Big[\chi(\sigma x)\chi(\mu^{2^t} f(x)^{2^t}) \sum_{y\in\mathbb{F}_{2^n}} \chi(\mu^{2^t} h(x,y)^{2^t})\Big] \\ &= \sum_{x\in\mathbb{F}_{2^n}} \Big[\chi(\sigma x)\chi(\mu f(x)) \sum_{y\in\mathbb{F}_{2^n}} \chi(\mu h(x,y))\Big] \\ &= \sum_{x\in\mathbb{F}_{2^n}} \Big[\chi(\sigma x + \mu f(x)) \sum_{y\in\mathbb{F}_{2^n}} \chi(\mu h(x,y))\Big] \\ &= \hat{f}^2(\sigma,\mu),\end{aligned}$$

where $\omega = \mu^{2^t}$. □

As a corollary we obtain the following

Lemma 3. *Let* $f(x) = x^a$ *and* $g(x) = x^{a2^t}$. *Then for any* $\sigma \in \mathbb{F}_{2^n}$

$$\hat{G}_k^2(\sigma, 1) = \hat{F}_\tau^2(\sigma, 1),$$

where $\tau = k^{2^{n-t}}$.

Proof. $G_k = (x^{a2^t} + k)^{a2^t} = (x^a + \tau)^{a2^{2t}} = ((x^a + \tau)^a)^{2^{2t}} = F_\tau^{2^{2t}}(x)$.
Now Lemma 2 gives the statement. □

Lemma 3 can be formulated for an arbitrary ω, not only for $\omega = 1$, but it is not necessary due to the statement given below.

Let $f : \mathbb{F}_{2^n} \to \mathbb{F}_{2^n}$ be a permutation polynomial with the property:

Property 1. For any $\lambda \in \mathbb{F}_{2^n}$ there exist $\eta = \eta(\lambda) \in \mathbb{F}_{2^n}$ such that $\lambda f(x) = f(\eta x)$ for any x.

Let $f, g : \mathbb{F}_{2^n} \to \mathbb{F}_{2^n}$ be permutation polynomials with the above property. Then for any $\sigma, \omega \in \mathbb{F}_{2^n}$ there exist $\lambda(\omega), \eta(\omega) \in \mathbb{F}_{2^n}$ such that

$$\begin{aligned} \sigma x + \omega g(f(x) + k) &= \sigma x + g(\lambda f(x) + \lambda k) = \sigma x + g(f(\eta x) + k_\omega) \\ &= \sigma\eta^{-1} y + g(f(y) + k_\omega), \end{aligned} \tag{1}$$

where $k_\omega = \lambda k$.

Therefore we can formulate the following lemma:

Lemma 4. *For suitable* η *and* k_ω *of* $\mathbb{F}_{2^n}$

$$\hat{F}_k(\sigma, \omega) = \hat{F}_{k_\omega}(\sigma\eta^{-1}, 1).$$

Proof. Let us first note that any permutation polynomial of the form $f(x) = x^a$ satisfies the Property 1 since there exists a unique η such that $\lambda = \eta^a$. Then repeating (1) we get

$$\sigma x + \omega F_k(x) = \sigma x + \omega f(f(x) + k) = \sigma\eta^{-1} y + f(f(y) + k_\omega) = \sigma\eta^{-1} y + F_{k_\omega}(y).$$

Hence

$$\sum_{x \in \mathbb{F}_{2^n}} \chi(\sigma x + \omega F_k(x)) = \sum_{y \in \mathbb{F}_{2^n}} \chi(\sigma\eta^{-1} y + F_{k_\omega}(y)).$$

□

The aforesaid shows that ω has influence on the correspondence between distributions and pairs (σ, k) but not on the structure of the set of distributions. If a given distribution corresponds to (σ, ω, k), it corresponds to $(\sigma\eta^{-1}, 1, k_\omega)$, too. That is, the set of distributions (in regard to (σ, k)) is one and the same for all ω. Therefore the case $\omega = 1$ assures enough generality for our study. From now on we assume $\omega = 1$ and we will follow the ideas presented in [1] in order to evaluate the correlation distribution depending on the triple $(\sigma, 1, k)$.

Lemma 5.

$$\hat{F}_k(\sigma^2, 1) = \hat{F}_{k^{1/2}}(\sigma, 1) \qquad \textit{and} \qquad \hat{F}_{k^2}(\sigma, 1) = \hat{F}_k(\sigma^{1/2}, 1).$$

Proof. Since $F_k(y^2) = (y^{2a} + k)^a = (y^a + k^{1/2})^{2a} = F_{k^{1/2}}(y)^2$ we have

$$\begin{aligned}\hat{F}_k(\sigma^2, 1) &= \sum_{x \in \mathbb{F}_{2^n}} \chi(\sigma^2 x + F_k(x)) = \sum_{y \in \mathbb{F}_{2^n}} \chi(\sigma^2 y^2 + F_k(y^2)) \\ &= \sum_{y \in \mathbb{F}_{2^n}} \chi((\sigma y + F_{k^{1/2}}(y))^2) = \sum_{y \in \mathbb{F}_{2^n}} \chi(\sigma y + F_{k^{1/2}}(y)) = \hat{F}_{k^{1/2}}(\sigma, 1).\end{aligned}$$

The proof of the second equality is similar. □

In the case $a = (2^t + 1)2^s$ the following fact holds.

Lemma 6. *If $f(x) = x^{(2^t+1)2^s}$ then for any $k \in \mathbb{F}_{2^n}$ we have*

$$\hat{F}_{k+1}(\sigma, 1) = -\hat{F}_k(\sigma, 1).$$

Proof. According to Lemma 3 with $a = 2^t + 1$ it is sufficiently to give a proof only when $f(x) = x^{(2^t+1)}$. Since

$$\begin{aligned}F_{k+1}(x) &= (x^{2^t+1} + k + 1)^{(2^t+1)} = [(x^{(2^t+1)} + k)^{2^t} + 1](x^{2^t+1} + k + 1) \\ &= F_k(x) + (x^{2^t+1} + k)^{2^t} + (x^{2^t+1} + k) + 1\end{aligned}$$

and $\chi((x^{2^t+1} + k)^{2^t}) = \chi((x^{2^t+1} + k))$ we get

$$\chi(\sigma x + F_{k+1}(x)) = \chi(\sigma x + F_k(x))\chi((x^{2^t+1} + k))^2\chi(1) = -\chi(\sigma x + F_k(x)).$$

Therefore,

$$\hat{F}_{k+1}(\sigma, 1) = \sum_{x \in \mathbb{F}_{2^n}} \chi(\sigma x + F_{k+1}(x)) = -\sum_{x \in \mathbb{F}_{2^n}} \chi(\sigma x + F_k(x)) = -\hat{F}_k(\sigma, 1).$$

□

The previous lemma shows that for a given σ the values of $\hat{F}_k(\sigma, 1)$ separate in pairs of opposite numbers. Hence the set of correlations values has the form

$$\{D_0 = 0, \pm D_1(\sigma), \pm D_2(\sigma), \ldots, \pm D_s(\sigma)\}. \tag{2}$$

Let $A_0, A_1, \ldots, A_s$ denote the numbers of keys for which $\hat{F}_k^2(\sigma, 1) = D_i^2$, respectively. Obviously A_i are even numbers and the following corollary holds.

Corollary 1.

$$\sum_{i=0}^{s} A_i D_i^2(\sigma) = W_\sigma = \sum_{k \in \mathbb{F}_{2^n}} \hat{F}_k^2(\sigma, 1).$$

Herein we call W_σ the ***distribution weight in respect to*** σ.

4 The number of nonzero trails and the distribution weight in the case $f(x) = x^{2^t+1}$, $t < n$

Let $f : \mathbb{F}_{2^n} \to \mathbb{F}_{2^n}$ be a permutation polynomial and let us approximate it linearly with masks σ and π. Let $g : \mathbb{F}_{2^n} \to \mathbb{F}_{2^n}$ be another permutation polynomial and let us linearly approximate it with masks π and ω.

Proposition 3. *For any $\sigma, \omega \in \mathbb{F}_{2^n}$ the equality*

$$\sum_{k \in \mathbb{F}_{2^n}} \hat{F}_k^2(\sigma, \omega) = \frac{1}{2^n} \sum_{\pi \in \mathbb{F}_{2^n}} \hat{f}^2(\sigma, \pi) \hat{g}^2(\pi, \omega) \tag{3}$$

holds.

Proof. According to Theorem 4 we have

$$\sum_{\mathbf{k} \in \mathbb{F}_2^n} c_{F_k}^2(\mathbf{u}, \mathbf{v}) = 2^n \sum_{\mathbf{w} \in \mathbb{F}_2^n} c_f^2(\mathbf{u}, \mathbf{w}) c_g^2(\mathbf{w}, \mathbf{v}).$$

Taking in account Proposition 1 we obtain

$$\frac{1}{2^{2n}} \sum_{k \in \mathbb{F}_{2^n}} \hat{F}_k^2(\sigma, \omega) = \frac{2^n}{2^{4n}} \sum_{\pi \in \mathbb{F}_{2^n}} \hat{f}^2(\sigma, \pi) \hat{g}^2(\pi, \omega).$$

Multiplying by 2^{2n} we obtain the statement. □

The Proposition shows that we have to evaluate the number of intermediate masks π for which the product $\hat{f}^2(\sigma, \pi)\hat{g}^2(\pi, \omega) \neq 0$.

According to Lemma 1 we have

$$\hat{f}^2(\sigma, \pi) = \sum_{x \in \mathbb{F}_{2^n}} \left[\chi(\sigma x + \pi f(x)) \sum_{y \in \mathbb{F}_{2^n}} \chi(\pi h(x, y)) \right],$$

where $f(x + y) = f(x) + f(y) + h(x, y)$.

If $f(x) = x^{2^t+1}$ then $(x+y)^{2^t+1} = (x^{2^t} + y^{2^t})(x+y)$, thus $h(x, y) = x^{2^t} y + x y^{2^t}$. Hence

$$\begin{aligned} \chi(\pi h(x, y)) &= \chi(\pi x^{2^t} y) + \chi(\pi x y^{2^t}) = \chi(\pi^{2^t} x^{2^{2t}} y^{2^t}) + \chi(\pi x y^{2^t}) \\ &= \chi((\pi^{2^t} x^{2^{2t}} + \pi x) y^{2^t}). \end{aligned}$$

Therefore,

$$\hat{f}^2(\sigma, \pi) = \sum_{x \in \mathbb{F}_{2^n}} \left[\chi(\sigma x + \pi x^{2^t+1}) \sum_{y \in \mathbb{F}_{2^n}} \chi((\pi^{2^t} x^{2^{2t}} + \pi x) y^{2^t}) \right].$$

If $P_\pi(x) = \pi^{2^t} x^{2^{2t}} + \pi x \neq 0$ then $(\pi^{2^t} x^{2^{2t}} + \pi x) y^{2^t}$ runs through all elements of $\mathbb{F}_{2^n}$, thus

$$\sum_{y \in \mathbb{F}_{2^n}} \chi((\pi^{2^t} x^{2^{2t}} + \pi x) y^{2^t}) = \begin{cases} 2^n, & \text{if } \pi^{2^t} x^{2^{2t}} + \pi x = 0 \\ 0, & \text{otherwise} \end{cases}.$$

Therefore,

$$\hat{f}^2(\sigma, \pi) = 2^n \sum_{P_\pi(x)=0} \chi(\sigma x + \pi x^{2^t+1}).$$

Similarly,

$$\hat{g}^2(\pi, 1) = 2^n \sum_{P_1(x)=0} \chi(\pi x + x^{2^t+1}).$$

The equation $P_\pi(x) = \pi^{2^t} x^{2^{2t}} + \pi x = 0$ has 2^d solutions see [4, Theorem 3.1], where $d = (t, n)$ for n odd. Indeed the set of roots coincides with the set $x_0 \mathbb{F}_{2^d}$, where x_0 is an arbitrary root and $\mathbb{F}_{2^d} = \mathbb{F}_{2^t} \cap \mathbb{F}_{2^n}$. One possible value is $x_0 = \pi^{-1/(2^t+1)}$. Note also that

$$\mathbb{F}_{2^d} = \{0, 1, \alpha^q, \alpha^{2q}, \ldots, \alpha^{(2^d-2)q}\},$$

where α is a primitive element of $\mathbb{F}_{2^n}$ and $q = (2^n - 1)/(2^d - 1)$.

Since for any $x = x_0 y \in \mathbb{F}_{2^d}$ we have

$$\sigma x_0 y + \pi (x_0 y)^{2^t+1} = \sigma \pi^{-\frac{1}{2^t+1}} y + \pi \pi^{-1} y^{2^t+1} = \sigma \pi^{-\frac{1}{2^t+1}} y + y^2 \quad (\text{note } d|t)$$

and then

$$\chi(\sigma x + \pi x^{2^t+1}) = \chi(\sigma \pi^{-\frac{1}{2^t+1}} y + y^2) = \chi((\sigma \pi^{-\frac{1}{2^t+1}} + 1) y).$$

Therefore,

$$\begin{aligned} \hat{f}^2(\sigma, \pi) &= 2^n \sum_{y \in \mathbb{F}_{2^d}} \chi((\sigma \pi^{-\frac{1}{2^t+1}} + 1) y), \\ \hat{g}^2(\pi, 1) &= 2^n \sum_{y \in \mathbb{F}_{2^d}} \chi((\pi + 1) y). \end{aligned} \tag{4}$$

4.1 The case $d = 1$.

For odd $n \leq 19$ the value of d differs from 1 only for $n = 9,\ t = 3, 6$; $n = 15,\ t = 3, 6, 9, 12$; and $n = 15,\ t = 5, 10$.

For the rest values $d = 1$ and thus

$$\begin{aligned} \hat{f}^2(\sigma, \pi) &= 2^n (1 + \chi(\sigma \pi^{-\frac{1}{2^t+1}} + 1)) = 2^n (1 - (-1)^{\mathrm{Tr}(\sigma \pi^{-\frac{1}{2^t+1}})}), \\ \hat{g}^2(\pi, 1) &= 2^n (1 + \chi(\pi + 1)) = 2^n (1 - (-1)^{\mathrm{Tr}(\pi+1)}). \end{aligned} \tag{5}$$

Therefore,

$$\hat{f}^2(\sigma,\pi)=\begin{cases}2^{n+1}, & \text{if } \ \mathrm{Tr}(\sigma\pi^{-\frac{1}{2^t+1}})=1\\ 0, & \text{otherwise}\end{cases},$$
$$\hat{g}^2(\pi,1)=\begin{cases}2^{n+1}, & \text{if } \ \mathrm{Tr}(\pi)=1\\ 0, & \text{otherwise}\end{cases}. \tag{6}$$

The product $\hat{f}^2(\sigma,\pi)\hat{g}^2(\pi,1)\neq 0$ if and only if π belongs to the set

$$M_\sigma=\{\pi \mid \mathrm{Tr}(\pi)=\mathrm{Tr}(\sigma\pi^{-\frac{1}{2^t+1}})=1\}.$$

Then according to Proposition 3

$$W_\sigma=\sum_{k\in\mathbb{F}_{2^n}}\hat{F}_k^2(\sigma,1)=\frac{1}{2^n}\sum_{\pi\in M_\sigma}2^{n+1}2^{n+1}.$$

Therefore,

$$W_\sigma=2^{n+2}|M_\sigma| \tag{7}$$

Let us consider the product $(1-(-1)^{\mathrm{Tr}(\pi)})(1-(-1)^{\mathrm{Tr}(\sigma\pi^{-\frac{1}{2^t+1}})})$. It equals $2.2=4$ for $\pi\in M_\sigma$ and zero otherwise. Hence

$$S=\sum_{\pi\in\mathbb{F}_{2^n}}(1-(-1)^{\mathrm{Tr}(\pi)})(1-(-1)^{\mathrm{Tr}(\sigma\pi^{-\frac{1}{2^t+1}})})=4|M_\sigma|.$$

On the other side,

$$\begin{aligned}S&=2^n-\sum_{\pi\in\mathbb{F}_{2^n}}(-1)^{\mathrm{Tr}(\pi)}-\sum_{\pi\in\mathbb{F}_{2^n}}(-1)^{\mathrm{Tr}(\sigma\pi^{-\frac{1}{2^t+1}})}\\&+\sum_{\pi\in\mathbb{F}_{2^n}}(-1)^{\mathrm{Tr}(\pi)}(-1)^{\mathrm{Tr}(\sigma\pi^{-\frac{1}{2^t+1}})}=2^n+\sum_{\pi\in\mathbb{F}_{2^n}}(-1)^{\mathrm{Tr}(\sigma\pi^{-\frac{1}{2^t+1}}+\pi)}\\&=\sum_{\pi\in\mathbb{F}_{2^n}}[1+(-1)^{\mathrm{Tr}(\sigma\pi^{-\frac{1}{2^t+1}}+\pi)}]=2|\{\pi \mid \mathrm{Tr}(\sigma\pi^{-\frac{1}{2^t+1}}+\pi)=0\}|.\end{aligned}$$

Therefore, substituting $\tau=\pi^{\frac{1}{2^t+1}}$ we obtain

Theorem 7.

$$W_\sigma=2^{n+2}|M_\sigma|=2^{n+1}|\{\tau \mid \mathrm{Tr}(\sigma\tau^{-1}+\tau^{2^t+1})=0\}|. \tag{8}$$

In [1] a formula for $|M_\sigma|$ when $\sigma=1$, thus for W_1, is given, namely

Theorem 8 ([1]). *For $\sigma = 1$ and $t = 1$ we have $W_1 = 2^n(2^n + 1 - S_n)$, where S_n is power-sum of roots of $x^4 + x^3 + 2x + 4$ in $\mathbb{C}$.*

Unfortunately, the approach based on algebraic curves over $\mathbb{F}_2$ which is used to find the above formula does not work in the case $\sigma \neq 1$. But for a given $\mathbb{F}_{2^n}$ it is not difficult (for reasonable size of n) to compute the number of τ with $\mathrm{Tr}(\sigma\tau^{-1} + \tau^{2^t+1}) = 0$. It can be done even only by shifting and permuting the elements of binary vectors.

4.2 The case $d > 1$.

For any $x \in \mathbb{F}_{2^n}$ the set $\{\mathrm{Tr}(x\mathbb{F}_{2^d})\}$ is a binary vector of length 2^d and a zero in the first position. Hence the elements of the set $\{\mathrm{Tr}(\mathbb{F}_{2^n}.\mathbb{F}^*_{2^d})\}$ form a binary linear space of vectors of length $2^d - 1$. Indeed it consists of all codewords of the $[2^d - 1,\ d,\ 2^{d-1}]$ simplex code (see [9]) where each codeword is repeated 2^{n-d} times.
Let T denote the set of x, for which $\mathrm{Tr}(x\mathbb{F}_{2^d})$ is the zero vector. Hence $\sum_{y\in\mathbb{F}_{2^d}}(-1)^{\mathrm{Tr}(xy)}$ equals 2^d if $x \in T$ and equals zero otherwise. Obviously $|T| = 2^{n-d}$. Therefore,

$$\hat{f}^2(\sigma,\pi) = \begin{cases} 2^{n+d}, & \text{if } \ \sigma\pi^{-\frac{1}{2^t+1}} + 1 \in T \\ 0, & \text{otherwise} \end{cases}, \qquad (9)$$

$$\hat{g}^2(\pi,1) = \begin{cases} 2^{n+d}, & \text{if } \ \pi + 1 \in T \\ 0, & \text{otherwise} \end{cases}.$$

In this case

$$M_\sigma = \{\pi \mid \pi \in (1+T) \text{ and } \sigma\pi^{-\frac{1}{2^t+1}} \in (1+T)\}.$$

Therefore

$$M_\sigma = (1+T) \cap (1+T)^{-(2^t+1)}\sigma^{(2^t+1)}.$$

Let the symbol α^{C_i} denote the set of all powers whose exponents form the cyclotomic coset C_i of i for 2 modulo $2^n - 1$. For example, $\alpha^{C_1} = \{\alpha, \alpha^2, \ldots, \alpha^{2^{n-1}}\}$.

It is easy to see that T, $1+T$ and $(1+T)^{-(2^t+1)}$ are unions of subsets $\alpha^{C_i} \subset \mathbb{F}_{2^n}$. Unfortunately $(1+T)^{-(2^t+1)}\sigma^{(2^t+1)}$ is not such an union. Nevertheless after determining the set T as a set of powers of the primitive element of $\mathbb{F}_{2^n}$ we can compute W_σ working only with integers modulo $2^n - 1$, that is, we can avoid the computations in $\mathbb{F}_{2^n}$.

Example. Consider the case $n = 9,\ t = 3$, which are the smallest possible parameters. In this case

$$\begin{aligned} T &= \{0\} \cup \alpha^{C_3} \cup \alpha^{C_{19}} \cup \alpha^{C_{23}} \cup \alpha^{C_{79}} \cup \alpha^{C_{85}} \cup \alpha^{C_{111}} \cup \alpha^{C_{119}}, \\ 1+T &= \{1\} \cup \alpha^{C_{39}} \cup \alpha^{C_{45}} \cup \alpha^{C_{47}} \cup \alpha^{C_{51}} \cup \alpha^{C_{91}} \cup \alpha^{C_{171}} \cup \alpha^{C_{223}}, \\ (1+T)^{-(2^t+1)} &= \{1\} \cup \alpha^{C_5} \cup \alpha^{C_{11}} \cup \alpha^{C_{13}} \cup \alpha^{C_{37}} \cup \alpha^{C_{53}} \cup \alpha^{C_{91}} \cup \alpha^{C_{127}}. \end{aligned}$$

Then $(1+T)\cap(1+T)^{-(2^t+1)} = \{1\}\cup\alpha^{C_{91}}$, thus, for $\sigma = 1$ we have $W_1 = 2^{15}\cdot 10$. For $\sigma \in \alpha^{C_1}$ one can find that $W_\sigma = 2^{15}.8$, and so on.

According to Lemma 3 and Lemma 4 the results obtained in this section hold for $f(x) = x^{(2^t+1)2^s}$, too.

5 Correlation distribution in the case $f(x) = x^{3.2^s}$

In this section we determine the possible values for $\hat{F}_k(\sigma,\omega)$ depending on the triple (σ,ω,k). According to Lemma 3 and Lemma 4 we can assume without loss of generality that $f(x) = x^3$ and $\omega = 1$.

Lemma 7. *If $f(x) = x^3$ then*

$$\hat{F_k}^2(\sigma,1) = 2^n \sum_{x:L_k(x)=0} \chi(\sigma x + F_k(x) + k^3), \tag{10}$$

where

$$L_k(x) = x^{64} + (k^{16}+k^4)x^{16} + (k^8+k^2)x^4 + x. \tag{11}$$

Proof. $F_k(x) = (x^3+k)^3$, thus $H(x,y,k) = k^3 + H_1(x,y,k)$, where

$$H_1(x,y,k) = x\,y^8 + kx^2\,y^4 + (k^2x + kx^4)\,y^2 + (x^8 + k^2x^2)\,y.$$

We apply Lemma 1 and since $\chi(H(x,y,k) = \chi(k^3)\chi(H_1(x,y,k))$ we have

$$\hat{F_k}^2(\sigma,1) = \sum_{x\in\mathbb{F}_{2^n}} \Big[\chi(\sigma x + F_k(x) + k^3) \sum_{y\in\mathbb{F}_{2^n}} \chi(H_1(x,y,k))\Big].$$

But using the properties of $\chi(.)$ we can write that

$$\begin{aligned}\chi(H_1(x,y,k)) &= \chi(x\,y^8 + k^2x^4\,y^8 + (k^2x + kx^4)^4\,y^8 + (x^8 + k^2x^2)^8\,y^8)\\ &= \chi([x + k^2x^4 + (k^8x^4 + k^4x^{16}) + (x^{64} + k^{16}x^{16})]\,y^8)\\ &= \chi(L_k(x)y^8).\end{aligned}$$

Hence

$$\hat{F_k}^2(\sigma,1) = \sum_{x\in\mathbb{F}_{2^n}} \Big[\chi(\sigma x + F_k(x) + k^3) \sum_{y\in\mathbb{F}_{2^n}} \chi(L_k(x)y^8)\Big].$$

If $L_k(x) \neq 0$ then $z = L_k(x)y^8$ runs through all elements of $\mathbb{F}_{2^n}$ and thus

$$\sum_{y\in\mathbb{F}_{2^n}} \chi(L_k(x)y^8) = \sum_{z\in\mathbb{F}_{2^n}} \chi(z) = 0.$$

If $L_k(x) = 0$ then

$$\sum_{y\in\mathbb{F}_{2^n}} \chi(L_k(x)y^8) = \sum_{y\in\mathbb{F}_{2^n}} \chi(0) = 2^n.$$

This proves the lemma. □

The polynomial $L_k(x)$ is a linearized polynomial and $L_k(x) = x\,\ell(x^3)$, where

$$\ell(y) = y^{21} + (k^{16} + k^4)y^5 + (k^8 + k^2)y + 1. \tag{12}$$

It coincides with the polynomial $P(x)$ studied in [1] and we can use the results given there.

Lemma 8 ([1]). *The possible number of roots of $L_k(x)$ in $\mathbb{F}_{2^n}$ is 2 or 8.*

Theorem 9. $\hat{F_k}(\sigma, 1) = \{0,\ \pm 2^{\frac{n+1}{2}},\ \pm 2^{\frac{n+3}{2}}\}$.

Proof. According to (10)

$$\hat{F_k}^2(\sigma, 1) = 2^n \sum_{x: L_k(x)=0} \chi(\sigma x + F_k(x) + k^3),$$

Denote $T(x) = \mathrm{Tr}(\sigma x + F_k(x) + k^3)$. Then $\chi(\sigma x + F_k(x) + k^3) = (-1)^{T(x)})$. Note also that $T(0) = 0$.

Let L denote the set of roots of $L_k(x)$ in $\mathbb{F}_{2^n}$. According to the properties of linearized polynomials L is a linear subspace of $\mathbb{F}_{2^n}$ over $\mathbb{F}_2$.

In the case $|L| = 2$, i.e., $L = \{0, x_0\}$, we have

$$\sum \chi(\sigma x + F_k(x) + k^3) = 1 + (-1)^{T(x_0)} = \{0 \text{ or } 2\}.$$

Hence

$$\hat{F_k}^2(\sigma, 1) = \{0 \text{ or } 2^{n+1}\}.$$

Let $|L| = 8$. Since L is a linear subspace over $\mathbb{F}_2$, there are three elements x_1, x_2, x_3 of L which form a basis, that is

$$L = \{x = ax_1 + bx_2 + cx_3 \mid a, b, c \in \mathbb{F}_2\}.$$

Then $T(x) = aT(x_1) + bT(x_2) + cT(x_3) = aT_1 + bT_2 + cT_3$ and we have

$$\sum_{x \in L} \chi(\sigma x + F_k(x) + k^3) = \sum_{(a,b,c)} (-1)^{aT_1}(-1)^{bT_2}(-1)^{cT_3}.$$

Depending on how many T_i are zeros and how many ones, the following cases are possible (up to permutation of a, b, c):

$$\sum_{x \in L} \chi(\sigma x + F_k(x) + k^3) = \begin{cases} \sum 1^a 1^b 1^c \\ \sum 1^a 1^b (-1)^c \\ \sum 1^a (-1)^b (-1)^c \\ \sum (-1)^a (-1)^b (-1)^c \end{cases}.$$

In any case with the exception of the first one, the numbers of 1s and (-1)s are equal, thus the sum is zero. In the first case the sum is equal to 8. Therefore,

$$\hat{F_k}^2(\sigma, 1) = \{0 \text{ or } 2^{n+3}\}.$$

and the proof of the theorem is completed. □

The proof of Lemma 8 shows how to find x_1, x_2, x_3. The cubes x_1^3, x_2^3, x_3^3 are the three roots of $p_1(y) = y^3 + k^2 y + 1$, if $\mathrm{Tr}(k^3) = 1$ and the roots of $p_2(y) = y^3 + (k^2+1)y + 1$ if $\mathrm{Tr}(k^3) = 0$, where $p_1(y)p_2(y)$ divides $\ell(y)$.

As a corollary of previous theorem and (7) we have

$$A_1(\sigma)2^{n+1} + A_2(\sigma)2^{n+3} = 2^{n+2}|M_\sigma|.$$

Hence we can formulate

Theorem 10.

$$A_1(\sigma) + 4A_2(\sigma) = |\{\tau \in \mathbb{F}_{2^n} \mid \mathrm{Tr}(\tau^3 + \sigma\tau^{-1}) = 0\}|.$$

Our main goal is to find the numbers A_1 and A_2 for any σ. After computing the right side of the above equality it is sufficiently to find only A_1 or A_2. Below we describe a method for computing $A_2(\sigma)$.

Theorem 11.

$$A_2(\sigma) = 2\left|\left\{k \in \mathbb{F}_{2^n} \;\middle|\; \begin{array}{l} p_1(y) \text{ has 3 roots } x_1^3,\ x_2^3,\ x_3^3: \\ \mathrm{Tr}(\sigma x_1) = \mathrm{Tr}(\sigma x_2) = \mathrm{Tr}(\sigma x_3) = 1 \end{array}\right\}\right|.$$

Proof. Let x_1, x_2, x_3 be such that the cubes x_1^3, x_2^3, x_3^3 are the three roots of $p_1(y) = y^3 + k^2 y + 1$, if $\mathrm{Tr}(k^3) = 1$ and the three roots of $p_2(y) = y^3 + (k+1)^2 y + 1$ if $\mathrm{Tr}(k^3) = 0$. It is ease to prove that x_1, x_2, x_3 are linearly independent, thus they form a basis of L when $|L| = 8$. The number $A_2(\sigma)$ is equal to the number of keys with $\hat{F_k}^2(\sigma, 1) = 2^{n+3}$. According to (10)

$$\hat{F_k}^2(\sigma, 1) = 2^n \sum_{x \in L} (-1)^{\mathrm{Tr}(\sigma x + F_k(x) + k^3)} = 2^{n+3}$$

if and only if $\mathrm{Tr}(\sigma x_i) = \mathrm{Tr}(F_k(x_i) + k^3)$ for $i = 1, 2, 3$. But it can be proved that $\mathrm{Tr}(F_k(x_i) + k^3) = 1$. Therefore,

$$\hat{F_k}^2(\sigma, 1) = 2^{n+3} \quad \text{iff} \quad \mathrm{Tr}(\sigma x_1) = \mathrm{Tr}(\sigma x_2) = \mathrm{Tr}(\sigma x_3) = 1.$$

Since $p_1(y) \Rightarrow p_2(y)$ by replacing k with $k+1$ and vice versa we can conclude that required number of sets $\{x_1, x_2, x_3\}$ is two times the number of ones obtained by $p_1(y)$.

Note: If x_1^3, x_2^3, x_3^3 are the three roots of $p_1(y)$ for a given k they are the roots of $p_2(y)$ but for $k := k + 1$. Also $\mathrm{Tr}((k+1)^3) = \mathrm{Tr}(k^3 + 1) = 1 + \mathrm{Tr}(k^3)$. □

Algorithm description

$$y \in \mathbb{F}_{2^n}^* : \; p_1(y) = 0 \Leftrightarrow k^2 = y^2 + y^{-1} \Leftrightarrow k = y + y^{2^{n-1}}.$$

While y runs trough $\mathbb{F}_{2^n}^*$ we obtain all k for which $p_1(y)$ has a root. The values of k that appear three times correspond to the case three roots. Also, $p_1(y; k) = y^3 + k^2 y + 1 = 0$ if and only if $p_1(y^2; k^2) = 0$. Hence if $y_0 = \alpha^i$ is a root of

Table 1. Polynomials generating self-dual normal basis

n	generating polynomial
3	x^3+x^2+1
5	$x^5+x^4+x^2+x+1$
7	$x^7+x^6+x^4+x+1$
9	$x^9+x^8+x^6+x^3+x^2+x+1$
11	$x^{11}+x^{10}+x^8+x^7+x^6+x^5+1$
13	$x^{13}+x^{12}+x^{10}+x^7+x^4+x^3+1$
15	$x^{15}+x^{14}+x^{12}+x^9+x^7+x^5+1$
17	$x^{17}+x^{16}+x^{14}+x^9+x^6+x^5+x^4+x^3+1$

$p_1(y;\alpha^j)$ then $p_1(\alpha^{C_i},\alpha^{C_j})=0$, where C_i, C_j are cyclotomic cosets and α is a primitive element of $\mathbb{F}_{2^n}$.

Therefore the set of k for which $p_1(y;k)$ has 3 roots is a union of α^{C_j} and we have to check only for representatives if $\mathrm{Tr}(\sigma x_i)=1$.

6 Computations

In our computations we use self-dual normal basis $\alpha,\alpha^2,\ldots,\alpha^{2^{n-1}}$, where α is a primitive element of $\mathbb{F}_{2^n}$. The list of used generation polynomials (i.e, the irreducible polynomials of α) for odd n is given in Table 1.

Let us consider the set of nonzero input masks as powers of the primitive element $\alpha\in\mathbb{F}_{2^n}$, that is, $\sigma:\ \alpha^0,\alpha,\alpha^2,\ldots,\alpha^{2^n-2}$. Lemma 5 shows that the elements of each set α^{C_i} leads to one and the same distribution. Hence we can compute distribution only for the leaders of the cyclotomic cosets. The same is true for keys k, too.

We have determined the correlation distributions for odd $n\le 17$ but the tables are too large to be given in a paper. We present here only the results for $n=5$ and $n=7$.(Tables 2 and 3)

Table 2. Correlation distribution for $n=5$.

σ	$\#\,\sigma$	$-2^{\frac{n+3}{2}}$	$-2^{\frac{n+1}{2}}$	0	$2^{\frac{n+1}{2}}$	$2^{\frac{n+3}{2}}$
$\sigma=\alpha^{C_0};\alpha^{C_7}$	6	0	6	20	6	0
$\sigma=\alpha^{C_1};\alpha^{C_3}$	10	1	4	22	4	1
$\sigma=\alpha^{C_5};\alpha^{C_{11}}$	10	1	6	18	6	1
$\sigma=\alpha^{C_{15}}$	5	0	8	16	8	0

Table 3. Correlation distribution for $n = 7$.

σ	$\#\ \sigma$	$-2^{\frac{n+3}{2}}$	$-2^{\frac{n+1}{2}}$	0	$2^{\frac{n+1}{2}}$	$2^{\frac{n+3}{2}}$
α^{C_0}	1	0	22	84	22	0
$\alpha^{C_1};\alpha^{C_5};\alpha^{C_9};\alpha^{C_{11}}$	28	3	20	82	20	3
α^{C_3}	7	1	24	78	24	1
α^{C_7}	7	1	26	74	26	1
$\alpha^{C_{13}};\alpha^{C_{19}};\alpha^{C_{21}};\alpha^{C_{63}}$	28	3	22	78	22	3
$\alpha^{C_{15}}$	7	2	24	76	24	2
$\alpha^{C_{23}}$	7	2	20	84	20	2
$\alpha^{C_{27}}$	7	4	22	76	22	4
$\alpha^{C_{29}}$	7	5	16	86	16	5
$\alpha^{C_{31}};\alpha^{C_{43}}$	14	2	22	80	22	2
$\alpha^{C_{47}}$	7	3	18	86	18	3
$\alpha^{C_{55}}$	7	2	28	68	28	2

7 Conclusion

In this paper we study two-round key-alternating block ciphers with nonlinear function $f(x) = x^a$ and prove several facts about their correlation distribution. In the case $a = 2^s(2^t + 1)$ we derive a formula for their distribution weights. The correlation distribution is obtained completely only for $t = 1$, but we hope that the used approach will enable the general case to be also solved. As a future work this analysis can be applied to a recently proposed lightweight block cipher MiMC [2] which uses x^3 as a round function.

Acknowledgements. This research was supported by the Research Council KU Leuven, grant C16/18/004 and C16/15/058, the National Science Fund of Bulgaria, grant DFNI-I02/8, FWO-BAS grant VS.077.18 and SB PhD fellowship at FWO.

References

1. Mohamed Ahmed Abdelraheem, Martin Ågren, Peter Beelen, and Gregor Leander. On the distribution of linear biases: Three instructive examples. In *CRYPTO*, volume 7417 of *Lecture Notes in Computer Science*, pages 50–67. Springer, 2012.

2. Martin R. Albrecht, Lorenzo Grassi, Christian Rechberger, Arnab Roy, and Tyge Tiessen. Mimc: Efficient encryption and cryptographic hashing with minimal multiplicative complexity. In *ASIACRYPT (1)*, volume 10031 of *Lecture Notes in Computer Science*, pages 191–219, 2016.
3. François Arnault, Erik Jarl Pickett, and Stéphane Vinatier. Construction of self-dual normal bases and their complexity. *Finite Fields and Their Applications*, 18(2):458–472, 2012.
4. Robert S. Coulter. On the evaluation of a class of weil sums in characteristic 2. *New Zealand J. Math*, 28:171–184, 1999.
5. Joan Daemen, René Govaerts, and Joos Vandewalle. Correlation matrices. In *FSE*, volume 1008 of *Lecture Notes in Computer Science*, pages 275–285. Springer, 1994.
6. Joan Daemen and Vincent Rijmen. *The Design of Rijndael: AES - The Advanced Encryption Standard*. Information Security and Cryptography. Springer, 2002.
7. Joan Daemen and Vincent Rijmen. Probability distributions of correlation and differentials in block ciphers. *J. Mathematical Cryptology*, 1(3):221–242, 2007.
8. Joan Daemen and Vincent Rijmen. Correlation analysis in $gf(2^n)$. *Cryptology and Information security series*, 7(3):115–131, 2011.
9. W. C. Huffman, Richard A. Brualdi, and Vera S. Pless. *Handbook of Coding Theory*. Elsevier Science Inc., USA, 1998.
10. Liliya Kraleva. Key-dependent correlation distribution for cube cipher. Master's thesis, Sofia University, Bulgaria, 2017.
11. Abraham Lempel and Marcelo J. Weinberger. Self-complementary normal bases in finite fields. *SIAM J. Discret. Math.*, 1(2):193–198, 1988.
12. Rudolf Lidl and Harald Niederreiter. *Finite Fields*. Encyclopedia of Mathematics and its Applications. Cambridge University Press, 2 edition, 1996.
13. Mitsuru Matsui. Linear cryptanalysis method for DES cipher. In *EUROCRYPT*, volume 765 of *Lecture Notes in Computer Science*, pages 386–397. Springer, 1993.
14. Gary L. Mullen and Daniel Panario. *Handbook of Finite Fields*. Chapman Hall/CRC, 1st edition, 2013.

A Proofs

We prove Theorems 2.1–2.3 in terms which correspond better to the style and terms of this paper.

A.1 Proof of Theorem 2

$$\begin{aligned} c_{f_k}(\mathbf{u},\mathbf{v}) &= 2\Pr(\langle \mathbf{u},\mathbf{x}\rangle + \langle \mathbf{v}, f(\mathbf{x})\rangle + \langle \mathbf{v},\mathbf{k}\rangle = 0) - 1 \\ &= 2\Pr(\langle \mathbf{u},\mathbf{x}\rangle + \langle \mathbf{v}, f(\mathbf{x})\rangle = \langle \mathbf{v},\mathbf{k}\rangle) - 1. \end{aligned}$$

Denote $P = \Pr_{\mathbf{x}\in\mathbb{F}_2^n}(\langle \mathbf{u},\mathbf{x}\rangle + \langle \mathbf{v}, f(\mathbf{x})\rangle = 0)$. Then

$$\Pr_{\mathbf{x}\in\mathbb{F}_2^n}(\langle \mathbf{u},\mathbf{x}\rangle + \langle \mathbf{v}, f(\mathbf{x})\rangle = \langle \mathbf{v},\mathbf{k}\rangle) = \begin{cases} P, & \text{if } \langle \mathbf{v},\mathbf{k}\rangle = 0 \\ 1-P, & \text{if } \langle \mathbf{v},\mathbf{k}\rangle = 1 \end{cases}$$

Hence

$$c_{f_k}(\mathbf{u},\mathbf{v}) = \begin{cases} c_f(\mathbf{u},\mathbf{v}), & \text{if } \langle \mathbf{v},\mathbf{k}\rangle = 0 \\ -c_f(\mathbf{u},\mathbf{v}), & \text{if } \langle \mathbf{v},\mathbf{k}\rangle = 1 \end{cases}.$$

□

A.2 Proof of Theorem 3

Let

$$P_{F_k}(\mathbf{u},\mathbf{v}) = \Pr_{\mathbf{x}\in\mathbb{F}_2^n}(\langle\mathbf{u},\mathbf{x}\rangle + \langle\mathbf{v},F_k(\mathbf{x})\rangle = 0),$$
$$P_{f_k}(\mathbf{u},\mathbf{v}) = \Pr_{\mathbf{x}\in\mathbb{F}_2^n}(\langle\mathbf{u},\mathbf{x}\rangle + \langle\mathbf{v},f_k(\mathbf{x})\rangle = 0),$$
$$P_{g}(\mathbf{u},\mathbf{v}) = \Pr_{\mathbf{x}\in\mathbb{F}_2^n}(\langle\mathbf{u},\mathbf{x}\rangle + \langle\mathbf{v},g(\mathbf{x})\rangle = 0).$$

But $f_k(x)$ is bijection, that is, $\{f_k(x) \mid x \in \mathbb{F}_2^n\} = \mathbb{F}_2^n$, thus, we can write

$$\begin{aligned} P_g(\mathbf{u},\mathbf{v}) &= \Pr_{\mathbf{x}\in\mathbb{F}_2^n}(\langle\mathbf{u},f_k(\mathbf{x})\rangle + \langle\mathbf{v},g(f_k(\mathbf{x}))\rangle = 0) \\ &= \Pr_{\mathbf{x}\in\mathbb{F}_2^n}(\langle\mathbf{u},f_k(\mathbf{x})\rangle + \langle\mathbf{v},F_k(\mathbf{x})\rangle = 0). \end{aligned}$$

Since $\langle\mathbf{u},\mathbf{x}\rangle + \langle\mathbf{v},F_k(\mathbf{x})\rangle = 0$ if and only if

$$\langle\mathbf{u},\mathbf{x}\rangle + \langle\mathbf{w},f_k(\mathbf{x})\rangle = \langle\mathbf{w},f_k(\mathbf{x})\rangle + \langle\mathbf{v},F_k(\mathbf{x})\rangle.$$

Let X_w and Y_w be the random binary variables $\langle\mathbf{u},\mathbf{x}\rangle + \langle\mathbf{w},f_k(\mathbf{x})\rangle = 0$ and $\langle\mathbf{w},f_k(\mathbf{x})\rangle + \langle\mathbf{v},F_k(\mathbf{x})\rangle = 0$, respectively. Then according to Piling-up lemma

$$\begin{aligned} \varepsilon_{X_w\oplus Y_w} &= 2(P_{f_k}(\mathbf{u},\mathbf{w}) - \frac{1}{2})(P_g(\mathbf{w},\mathbf{v}) - \frac{1}{2}) \\ &= 2P_{f_k}(\mathbf{u},\mathbf{w})P_g(\mathbf{w},\mathbf{v}) - P_{f_k}(\mathbf{u},\mathbf{w}) - P_g(\mathbf{w},\mathbf{v}) + \frac{1}{2}. \end{aligned}$$

Hence,

$$\begin{aligned} P_{F_k}(\mathbf{u},\mathbf{v}) - \frac{1}{2} &= \sum_{\mathbf{w}\in\mathbb{F}_2^n} \varepsilon_{X_w\oplus Y_w} \\ &= \sum_{\mathbf{w}\in\mathbb{F}_2^n} (2P_{f_k}(\mathbf{u},\mathbf{w})P_g(\mathbf{w},\mathbf{v}) - P_{f_k}(\mathbf{u},\mathbf{w}) - P_g(\mathbf{w},\mathbf{v}) + \frac{1}{2}). \end{aligned}$$

Therefore, we have

$$\begin{aligned} c_{F_k}(\mathbf{u},\mathbf{v}) &= \sum_{\mathbf{w}\in\mathbb{F}_2^n} [4P_{f_k}(\mathbf{u},\mathbf{w})P_g(\mathbf{w},\mathbf{v}) - 2P_{f_k}(\mathbf{u},\mathbf{w}) - 2P_g(\mathbf{w},\mathbf{v}) + 1] \\ &= \sum_{\mathbf{w}\in\mathbb{F}_2^n} (2P_{f_k}(\mathbf{u},\mathbf{w}) - 1)\,(2P_g(\mathbf{w},\mathbf{v}) - 1) \\ &= \sum_{\mathbf{w}\in\mathbb{F}_2^n} c_{f_k}(\mathbf{u},\mathbf{w})c_g(\mathbf{w},\mathbf{v}). \end{aligned}$$

On the other hand,

$$\begin{aligned} \sum_{\mathbf{w}\in\mathbb{F}_2^n} c_{f_k}(\mathbf{u},\mathbf{w})c_g(\mathbf{w},\mathbf{v}) &= \sum_{\mathbf{w}\in\mathbb{F}_2^n} (2P_{f_k}(\mathbf{u},\mathbf{w}) - 1)\,(2P_g(\mathbf{w},\mathbf{v}) - 1) \\ &= \sum_{\mathbf{w}\in\mathbb{F}_2^n} [4P_{f_k}(\mathbf{u},\mathbf{w})P_g(\mathbf{w},\mathbf{v}) - 2P_{f_k}(\mathbf{u},\mathbf{w}) - 2P_g(\mathbf{w},\mathbf{v}) + 1]\,. \end{aligned}$$

Hence,

$$\begin{aligned} c_{F_k}(\mathbf{u},\mathbf{v}) &= 2P_{F_k}(\mathbf{u},\mathbf{v}) - 1 \\ &= \sum_{\mathbf{w}\in\mathbb{F}_2^n} (4P_{f_k}P_g - 2P_{f_k} - 2P_g + 2) - 1 \\ &= \sum_{\mathbf{w}\in\mathbb{F}_2^n} c_{f_k}(\mathbf{u},\mathbf{w})c_g(\mathbf{w},\mathbf{v}). \end{aligned}$$

□

A.3 Proof of Theorem 4

According to the previous theorem,

$$\begin{aligned} c^2_{F_k}(\mathbf{u},\mathbf{v}) &= \left(\sum_{\mathbf{w}\in\mathbb{F}_2^n} (-1)^{\langle \mathbf{w},\mathbf{k}\rangle} c_f(\mathbf{u},\mathbf{w})c_g(\mathbf{w},\mathbf{v})\right) \times \\ &\times \left(\sum_{\mathbf{t}\in\mathbb{F}_2^n} (-1)^{\langle \mathbf{t},\mathbf{k}\rangle} c_f(\mathbf{u},\mathbf{t})c_g(\mathbf{t},\mathbf{v})\right) \\ &= \sum_{\mathbf{w}\in\mathbb{F}_2^n}\sum_{\mathbf{t}\in\mathbb{F}_2^n} (-1)^{\langle \mathbf{k},\mathbf{w}+\mathbf{t}\rangle} c_f(\mathbf{u},\mathbf{w})c_g(\mathbf{w},\mathbf{v})c_f(\mathbf{u},\mathbf{t})c_g(\mathbf{t},\mathbf{v}). \end{aligned}$$

Hence,

$$\sum_{\mathbf{k}\in\mathbb{F}_2^n} c^2_{F_k}(\mathbf{u},\mathbf{v}) = \sum_{\mathbf{w}\in\mathbb{F}_2^n}\sum_{\mathbf{t}\in\mathbb{F}_2^n}\left(\sum_{\mathbf{k}\in\mathbb{F}_2^n} (-1)^{\langle \mathbf{k},\mathbf{w}+\mathbf{t}\rangle}\right) c_f(\mathbf{u},\mathbf{w})c_g(\mathbf{w},\mathbf{v})c_f(\mathbf{u},\mathbf{t})c_g(\mathbf{t},\mathbf{v}).$$

But

$$\sum_{\mathbf{k}\in\mathbb{F}_2^n} (-1)^{\langle \mathbf{k},\mathbf{w}+\mathbf{t}\rangle} = \begin{cases} 0, & \mathbf{w}+\mathbf{t}\neq 0 \\ 2^n, & \mathbf{w}+\mathbf{t}=0 \end{cases}.$$

Therefore,

$$\sum_{\mathbf{k}\in\mathbb{F}_2^n} c^2_{F_k}(\mathbf{u},\mathbf{v}) = 2^n \sum_{\mathbf{w}\in\mathbb{F}_2^n} c_f(\mathbf{u},\mathbf{w})c_g(\mathbf{w},\mathbf{v})c_f(\mathbf{u},\mathbf{w})c_g(\mathbf{w},\mathbf{v}).$$

□

Curriculum Vitae

Liliya Kraleva obtained her bachelor degree in "Applied Mathematics" at 2015 and her master degree in "Discrete and algebraic structures" in 2017 from Sofia University "St. Kliment Ohridski", Sofia, Bulgaria. During her bachelor studies she spent a semester at Linnaeus University, Vaxjo, Sweden through the ERASMUS exchange program, and had a short research visit to COSIC. In 2016 she spent 3 months at COSIC to work on her master thesis. She started her PhD at COSIC in 2018 and was granted an FWO scholarship soon after. Liliya's research was focused on symmetric-key cryptanalysis and she worked on various primitives. Liliya taught the symmetric-key exercises of the course Cryptography and network security from 2018 to 2022.

List of Publications

1. L.Kraleva, M. Mahzoun, R. Posteuca, D. Toprakhisar, T. Ashur, I. Verbauwhede: Cryptanalysis of Strong Physically Unclonable Function, invited paper at IEEE Open Journal of the Solid-State Circuits Society (OJ-SSCS), 2022

2. M. Mahzoun, L. Kraleva, R. Posteuca, T. Ashur: Differential Cryptanalysis of K-Cipher, 27th IEEE International Symposium on Computers and Communications (ISCC) 2022, 11 pages, 2022.

3. L. Kraleva, C. Li: On the SNOW Stream Ciphers, In Proceedings of the 41st Symposium on Information Theory in the Benelux, R. Van Sloun, and B. Škoric (eds.), Werkgemeenschap voor Informatie- en Communicatietheorie, 2021, pp. 116-123.

4. L. Kraleva, R. Posteuca, V. Rijmen: Cryptanalysis of the Permutation Based Algorithm SpoC. INDOCRYPT 2020, Springer-Verlag, LNCS volume 12578, pp. 273-293

5. L. Kraleva, T. Ashur, V. Rijmen: Rotational Cryptanalysis on MAC Algorithm Chaskey. ACNS (1) 2020, Springer-Verlag, LNCS volume 12146, pp. 153-168

6. L. Kraleva, N.L. Manev, V. Rijmen: Correlation Distribution Analysis of a Two-Round Key-Alternating Block Cipher, Tatra Mountains Mathematical Publications 73(1), 2019, pp. 109-130



FACULTY OF ENGINEERING SCIENCE
DEPARTMENT OF ELECTRICAL ENGINEERING
IMEC-COSIC KU LEUVEN
Kasteelpark Arenberg 10, bus 2452
B-3001 Leuven
Liliya.Kraleva@esat.kuleuven.be
http://www.esat.kuleuven.be/cosic/