UNIVERSITÉ DU LUXEMBOURG

PhD-FSTC-2019-25
The Faculty of Sciences, Technology and Communication

DISSERTATION

Defence held on 09/04/2019 in Esch-sur-Alzette

to obtain the degree of

DOCTEUR DE L'UNIVERSITÉ DU LUXEMBOURG

EN INFORMATIQUE

by

Aleksei Nikolaevich UDOVENKO
Born on 12th of January 1992 in Tiksi (Russian Federation)

DESIGN AND CRYPTANALYSIS OF SYMMETRIC-KEY ALGORITHMS IN BLACK AND WHITE-BOX MODELS

Dissertation defence committee

Dr Alex Biryukov, Dissertation Supervisor
Professor, Université du Luxembourg

Dr Gregor Leander
Professor, Ruhr-Universität Bochum

Dr Jean-Sébastien Coron, Chairman
Professor, Université du Luxembourg

Dr Matthieu Rivain
Cryptographer, CryptoExperts

Dr Volker Müller, Vice Chairman
Associate Professor, Université du Luxembourg

Abstract

Cryptography studies secure communications. In symmetric-key cryptography, the communicating parties have a shared secret key which allows both to encrypt and decrypt messages. The encryption schemes used are very efficient but have no rigorous security proof. In order to design a symmetric-key primitive, one has to ensure that the primitive is secure at least against known attacks. During 4 years of my doctoral studies at the University of Luxembourg under the supervision of Prof. Alex Biryukov, I studied symmetric-key cryptography and contributed to several of its topics.

Part I is about the *structural* and *decomposition* cryptanalysis. This type of cryptanalysis aims to exploit properties of the algorithmic structure of a cryptographic function. The first goal is to *distinguish* a function with a particular structure from random, structure-less functions. The second goal is to *recover* components of the structure in order to obtain a *decomposition* of the function. Decomposition attacks are also used to uncover secret structures of S-Boxes, cryptographic functions over small domains. In this part, I describe structural and decomposition cryptanalysis of the Feistel Network structure, decompositions of the S-Box used in the recent Russian cryptographic standard, and a decomposition of the only known APN permutation in even dimension.

Part II is about the *invariant*-based cryptanalysis. This method became recently an active research topic. It happened mainly due to recent "extreme" cryptographic designs, which turned out to be vulnerable to this cryptanalysis method. In this part, I describe an invariant-based analysis of NORX, an authenticated cipher. Further, I show a theoretical study of linear layers that preserve low-degree invariants of a particular form used in the recent attacks on block ciphers.

Part III is about the *white-box* cryptography. In the white-box model, an adversary has full access to the cryptographic implementation, which in particular may contain a secret key. The possibility of creating implementations of symmetric-key primitives secure in this model is a long-standing open question. Such implementations have many applications in industry; in particular, in mobile payment systems. In this part, I study the possibility of applying *masking*, a side-channel countermeasure, to protect white-box implementations. I describe several attacks on direct application of masking and provide a provably-secure countermeasure against a strong class of the attacks.

Part IV is about the *design* of symmetric-key primitives. I contributed to design of the block cipher family SPARX and to the design of a suite of cryptographic algorithms, which includes the cryptographic permutation family SPARKLE, the cryptographic hash function family ESCH, and the authenticated encryption family SCHWAEMM. In this part, I describe the security analysis that I made for these designs.

Acknowledgements

This dissertation would not be possible without the help and support of many people. First of all, I would like to thank my supervisor, Prof. Alex Biryukov, for leading me through my doctoral studies and especially for giving me freedom in the choice of research topics. I would like to thank my thesis supervision committee members, Prof. Jean-Sébastien Coron, and Prof. Volker Müller, for following my research and giving valuable advice. I am grateful to Prof. Gregor Leander and to Dr. Matthieu Rivain for agreeing to serve on my defense as jury members.

I would like to thank the Luxembourg National Research Fund for funding my research (project reference 9037104) and Pierre Fuhrer for coordinating the project. I am grateful for the hospitable country of Luxembourg, the University of Luxembourg and to the Interdisciplinary Centre for Security, Reliability and Trust for providing me an excellent research environment.

It was very exciting and educative for me to work on interesting topics with my coauthors, which include Alex Biryukov, Christof Beierle, Daniel Dinu, Johann Großschädl, Léo Perrin, Luan Cardoso dos Santos, Qingju Wang, Vesselin Velichkov, Yann Le Corre.

I am thankful for all colleagues that I have met at the university for interesting discussions by the cup of coffee: Benoît Cogliati, Brian Shaft, Dag Arne Osvik, Daniel Feher, Dmitry Khovratovich, Giuseppe Vito, Ivan Pustogarov, Moon Sung Lee, Najmeh Soroush, Patrick Derbez, Praveen Vadnala, Ritam Baumik, Sergei Tikhomirov, Shange Fu, Srinivas Vivek, Vitor Pereira. I am also delighted to have met interesting people from other research groups: Antonio Di Maio, Balazs Pejo, Christoph Lambert, Ivana Vukotic, Chista Nadimi, Masoud Tabatabaei, Marharyta Aleksandrova, Vincent Rahli.

I am grateful to the university secretaries and to the doctoral school for their administrative support: Fabienne Schmitz, Claudia Thür, Natali Kirf, Jessica Giro, Céline Lecarpentier.

My warmest gratitude is to my family for their support, especially to my mother, to my wife Asya and to my daughter Nicole.

I would like to acknowledge organizers of various competitions that I participated in: the WhibOx Contest, the SKINNY cryptanalysis competition, the NSUCRYPTO Olympiad and all the Capture-The-Flag competitions that I participated as a part of the teams MSLC and LC↯BC. Competitions always kept me inspired and willing to learn new things.

Finally, I appreciate the work of the developers of the open source software that I used, including the Linux Ecosystem, LaTeX, Python and SageMath [SD19], TikZ for Cryptographers [Jea16], Git.

Contents

Chapter 1

Introduction

In this chapter I give a brief introduction to cryptology and, especially, to symmetric cryptography and cryptanalysis. I provide a high-level overview of this dissertation and I list all publications that I contributed to during my doctoral studies.

1.1 Introduction

Cryptography is the science of secure communication and storage. What does "secure" mean in this definition? First, it means that, apart from the two communicating parties, there may be a third party trying to learn confidential information, to disrupt the communication, or to mislead the communicating parties. Second, "secure" means that a predefined set of goals cannot be achieved by any adversary with set capabilities, such as an ability to read or modify the communications, or limitations, such as having limited computational power.

Cryptography is often divided into design and cryptanalysis. *Cryptographic design* is the design of secure communication systems, called *cryptosystems*. *Cryptanalysis* is *breaking* the security of cryptosystems. In order to understand the security of a cryptosystem better, simplified versions of the cryptosystem are often analyzed. Of course, cryptography and cryptanalysis are not independent. Design of a cryptosystem usually follows alternating steps: design -

cryptanalysis - design - cryptanalysis - ..., until the designers can not cryptanalyze the cryptosystem. After that, the cryptosystem is published and for others to analyze.

Modern cryptography is broadly split into private-key and public-key cryptography, also called symmetric-key and asymmetric-key cryptography.

Symmetric-key cryptography assumes that the communicating parties have a shared private key. For example, they could meet physically and agree on the common secret key. In this case, the same shared key can be used both for encrypting and decrypting communications.

Symmetric-key cryptosystems are usually constructed from low-level, bitwise operations and functions with small domains. They are very efficient. However, their security is not mathematically proven and is not based on any simple mathematical problem.

Asymmetric-key cryptography does not necessarily require pre-shared keys. The defining property, however, is that the key may consist of two parts - a public key and a private key. For example, the public key may be used for encrypting messages and may be openly published. The private key is then used for decrypting the messages and must be kept secret. Asymmetric-key cryptography can also be used to establish a shared secret securely while using an insecure channel.

Asymmetric-key cryptosystems are usually constructed around mathematical structures from number theory or algebraic geometry. Most often these cryptosystems are relatively inefficient. However, their security is based on the hardness of solving a mathematical problem, such as factoring large integers. It means that, if the cryptosystem is cryptanalyzed and broken, then it would mean that the underlying mathematical problem is not hard and can be solved efficiently.

The Public-key cryptography is a very rich area which gives rise to many beautiful cryptosystems and protocols. It is a very active field and many challenging problems are continuously solved and new ones are identified. In this dissertation, however, I dive into symmetric-key cryptography and do not study public-key cryptosystems. As an exception, in Part III, I study white-box cryptography, which, in particular, aims to construct a public-key cryptosystem from a symmetric-key primitive.

In practice, a hybrid method is used. The public-key cryptography is used to establish a shared secret key between the communicating parties, and all the consequent communications are encrypted using fast symmetric-key cryptography.

1.1.1 Authenticated Encryption

The main goal of symmetric cryptography is to provide *authenticated encryption*. Authenticated encryption is a cryptosystem providing *confidentiality*, *integrity*, and *authenticity*.

- *Confidentiality* guarantees that any adversary with predefined capabilities cannot recover any information about the original messages (called plaintexts) from the encrypted messages (called ciphertexts).

- *Integrity* guarantees that any adversary with predefined capabilities cannot modify a transmitted ciphertext without the change being noticed by the receiver.

- *Authenticity* guarantees that the receiving party can be assured that the message was generated by the sender.

There are several ways to construct an authenticated encryption scheme.

Authenticated Encryption from Block Ciphers

Block ciphers are the classical and the most widely used symmetric-key primitives. Formally, a block cipher is a family of permutations, where the secret key selects one of the permutations. The domain of the permutation is the message space.

The most widely spread block cipher is the AES block cipher [DR98], also called Rijndael, designed by Vincent Rijmen and Joan Daemen. It was standardized in 2001 by the US standardization agency NIST.

I and my colleagues designed a block cipher called SPARX [DPU$^+$16]. The design process and analysis are described in Chapter 10.

A plain block cipher can only encrypt fixed-width messages. For example, AES has a 128-bit block size. The bigger problem is that direct encryption of message blocks under the same key (i.e., by the same permutation) leaks information about the equality of message blocks: if the two plaintext blocks are equal, then the two ciphertext blocks are equal too, which contradicts the confidentiality requirement. Another big problem is that authenticity is not guaranteed. The blocks can be removed arbitrarily without being noticed.

An *authenticated block cipher mode* is a construction that uses a block cipher to create an authenticated encryption scheme. Such a construction often consists of two parts: an encryption scheme for confidentiality and a message authentication code (MAC) for integrity. For example, the Encrypt-then-MAC [BN08] is a very generic mode that can combine an arbitrary (secure) encryption scheme and an arbitrary (secure) message authentication code in order to create the authenticated encryption. More specific authenticated encryption modes (e.g. GCM [MV04], OCB [RBBK01]) partially reuse the computations of the two components and achieve better performance. Another class of modes (e.g. SCT [PS16], COPA [ABL$^+$13], POET [AFF$^+$14]) requires a *tweakable block cipher* [LRW02]. Tweakable block ciphers take as input an extra *public* parameter called a *tweak*, and different tweaks should produce indistinguishable block ciphers.

Authenticated Encryption from Stream Ciphers

The *one-time pad* is one of the first modern encryption schemes. It is a very simple cipher, but it is famous for achieving *perfect secrecy*. It means that *no information* is revealed from a ciphertext, even for a computationally-unbounded adversary. This property is also called *information-theoretic* security. The one-time pad accepts an n-bit plaintext and an n-bit key. It combines the key and the plaintext using the exclusive-or operation. The plaintext and the key bits at the same position are added modulo 2. The requirement, however, is that the key should be sampled uniformly at random and used to encrypt only one message. These restrictions are not very practical and are the price for *perfect secrecy*. Indeed, Shannon [Sha49] proved that these restrictions are *necessary* if perfect secrecy must be kept.

Stream ciphers, similarly to block ciphers, discard the perfect secrecy requirement and aim at more practical cryptosystems. Unlike block ciphers, stream ciphers attempt to simulate the one-time pad by generating the required large key (a *keystream*) from a small key on the fly. The keystream is then called *pseudorandom*. The security is based on the requirement that any (computationally-bounded) adversary cannot distinguish the pseudorandom keystream from a purely random sequence.

As in block ciphers, stream ciphers can be combined with a message authentication code (MAC) to create an authenticated encryption (e.g. ChaCha20-Poly1305 [NL18]). Authenticated modes for stream ciphers were studied in [Sar14]. Another approach is to design an authenticated stream cipher from scratch (e.g. ACORN [Wu16]).

Authenticated Encryption from Permutations via Sponge construction

A (cryptographic) *hash function* is a cryptographic primitive that maps a bit-string of arbitrary length into a fixed-length bit-string. For a secure hash function, it should not be computationally easy to invert it (*preimage resistance*) or find two messages that have the same hash value (*collision resistance*). Furthermore, given a fixed message it should be computationally difficult to find another distinct message that has the same hash value (*second preimage resistance*). In general, hash functions are often modeled as *random oracles*. These oracles always return a truly random element of the hash function's codomain, except that for repeated queries with the same message they always return the same hash value. Hash functions are used in a huge number of protocols and public-key constructions. Since hash functions are *keyless*, it is not clear whether they belong to symmetric-key or asymmetric-key cryptography. In practice, hash functions used are made in the symmetric-key style: created from low-level binary operations, very efficient but with heuristic security. However, there exist hash functions from more algebraic constructions, but they are typically only used in theoretic studies due to their inefficiency.

The *sponge* construction was first formally presented in [BDPVA07], though similar ideas had already been used before (e.g. [Bir06, BDPA06]). It was used to design the winner Keccak [BDPVA11] of the SHA-3 [NIS12] hash function competition organized by NIST. The sponge construction uses a primitive called *cryptographic permutation*. The state is divided into the *rate* part and the *capacity* part. The rate part is usually controlled by an adversary, while the capacity part is uncontrolled. The sponge *absorbs* message blocks in-between calls to the permutation. Afterward, it *squeezes* pseudorandom outputs (e.g., hash values) in-between calls to the permutation. The construction is illustrated in Figure 1.1. The sponge is a *provably secure* construction: if the chosen

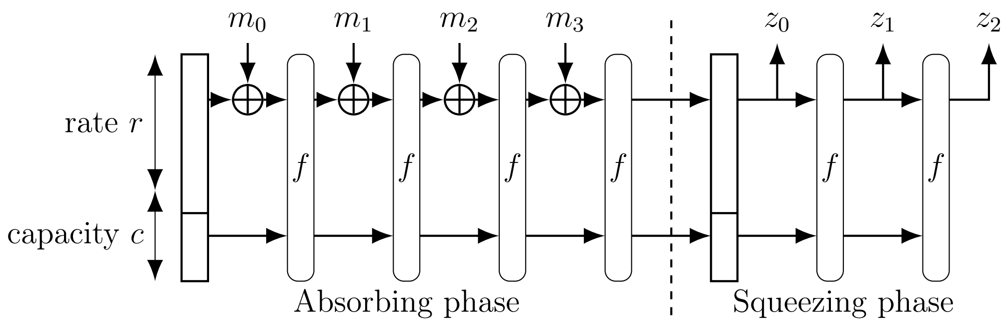


FIGURE 1.1: The Sponge construction for hash functions.

cryptographic permutation is *ideal* (e.g., a purely random permutation), then the construction is guaranteed to be secure up to some level.

The designers of Keccak further showed [BDPA11,BDPVA12b] that sponges can be used to construct authenticated encryption. The latter variant of the mode is called MonkeyDuplex. This mode and its variants were used in several encryption schemes (e.g. [AJN16, BDP+16, DEMS16]). Since a sponge only requires a cryptographic permutation, it inspired cryptography designs called *permutation-based cryptography*. A recent improvement is the Beetle mode [CDNY18], which achieves better security bounds.

I and my colleagues designed a hash function family `Esch` and an authenticated encryption family `Schwaemm`. They are based on the recent sponge-based mode called Beetle and a cryptographic permutation family derived from our SPARX block cipher. These designs are submitted to the NIST Call for Lightweight Cryptography [NIS19]. I describe the design process and analysis of these primitives in Chapter 11.

The CAESAR Competition

Recently, the CAESAR competition was organized [Com19]. Its name stands for “Competition for Authenticated Encryption: Security, Applicability, and Robustness”. The competition started in 2014 when 53 authenticated encryption schemes were submitted. After 5 years of selection process consisting of 3 rounds, the committee selected 8 portfolio members, from which 4 are the preferred choice. The portfolio is split into 3 use cases:

1. *Lightweight applications (resource constrained environments).* The preferred choice is ASCON [DEMS16] which is based on MonkeyDuplex sponge mode. The second choice is ACORN [Wu16], a dedicated authenticated stream cipher.

2. *High-performance applications.* The following two choices are chosen without a preference. The first one is AEGIS-128 [WP16], a dedicated design using a reduced-round AES as a component. The second one is OCB [KR16], a block cipher mode.

3. *Defense in depth.* The preferred choice is Deoxys-II [Jé16], an authenticated encryption scheme based on a tweakable block cipher. The alternative choices are COLM [ABD+], AES-COPA [ABL+13], and ELmD [DN16], block cipher modes.

1.1.2 Black, Gray and White-box Models

Security of cryptosystems is most often analyzed in a *game-based* setting. The game usually happens between a *challenger* and an *adversary.* The challenger possesses secret information, for example, a secret key. The adversary is allowed to ask specified queries to the challenger. The goal of the adversary is to recover the secret information or, at least, a part of it.

Consider an encryption scheme. The challenger flips a coin and decides whether he will use the encryption scheme or its *ideal* equivalent. In the first case, the challenger chooses the secret key uniformly at random. In the second case, the encryption is performed in the best possible way while maintaining the interface and semantics of the encryption scheme. For example, for each plaintext, the ciphertext may be assigned uniformly at random. Note that the challenger is not necessarily an algorithm and usually is not computationally-bounded, unlike the adversary. The game continues. The adversary can ask the challenger to encrypt several plaintexts chosen by the adversary. The challenger performs the encryption (either using the encryption scheme or its idealized version) and gives ciphertexts to the adversary. It is said the adversary is given access to the *encryption oracle.* The adversary finally has to guess, what the outcome of the challenger's coin flip was. That is, the adversary has to decide, whether the encryption was done using the encryption scheme or using its idealized version. If the adversary succeeds with non-negligible probability, then it is said that the encryption scheme has an *adaptive chosen-plaintext distinguisher.* If the adversary accesses the encryption oracle only once, it is said that the scheme has a *(non-adaptive) chosen-plaintext distinguisher.*

There are three major models in which cryptosystems are analyzed.

The Black-box Model

The *black-box* model restricts the analysis to the "functional" side of cryptosystems. An adversary in this model is usually given access to encryption and/or decryption oracles. That is, the adversary is allowed to ask the challenger to encrypt and/or decrypt arbitrary messages. Any intermediate computations or events are not visible to the adversary, thus the name "black-box". This model is fundamental - any weakness in this model is inherited to the gray-box and white-box models.

The Gray-box Model

The *gray-box* model studies the "physical" side of cryptosystems, more precisely, of their *implementations* and the devices on which the implementation is deployed. Indeed, this side provides much more information to the adversary. The adversary may be allowed to measure the time of execution of a query, the power consumption of the device, the electromagnetic radiation. This information is usually referred to as *side-channel* information. The adversary may be *active* - for example, introduce faults in the computations, by heating the device or tweaking the voltage. It is an interesting phenomenon that physical access to the device often enables much more efficient attacks on the cryptosystem. Cryptanalysis in this model is called *side-channel* cryptanalysis.

Countermeasures against *side-channel* attacks may be introduced both in the implementation code and on the physical side. Protections that can be added to the implementation are more generic and, therefore, more preferable. In practice, both methods are used to ensure maximum security. Importantly, implementations typically may use (pseudo)randomness in order to protect computations. Together with the noise and uncertainty of the physical observations, these properties allow the creation of sound countermeasures against side-channel attacks.

The White-box Model

The *white-box* model considers the extreme case when the adversary has *full* access to the implementation, in the form of compiled code or Boolean circuits. Typically, the implementation contains a secret key and the adversary's main goal is to recover it. The hardness of the key recovery is often called the *weak white-box* requirement. Other goals may be considered, such as compressing the implementation, inverting the computed function, or removing hidden "watermarks" allowing the user possessing the implementation to be traced. The respective security properties are called *unbreakability*, *incompressibility*, *one-wayness*, *(traitor) traceability* (see [SWP09, DLPR13]). Unbreakability together with one-wayness result in a public-key scheme, if the embedded secret key allows efficient decryption. Such implementation is also called a *strong white-box*. Indeed, the implementation secure in the white-box model can be seen as a public key, and the embedded secret key can be seen as a private key. A white-box implementation of a common symmetric encryption scheme would then have a very efficient decryption code. However, it turns out to be a challenging, if not impossible problem.

The white-box model was first introduced by Chow *et al.* [CEJvO02b, CEJvO02a] in 2002. The authors proposed rather efficient white-box implementations of the AES and DES block ciphers. Unfortunately, they were broken with practical attacks. All consequent attempts to fix the scheme failed as well. A secure white-box implementation of a block cipher remains an open problem today.

White-box implementations are closely related to the notion of *cryptographic obfuscation*. Indeed, a basic implementation has to be obfuscated in order to hide the secret key. There is an active research direction related to *indistinguishability obfuscation* (iO), which is widely believed to be "the best possible" obfuscation. iO has many applications in theory: it is known that many provably secure cryptographic primitives can be created from secure iO. Unfortunately, many recent iO candidates were broken. Furthermore, all constructions are very inefficient. For example, a recent framework 5Gen-c [CMR17] can be used to obfuscate only a single round of the AES block cipher. I remark though, that there is no established provable link between white-box and iO.

The WhibOx Competition

In 2017, the WhibOx competition [ECR17] was organized. Any person or team in the world could submit a white-box AES-128 implementation in C code of size up to 50 megabytes, then the implementation was publicly available for analysis. The goal was to recover the secret key from the implementation.

Among 94 submissions, most implementations were broken in less than a day. Only 13 implementations required at least one day to be broken, and only 8 of them required at least two days. The winning implementation survived 28 days, and the following implementation only 12 days. The winning design was created by myself and Alex Biryukov. The implementation did not involve any new provable security techniques, but relied on many interesting obfuscation tricks, effectively slowing down the reverse-engineering effort. We were also first to successfully cryptanalyze the best 3 implementations besides ours. Our participation in the competition initiated the research that resulted in Part III of this thesis.

1.1.3 Cryptanalysis of Symmetric-key Primitives

The framework for cryptanalysis is most developed for block ciphers. Indeed, block ciphers were used from the 1970s with the designs of the LUCIFER and DES block ciphers. Together with a proper mode, a block cipher can be used to construct an authenticated encryption scheme. Furthermore, block ciphers tend to have a reasonably simple structure. This simplicity attracts cryptanalysts, who try to break the cipher using both established and novel methods of cryptanalysis. Since the same low-level operations are used in most symmetric-key primitives, cryptanalysis methods for block ciphers are usually very generic and can be applied to other primitives, such as stream ciphers, hash functions, message authentication codes, and authenticated encryption.

What does it mean to break a cipher? In the scientific community, successful cryptanalysis means an algorithm that disproves a security claim of the designers. A typical security claim is that the secret key can not be recovered faster than exhaustive search over the whole key space. A block cipher with a fixed secret key should not be distinguishable from a truly random permutation. Even if the attack is impossible in practice, it only matters that it is faster than the generic attack. The reason is that such an attack shows a *weakness* of the block cipher. Since block ciphers are not provably secure, any weakness should be avoided.

The *complexity* of a cryptanalytic attack is measured by the time, memory and data complexities of the algorithm. The data complexity corresponds to the number of queries that it makes.

In the simplest form, cryptanalytic attacks lead to a *distinguisher* from a random permutation. In most cases, such an attack can be extended into the secret key recovery. This is done by guessing a part of the secret key and decrypting a part of the ciphertext. Then the correctness of the key guess is verified by using the established distinguisher.

Cryptanalysis Methods

In *differential* cryptanalysis, an adversary encrypts two plaintexts with a fixed XOR difference. By an analysis of the cipher's structure, the adversary predicts a difference between ciphertexts with high enough probability. More precisely, the cryptanalyst studies the evolution of the plaintexts difference through all computations, until the ciphertext difference. A transition through nonlinear components is usually probabilistic, and all transitions' probabilities accumulate in an approximation of the probability of observing a particular ciphertext difference.

In *linear* cryptanalysis, an adversary receives many plaintext-ciphertext pairs generated by the analyzed block cipher. The cipher is approximated by *linear* equations, i.e. equations involving only the XOR operation. As in differential cryptanalysis, approximations of nonlinear components induce a cost in the form of probability. As a result, the resulting equations linking the key, the plaintext and the ciphertext hold only with particular probability. If the adversary observes enough data, then correct equations may be established with high probability. In practice, only the ratio of plaintext-ciphertext pairs for which the equation is correct is computed. For a random permutation, this ratio will be close to $1/2$. For a weak block cipher, this ratio may be distinguishable from $1/2$ with high probability.

In *integral* cryptanalysis, the *algebraic degree* of a block cipher is studied. It corresponds to the degree of the multivariate polynomial representation of the cipher. If the algebraic degree is not high enough, the cryptanalyst can deduce a set of plaintexts, for which the corresponding ciphertexts XOR to zero, independently of the secret key. Evaluation of the algebraic degree of a cryptographic primitive is a challenging problem and usually, only upper bounds on the degree can be proved. In Chapter 3 I describe a method to obtain such upper bounds for the particular block cipher structure, called a *Feistel Network*. It is based on the joint work [PU16] with my colleague Léo Perrin.

Integral cryptanalysis is one of the main tools for *structural* cryptanalysis. This branch of cryptanalysis studies ways to distinguish *structures* of cryptographic functions and further decompose the function into components of the structure. It means that only the structure of the function is known to the adversary, and its components are kept secret. The most common structures are the substitution-permutation-network (SPN) and the Feistel network (FN). My colleagues Léo Perrin and Alex Biryukov found an intriguing application of structural and decomposition cryptanalysis. They applied it to *small* functions called S-Boxes, which are used to build cryptographic primitives. S-Boxes are usually represented by tables in specifications and the process of their generation may be kept undisclosed. Structural cryptanalysis allows distinguishing particular structures in an S-Box. Together with analysis of resistance against linear and differential attacks, these methods can often reveal secret criteria behind an S-Box of unexplained origin. This direction is called the *reverse-engineering* of S-Boxes. I contributed to the work of my colleagues in reverse-engineering of the S-Box used in the latest Russian cryptographic standards, and reverse-engineering of an S-Box of a mathematical origin. These results are described in Chapter 4 and Chapter 5 respectively.

A recent direction of cryptanalysis is the search for *invariants* of the cryptographic primitives. *Linear* invariants correspond to a critical flaw in the primitive and are usually easy to avoid. *Nonlinear* invariants are much harder to find. Indeed, the ideas of invariant-based cryptanalysis appeared a long time ago, but the actual applications of the method appeared only recently. A special case of a nonlinear invariant is an *invariant subspace*. Invariant subspace cryptanalysis was introduced in [LAAZ11] and was used to break the PRINTcipher, designed in 2010. Another class of nonlinear invariants is formed by *quadratic* invariants. This class was used in [TLS16] to show a practical distinguisher of recently designed block ciphers Midori, SCREAM, and iSCREAM. In Part II I describe invariant subspaces in NORX, a CAESAR third round candidate; I also show a theoretical study of generalization of quadratic invariants to higher degrees. This part is based on joint work [BUV17] with Alex Biryukov and Vesselin Velichkov, and on joint work [BBU18] with Christof Beierle and Alex Biryukov.

1.2 Thesis Overview

In this section, I provide a brief overview of this dissertation. The introduction, the thesis overview, and the list of publications are given in **Chapter 1** (this chapter). **Chapter 2** introduces definitions and notations, together with well-established facts about mathematical structures that are used throughout the thesis. The rest of the work is split into four parts. Each part corresponds to a separate research area that I contributed to during my doctoral studies.

1.2.1 Part I. Structural and Decomposition Cryptanalysis

In this part, I present my contribution to structural and decomposition cryptanalysis and S-Box reverse-engineering. It consists of three chapters.

Chapter 3 shows an application of structural cryptanalysis to the Feistel Network structure. Our research emerged from observing interesting patterns in the linear approximation table of 4- and 5-round Feistel networks with tiny block size. The linear approximation table is used to measure the resistance of the structure to linear cryptanalysis (see Section 2.4 of Chapter 2). By studying the artifacts, we deduced and proved their relation to integral cryptanalysis. Further, we generalized and proved these integral properties of Feistel networks. As a result, we obtained a simple closed formula on the number of rounds of Feistel networks, when the integral distinguisher is possible. We represented the integral distinguishers in the form of the *high-degree indicator matrix* (HDIM). Further, we showed the usefulness of HDIM as a tool by cryptanalyzing Feistel networks composed with random affine layers. In addition, we proposed a decomposition attack on Feistel networks based on the integral distinguishers.

Chapter 4 describes decompositions of the S-Box used in the recent Russian cryptographic standard. With my coauthors, we discovered interesting and unique structures of the S-Box based on the finite field arithmetic. In the chapter, I describe the step-by-step decomposition process. The methods developed in this work will prove their usefulness in Chapter 5.

Chapter 5 shows the decomposition of the only known APN permutation in even dimension. APN stands for *almost perfect nonlinear* and corresponds to optimal resistance against differential cryptanalysis. The existence of APN permutations in even dimensions was a long-standing problem until the 6-bit APN permutation was published by Dillon *et al.* in 2009. Since then, no new significant progress on the problem was achieved. Furthermore, the method that was used to find the S-Box does not provide any insight on how to generalize the APN permutation. With Alex Biryukov and Léo Perrin, we applied the methods of S-Box reverse-engineering to this S-Box of mathematical origin and surprisingly discovered an interesting structure, which we called a *butterfly*. We studied its properties and generalized it to higher dimensions. Even though we did not find any new APN permutation in even dimension, the generalized butterfly is only slightly weaker than APN permutations.

1.2.2 Part II. Nonlinear Invariants

In this part, I describe my contribution to the method of cryptanalysis based on nonlinear invariants. It consists of two chapters.

Chapter 6 describes an analysis of the core permutation of the NORX authenticated encryption scheme, a third round CAESAR candidate. First, I describe invariant subspaces of the permutation obtained from rotational symmetry of the structure. Second, I show probabilistic invariant subspaces obtained from rotational word symmetry. To illustrate the dangers of such properties, I describe two attacks on slightly modified variants of NORX. Further, I provide the cycle decomposition of a 32-bit mapping G used in the NORX8 instance. I propose an algorithm for the search of low-degree non-linear invariants from a cycle decomposition and apply it to G. The results show that there are no low-degree invariants of G holding with probability one. This chapter is based on the joint work [BUV17] with Alex Biryukov and Vesselin Velichkov, which is currently available as an online report.

Chapter 7 shows a theoretical study of linear layers that preserve a particular class of low-degree invariants. It is a generalization of the theorem from [TLS16], where it was shown that orthogonal linear layers preserve a particular class of quadratic invariants. Our study shows that no *bijective* linear layers preserve a similar class of cubic invariants. However, there are such *expanding* linear layers. The linear layers that are studied, correspond to subsets of $\mathbb{F}_2^n$ on which every Boolean function of algebraic degree at most d, sums to zero. Furthermore, these sets of vectors must have full rank. We call them degree-d zero-sum sets of full rank. The rest of the chapter is devoted to studying the minimum possible size of such sets. This size is related to the minimum expansion rate of the corresponding linear layers. I describe several nontrivial bounds for the minimum possible size of full-rank degree-d zero-sum sets. This chapter is based on the joint work [BBU18] with Christof Beierle and Alex Biryukov, which is in the process of submission to a Boolean function journal.

1.2.3 Part III. White-box Cryptography

In this part, I describe my contribution to symmetric cryptography in the white-box model. This part is based on the joint work [BU18a] with Alex Biryukov. It consists of two chapters.

Chapter 8 describes several attacks on white-box implementations using masking schemes. Recently, Bos *et al.* [BHMT16] showed that most public white-box implementations can be broken in an automated way by a classic side-channel attack. It is therefore natural to apply side-channel protection - masking - to protect white-box implementations. However, this chapter shows many caveats that appear in the white-box setting. The described attacks result in constraints that a secure white-box implementation based on masking has to satisfy. In particular, the classic Boolean masking of any order is not secure since it is a linear scheme.

Chapter 9 describes a general methodology for securing a white-box implementation. The attacks are split into two groups, and each group corresponds to a separate component of protection. The two components are called *structure hiding* and *value hiding*. Structure hiding protection must hide structural patterns and prevent locating of critical computation points in the circuit by graph-based analysis. It also should include protection against fault attacks, though this may be considered as a separate component. Value hiding protection must prevent attacks based on analysis of values computed in the white-box circuit, such as side-channel power analysis attacks. In our research, we focused on the value hiding protection, in particular on the novel attack against Boolean masking. We develop a security model and a game-based security definition. Further, we develop a framework of provable security against the attack. Finally, we propose a novel quadratic masking scheme instantiating the developed framework of provable security. We implement AES-128 encryption protected by the novel masking scheme together with the classic Boolean masking scheme to estimate the overhead.

1.2.4 Part IV. Symmetric Algorithm Design

In this part, I describe my contribution to the design of symmetric-key algorithms. It consists of two chapters.

Chapter 10 describes SPARX, a lightweight block cipher designed by my colleagues and me. The cipher follows a novel design strategy called a *long-trail strategy*. This strategy provides provable security against single-trail linear and differential cryptanalysis for ARX-based structures, where the classic *wide-trail* strategy is not efficient. I developed two algorithms for long-trail evaluation of a given structure. We evaluated a large class of linear layer structures for the block cipher. During the evaluation, my algorithms were used in order to measure the resistance of the linear layer against linear and differential attacks. I also evaluated the linear layer candidates for resistance against integral attacks using the division property, a state-of-the-art technique. The final choice was done by finding an optimal ratio between the two parameters, and also considering implementation properties. It turned out that a linear Feistel round leads to the best compromise between the parameters. This chapter is based on the joint work [DPU+16] with Daniel Dinu, Léo Perrin, Vesselin Velichkov, Johann Großschädl, and Alex Biryukov.

Chapter 11 describes a suite of symmetric-key algorithms. SPARKLE is a family of three cryptographic permutations motivated by the SPARX design. ESCH is a family of two hash functions built using the sponge construction and the SPARKLE permutations. SCHWAEMM is a family of authenticated encryption algorithms, built using the recent Bettle sponge-based mode and the SPARKLE permutations. I performed various analyses of the SPARKLE permutation and its components, including nonlinear invariant analysis, a linearization study of the ARX-based S-Box, evaluation of resistance against integral attacks. Furthermore, I propose a generic algorithm for building the matrix of transitions of truncated differential trails through the linear layer. The algorithm takes as input the binary matrix of the linear layer and computes precise probabilities. The advantage of this method is that it automatically takes into account all dependencies between computations in the linear layer, helping to avoid possible mistakes. Finally, I propose several attacks on reduced-round versions of SCHWAEMM. The suite of primitives is submitted to the NIST Call for Lightweight Cryptographic Algorithms [NIS19]. It is a joint work with Christof Beierle, Alex Biryukov, Luan Cardoso dos Santos, Johann Großschädl, Léo Perrin, Vesselin Velichkov and Qingju Wang.

1.3 Publications

Most of my research results were peer-reviewed and published in conference proceedings.

Journal Publications

[PU17] Léo Perrin and Aleksei Udovenko. Exponential S-Boxes: a Link Between the S-Boxes of BelT and Kuznyechik/Streebog. *IACR Trans. Symmetric Cryptol.*, 2016(2):99–124, 2017.

Conference Proceedings

[BDCU17] Alex Biryukov, Daniel Dinu, Yann Le Corre, and Aleksei Udovenko. Optimal First-Order Boolean Masking for Embedded IoT Devices. In *Smart Card Research and Advanced Applications - 16th International Conference, CARDIS 2017, Lugano, Switzerland, November 13-15, 2017, Revised Selected Papers*, pages 22–41, 2017.

[BPU16] Alex Biryukov, Léo Perrin, and Aleksei Udovenko. Reverse-Engineering the S-Box of Streebog, Kuznyechik and STRIBOBr1. In *Advances in Cryptology - EUROCRYPT 2016 - 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part I*, pages 372–402, 2016.

[BU18] Alex Biryukov and Aleksei Udovenko. Attacks and Countermeasures for White-box Designs. In *Advances in Cryptology - ASIACRYPT 2018 - 24th International Conference on the Theory and Application of Cryptology and Information Security, Brisbane, QLD, Australia, December 2-6, 2018, Proceedings, Part II*, pages 373–402, 2018.

[DPU+16] Daniel Dinu, Léo Perrin, Aleksei Udovenko, Vesselin Velichkov, Johann Großschädl, and Alex Biryukov. Design Strategies for ARX with Provable Bounds: Sparx and LAX. In *Advances in Cryptology - ASIACRYPT 2016 - 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part I*, pages 484–513, 2016.

[PU16] Léo Perrin and Aleksei Udovenko. Algebraic Insights into the Secret Feistel Network. In *Fast Software Encryption - 23rd International Conference, FSE 2016, Bochum, Germany, March 20-23, 2016, Revised Selected Papers*, pages 378–398, 2016.

[PUB16] Léo Perrin, Aleksei Udovenko, and Alex Biryukov. Cryptanalysis of a Theorem: Decomposing the Only Known Solution to the Big APN Problem. In *Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part II*, pages 93–122, 2016.

Unrefereed Publications

[BBU18] Christof Beierle, Alex Biryukov, and Aleksei Udovenko. On Degree-d Zero-Sum Sets of Full Rank. Cryptology ePrint Archive, Report 2018/1194, 2018. https://eprint.iacr.org/2018/1194.

[BUV17] Alex Biryukov, Aleksei Udovenko, and Vesselin Velichkov. Analysis of the NORX Core Permutation. Cryptology ePrint Archive, Report 2017/034, 2017. https://eprint.iacr.org/2017/034.

Presentations

I gave talks at the following conferences and workshops.

1. Fast Software Encryption 2016, Bochum, Germany. Presentation of [PU16].
2. CRYPTO 2016, Santa-Barbara, USA. Presentation of [PUB16].
3. Early Symmetric Crypto 2017, Canach, Luxembourg. Presentation of [PU17].
4. Grande Region Security and Reliability Day 2017, Luxembourg, Luxembourg. Presentation of [DPU+16].
5. CARDIS 2017, Lugano, Switzerland. Presentation of [BDCU17].
6. ASIACRYPT 2018, Brisbane, Australia. Presentation of [BU18].

Source Code

The code for white-box implementations and analysis that I wrote for [BU18] is publicly available online [BU18b]:

https://github.com/cryptolu/whitebox

Chapter 2

Preliminaries and Framework

In this chapter, I describe the necessary background required to understand the thesis, as well as notations and definitions used. The framework of Boolean functions is extensively used. The contents of this chapter are rather dense and cover only the notions used in this thesis. For a more detailed source about Boolean functions, I refer to [Car10a, Car10b].

Common mathematical notations are used. The notation $:=$ means "by definition". $\mathbb{Z}$ denotes the set of integers, $\mathbb{Z}_+$ denotes the set of positive integers, $\mathbb{Z}_{\geq 0} := \{0\} \cup \mathbb{Z}_+$ denotes the set of non-negative integers. For $n \in \mathbb{Z}_+$, $\mathbb{Z}_n$ denotes the set $\{0, 1, 2, \ldots, n-1\}$. For $a, b \in \mathbb{Z}, a \leq b$, $[a \ldots b]$ denotes the tuple of integers $(a, a+1, \ldots, b-1, b)$. $\binom{n}{\leq k}$ denotes the sum $\sum_{i=0}^{k} \binom{n}{i}$.

2.1 Boolean Functions

2.1.1 Binary Fields and Functions

Let $\mathbb{F}_2$ denote the finite field with two elements. For a positive integer n let $\mathbb{F}_2^n$ denote the vector space over $\mathbb{F}_2$ of dimension n. An n-bit *Boolean function* is a function mapping $\mathbb{F}_2^n$ to $\mathbb{F}_2$. The set of all n-bit Boolean functions is denoted by $\mathcal{BF}_n$. The *value vector* $\mathcal{V}_f$ of a Boolean function f is the vector of length 2^n consisting of the values of f on all possible inputs in the lexicographic order. $\mathbf{0}, \mathbf{1}$ denote the two constant functions.

For $n \in \mathbb{Z}_+$ let $\mathbb{F}_{2^n}$ denote the field with 2^n elements. Such field is defined as the set of polynomials with coefficients from $\mathbb{F}_2$ and degree at most $n-1$. The field addition is the usual addition of polynomials, and the field multiplication is the multiplication of polynomials modulo a fixed irreducible polynomial of degree n. It can be summarized by the isomorphism

$$\mathbb{F}_{2^n} \simeq \mathbb{F}_2[X]/P(x),$$

where $P(x)$ is an irreducible polynomial, i.e. $P(x)$ cannot be factored into polynomials of strictly lower degree.

2.1.2 Vectors and Weights

Elements in vectors are indexed starting from 1. For a vector v from $\mathbb{F}_2^n$ it is written $v = (v_1, \ldots, v_n)$. $|X|$ denotes the size of the vector/set X. The *weight* of a vector v is the number of nonzero entries in it and is denoted $\mathbf{wt}(v)$. *Weight* of a Boolean function is the weight of its value vector. An n-bit Boolean function is said to be *balanced*, if its weight is equal to 2^{n-1}.

The *correlation* of a vector $v \in \mathbb{F}_2^t$ is defined as

$$\mathbf{cor}(v) \coloneqq 2 \cdot \mathbf{wt}(v)/t - 1, \quad -1 \le \mathbf{cor}(v) \le 1.$$

The *correlation* of a Boolean function $f \colon \mathbb{F}_2^n \to \mathbb{F}_2$ is defined as the correlation of its value vector $\mathcal{V}_f$:

$$\mathbf{cor}(f) \coloneqq \mathbf{cor}(\mathcal{V}_f) = \mathbf{wt}(f)/2^{n-1} - 1.$$

For any $n \in \mathbb{Z}_+$, $I_n \in \mathbb{F}_2^n$ denotes the all-one vector $(1, 1, \ldots, 1)$. For $j \in \mathbb{Z}_+, 1 \le j \le n$, the j-th unit vector e_j is the vector having 1 at position j and 0 otherwise. $e_1, \ldots, e_n$ form a linear basis of $\mathbb{F}_2^n$.

2.1.3 Bit-wise Arithmetic

Let $\wedge, \vee, \oplus, \neg$ denote the Boolean operations AND, OR, XOR and NOT respectively. The corresponding operations on $\mathbb{F}_2^n$ are defined component-wise, e.g.

$$(x_1, \ldots, x_n) \wedge (y_1, \ldots, y_n) \coloneqq (x_1 \wedge y_1, \ldots, x_n \wedge y_n).$$

The operation of addition modulo 2^w is denoted $\boxplus$, and w should be clear from the context; the bits in a vector are ordered in the decreasing order of significance (see Section 2.1.4). The rotations a vector x to the left and to the right are denoted by $\lll$ and $\ggg$ respectively.

For $x, y \in \mathbb{F}_2^n$ the *inner product* of x and y is defined as

$$\langle x, y \rangle \coloneqq \bigoplus_{i=1}^{n} x_i y_i \in \mathbb{F}_2.$$

This notion is generalized to more arguments. Let $x_1, \dots, x_d \in \mathbb{F}_2^n$. Then define

$$\langle x_1, \dots, x_d \rangle \coloneqq \bigoplus_{i=1}^{n} \prod_{j=1}^{d} x_{j,i} \in \mathbb{F}_2.$$

For $x, y \in \mathbb{F}_2^n$, x^y is defined as (note $0^0 = 1$)

$$x^y \coloneqq x_1^{y_1} x_2^{y_2} \dots x_n^{y_n} \coloneqq x \vee \neg y = I_n \oplus y \wedge (x \oplus I_n) \in \mathbb{F}_2.$$

Let $\preceq$ be the partial relation on $(\mathbb{F}_2^n)^2$ defined by $x \preceq y$ if and only if, for all $i \in \{1, \dots, n\}$, $x_i \leq y_i$. I remark that

$$x \preceq y \;\Leftrightarrow\; y^x = 1 \;\Leftrightarrow\; (\neg x)^{\neg y} = 1.$$

2.1.4 Implicit Isomorphisms

For any $n, m \in \mathbb{Z}_+$, the vector spaces $\mathbb{F}_2^{n+m}$ and $\mathbb{F}_2^n \times \mathbb{F}_2^m$ are considered to be the same with an implicit isomorphism splitting an $(n+m)$-bit vector $v \in \mathbb{F}_2^{n+m}$ into two components: n leftmost bits $l \in \mathbb{F}_2^n$ and m rightmost bits $r \in \mathbb{F}_2^m$.

For any $n \in \mathbb{Z}_+$, the vectors from $\mathbb{F}_2^n$ can be implicitly represented as integers, such that the leftmost bits correspond to the most significant bits. Let $v \in \mathbb{F}_2^n$. Then, by abuse of notation, it can be written:

$$v = (v_1, \dots, v_n) \in \mathbb{F}_2^n, \quad \Leftrightarrow \quad v = \sum_{i=1}^{n} v_i 2^{n-i} \in \mathbb{Z}_n.$$

A hexadecimal vector notation may be used and indicated by a monospace font, for example

$$163 \in \mathbb{Z}_{256} = \mathtt{A3} \in \mathbb{F}_2^8 = (1,0,1,0,0,0,1,1) \in \mathbb{F}_2^8.$$

Another implicit isomorphism is allowed between the vector space $\mathbb{F}_2^n$ and the polynomial ring $\mathbb{F}_2[X]$:

$$v = (v_1, \dots, v_n) \in \mathbb{F}_2^n \quad \Leftrightarrow \quad \sum_{i=1}^{n} v_i X^{n-i} \in \mathbb{F}_2[X].$$

For example,

$$\mathtt{A3} \in \mathbb{F}_2^8 = (X^7 + X^5 + X + 1) \in \mathbb{F}_2[X]$$

Assuming that an irreducible polynomial $P(x)$ defining

$$\mathbb{F}_{2^n} \simeq \mathbb{F}_2[X]/(P(x))$$

is clear from the context, the multiplication operation in the finite field is denoted $\odot$. The division in the finite field is denoted by $\oslash$.

2.1.5 Algebraic Normal Form

Any Boolean function $f\colon \mathbb{F}_2^n \to \mathbb{F}_2$ has a unique representation of the form

$$f(x) = \bigoplus_{u \in \mathbb{F}_2^n} a_u x^u, \;\; f(x) \in \mathbb{F}_2[x_1, \dots, x_n]/(x_1^2 + x_1, \dots, x_n^2 + x_n)$$

called the *algebraic normal form (ANF)*. Here x^u is a shorthand for $x_1^{u_1} \dots x_n^{u_n}$ and such products are called *monomials*. Let $\rho_u[f] \in \mathbb{F}_2$ denote the coefficient of the monomial x^u in the ANF of f. It can be computed by the Möbius transform:

$$\rho_u[f] \coloneqq a_u = \bigoplus_{z \in \mathbb{F}_2^n, z \preceq u} f(z).$$

The *algebraic degree* of a Boolean function f is the maximum Hamming weight of all u such that $a_u = 1$. Equivalently, it is the maximum degree of a monomial in the ANF of f. It is denoted $\deg f$. The zero-function is said to have the algebraic degree $-\infty$. The set of all Boolean functions with n input bits and degree at most d is denoted by $\mathcal{BF}_{n,d}$. A Boolean function of algebraic degree at most 1 is called an *affine* function. An affine Boolean function f is said to be *linear* if $f(0) = 0$. Any affine Boolean function $f\colon \mathbb{F}_2^n \to \mathbb{F}_2$ can be expressed as $f(x) = \langle a, x \rangle + c$ for unique $a \in \mathbb{F}_2^n$ and $c \in \mathbb{F}_2$, where $c = 0$ if and only if f is linear.

2.1.6 Derivatives

For a Boolean function $f\colon \mathbb{F}_2^n \to \mathbb{F}_2$ and a vector $\alpha \in \mathbb{F}_2^n$, I denote the function $\delta_\alpha f\colon \mathbb{F}_2^n \to \mathbb{F}_2$ to be the *derivative* of f with respect to α, given by

$$\delta_\alpha f(x) \coloneqq f(x) \oplus f(x \oplus \alpha).$$

It is well known that $\deg \delta_\alpha f \leq \max(-1, \deg f - 1)$ for any Boolean function f and any α, see [Lai94]. The derivation can be iterated multiple times resulting in a *higher-order derivative*. For d linearly independent vectors $\alpha_1, \dots, \alpha_d \in \mathbb{F}_2^n$ it holds that

$$\delta_{\alpha_1} \dots \delta_{\alpha_d} f(x) = \bigoplus_{z \in \operatorname{span}(\alpha_1, \dots, \alpha_d)} f(x \oplus z).$$

If the vectors $\alpha_1, \dots, \alpha_d$ are linearly dependent, then the derivative is equal to zero.

2.2 Vectorial Boolean Functions

A *Vectorial* Boolean function S is a function mapping $\mathbb{F}_2^n$ to $\mathbb{F}_2^m$ for some positive integers n, m. When n is relatively small, such functions are often called *S-Boxes*. Each output bit of a vectorial Boolean function naturally defines a Boolean function. The corresponding m Boolean functions are called *coordinates* of S. For any nonzero $a \in \mathbb{F}_2^m$ the mapping $x \mapsto \langle a, S(x) \rangle$ is called a *component* of S and is denoted by S_a. A component is a linear combination of coordinates of S. The function S is said to be *balanced*, if each $y \in \mathbb{F}_2^m$ has exactly 2^{n-m} preimages. In particular, S is a bijection if and only if $m = n$ and S is balanced.

A vectorial function $S\colon \mathbb{F}_2^n \to \mathbb{F}_2^m$ can be given by the vector of its values using the following notation:

$$\mathsf{LookupTable(S)} \coloneqq (S(0), S(1), \dots, S(2^n - 1)), \text{ where } S(x) \in \mathbb{F}_2^m.$$

The *algebraic degree* of a vectorial Boolean function is defined to be the maximum algebraic degree of its coordinates.

For any $n \in \mathbb{Z}_+$ the following maps are defined:

$$\begin{aligned}
&\mathsf{left}\colon \mathbb{F}_2^n \times \mathbb{F}_2^n \to \mathbb{F}_2^n, \quad (a, b) \mapsto a,\\
&\mathsf{right}\colon \mathbb{F}_2^n \times \mathbb{F}_2^n \to \mathbb{F}_2^n, \quad (a, b) \mapsto b,\\
&\mathsf{swap}\colon \mathbb{F}_2^n \times \mathbb{F}_2^n \to \mathbb{F}_2^n \times \mathbb{F}_2^n, \quad (a, b) \mapsto (b, a).
\end{aligned}$$

2.2.1 Linear maps

The vectors from $\mathbb{F}_2^n$ are considered as column vectors. The transpose of a vector or matrix v is denoted $v^\top$. The $n \times n$ identity matrix is denoted $I_{n\times n}$.

A vectorial Boolean function $S\colon \mathbb{F}_2^n \to \mathbb{F}_2^m$ is called *linear* (resp. *affine*) if all its coordinates are linear (resp. affine). If S is affine, then it can be expressed as $S(x) = A \times x \oplus b$ for a unique $m \times n$ matrix A over $\mathbb{F}_2$ and $b = S(0) \in \mathbb{F}_2^m$, where $b = 0$ if and only if S is linear.

For $m, n \in \mathbb{Z}_+$, the set of all $m \times n$ matrices over $\mathbb{F}_2$ is denoted $\mathbb{F}_2^{m\times n}$. Any such matrix M defines a linear map from $\mathbb{F}_2^n$ to $\mathbb{F}_2^m$, given by $x \mapsto M \times x$. The set of all bijective linear maps are denoted $\mathsf{GL}_n(\mathbb{F}_2) \subseteq \mathbb{F}_2^{n\times n}$. The set of all bijective affine maps is denoted $\mathsf{GA}_n(\mathbb{F}_2)$.

2.2.2 Equivalence Notions

There are several important notions of *equivalence* between vectorial Boolean functions. Let $S_1, S_2\colon \mathbb{F}_2^n \to \mathbb{F}_2^m$ be vectorial Boolean functions. Let

$$\Gamma_1 = \{(x, S_1(x)) \mid x \in \mathbb{F}_2^n\} \subseteq \mathbb{F}_2^{n+m},$$

$$\Gamma_2 = \{(x, S_2(x)) \mid x \in \mathbb{F}_2^n\} \subseteq \mathbb{F}_2^{n+m}$$

be the functional graphs of S_1 and S_2 respectively.

- S_1, S_2 are *linear* (resp. *affine*) equivalent if there exist linear (resp. affine) mappings A, B such that $S_2 = B \circ S_1 \circ A$.
- S_1, S_2 are *extended-affine* equivalent (EA-equivalent) if there exist affine mappings A, B, C such that $S_2 = B \circ S_1 \circ A \oplus C$.
- S_1, S_2 are *CCZ-equivalent* if there exists an affine mapping L such that $\Gamma_2 = L(\Gamma_1) \coloneqq \{L(x) \mid x \in \Gamma_1\}$, i.e. the functional graphs of S_1 and S_2 are affine equivalent.

2.3 Set Indicators and Subspaces

Let $V \subseteq \mathbb{F}_2^n$. The *indicator* of the set V is defined as

$$\mathbb{1}_V \colon \mathbb{F}_2^n \to \mathbb{F}_2,$$
$$\mathbb{1}_V(x) := \begin{cases} 1 & \text{if } x \in V, \\ 0 & \text{if } x \notin V. \end{cases}$$

The *degree* of the set V is defined as the algebraic degree of its indicator:

$$\deg V := \deg \mathbb{1}_V.$$

In the case of *multiset* over $\mathbb{F}_2^n$, only the elements with an even multiplicity are considered.

A set $V \subseteq \mathbb{F}_2^n$ is said to be a *linear subspace* if V is closed under the addition in $\mathbb{F}_2^n$ (i.e., under the XOR operation). A set $U \subseteq \mathbb{F}_2^n$ is said to be an *affine subspace* if there exists $a \in \mathbb{F}_2^n$ such that $V := a \oplus U := \{a \oplus u \mid u \in U\}$ is a linear subspace. It is then said that $U = a \oplus V$ is a *coset* of the linear subspace V. Such a may not be unique, but the corresponding linear subspace is unique.

Let U be any affine subspace. The *dimension* of U is the maximum number of linearly independent vectors in the linear part of U; it is denoted $\dim U$. Furthermore, U has $2^{\dim U}$ elements. U can be viewed a solution to a system of $k := n - \dim U$ linear equations defined by affine functions $l_1, \ldots, l_k$:

$$U = \{x \in \mathbb{F}_2^n \mid l_1(x) = 0, \ldots, l_k(x) = 0\}.$$

It follows that the indicator of U is affine equivalent to a monomial function of degree $n - \dim U$, i.e. it has the following form:

$$\mathbb{1}_U(x) = (l_1(x) + 1) \cdot \ldots \cdot (l_k(x) + 1).$$

Consider a Boolean function $f \colon \mathbb{F}_2^n \to \mathbb{F}_2, f \neq 0$ and let $d = \deg f$. The minimum possible weight of f is equal to 2^{n-d}, i.e.

$$\mathbf{wt}(f) \geq 2^{n - \deg f}.$$

2.4 Resistance against Linear and Differential Cryptanalysis

Linear and differential cryptanalysis are powerful methods of attacking symmetric cryptographic primitives.

In most block ciphers, S-Boxes are usually the only source of nonlinearity. The resistance of a cipher depends largely on the cryptographic strength of the S-Boxes it uses. Due to typically small sizes of S-Boxes, the linear and differential propagations through them may be analyzed in an exhaustive manner.

For this purpose, the *Linear Approximation Table (LAT)* and the *Difference Distribution Table (DDT)* are used. Even though these objects are motivated by the analysis of S-Boxes, they are also useful theoretical tools in the analysis of larger cryptographic functions.

Definition 2.1 (Walsh Transform)**.** *The* Walsh transform $\mathcal{W}_f$ *of a Boolean function* $f\colon \mathbb{F}_2^n \to \mathbb{F}_2$ *is defined as:*

$$\begin{aligned}&\mathcal{W}_f\colon \mathbb{F}_2^n \to \mathbb{Z},\\ &\mathcal{W}_f(a) := \sum_{x\in\mathbb{F}_2^n} (-1)^{f(x)\oplus\langle a,x\rangle} = -2^n\mathbf{cor}(f\oplus\varphi_a),\end{aligned}$$

where $\varphi_a(x) := \langle a, x\rangle$*. It can be seen as a multidimensional Fourier transform of the function* $x \mapsto (-1)^{f(x)}$*. The multiset of all values of the Walsh transform of* f *is called the* Walsh spectrum *of* f*.*

Definition 2.2 (Linear Approximation Table (LAT))**.** *Let* $S : \mathbb{F}_2^n \to \mathbb{F}_2^m$*. The linear approximation table (LAT) of* S *is the mapping*

$$\begin{aligned}&\mathsf{LAT}_S : \mathbb{F}_2^n \times \mathbb{F}_2^n \to \mathbb{Z},\\ &\mathsf{LAT}_S(a,b) := \mathcal{W}_{S_b}(a) = 2\left|\{x\in\mathbb{F}_2^n \mid \langle a,x\rangle = \langle b, S(x)\rangle\}\right| - 2^n = \sum_{x\in\mathbb{F}_2^n}(-1)^{\langle a,x\rangle\oplus\langle b,S(x)\rangle}.\end{aligned}$$

LAT_S *naturally defines a* $2^n\times 2^n$ *matrix over* $\mathbb{Z}$ *(where the inputs* a, b *are ordered in the lexicographic order). The columns of* LAT_S *correspond to* Walsh transforms *of the components of* S*.*

I remark that in several papers the LAT is defined with a coefficient 1/2 or −1/2, e.g. in [PU16].

Definition 2.3 (Difference Distribution Table (DDT))**.** *Let* $S : \mathbb{F}_2^n \to \mathbb{F}_2^m$*. The difference distribution table (DDT) of* S *is the mapping*

$$\begin{aligned}&\mathsf{DDT}_S : \mathbb{F}_2^n \times \mathbb{F}_2^n \to \mathbb{Z}_{\geq 0},\\ &\mathsf{DDT}_S(a,b) = \left|\{x\in\mathbb{F}_2^n \mid S(x\oplus a)\oplus S(x) = b\}\right|.\end{aligned}$$

DDT_S *naturally defines a* $2^n\times 2^n$ *matrix over* $\mathbb{Z}_{\geq 0}$ *(where the inputs* a, b *are ordered in the lexicographic order).*

The maximum absolute values of the LAT and the DDT of an S-Box are used to measure the cryptographic strength of the S-Box. For this purpose, the *linearity* and the *differential uniformity* of an function are defined.

Definition 2.4 (Linearity)**.** *Let* $f\colon \mathbb{F}_2^n \to \mathbb{F}_2$ *be a Boolean function. The* linearity *of* f *is denoted by* $\mathcal{L}(f)$ *and is defined to be the maximum absolute value in the Walsh spectrum of* f*:*

$$\mathcal{L}(f) := \max_{a\in\mathbb{F}_2^n} |\mathcal{W}_f(a)| = \max_{a\in\mathbb{F}_2^n}\left|\sum_{x\in\mathbb{F}_2^n}(-1)^{f(x)\oplus\langle a,x\rangle}\right|.$$

Let $S : \mathbb{F}_2^n \to \mathbb{F}_2^m$ be a Vectorial Boolean function. The linearity *of S is denoted by $\mathcal{L}(S)$ and is equal to the maximum linearity among the components of S:*

$$\mathcal{L}(S) \coloneqq \max_{b \in \mathbb{F}_2^n, b \neq 0} \mathcal{L}(S_b) = \max_{a \in \mathbb{F}_2^n, b \in \mathbb{F}_2^n, b \neq 0} \left| \sum_{x \in \mathbb{F}_2^n} (-1)^{\langle b, f(x) \rangle \oplus \langle a, x \rangle} \right|.$$

Definition 2.5 (Differential Uniformity)**.** *Let $f : \mathbb{F}_2^n \to \mathbb{F}_2^m$. The differential uniformity of f is denoted by $\delta(f)$ and is given by:*

$$\delta(f) = \max_{a \in \mathbb{F}_2^n, b \in \mathbb{F}_2^m, a \neq 0} \mathsf{DDT}_f(a, b).$$

The entries of the DDT of any S-Box are always even. It follows that the differential uniformity can never be smaller than 2. The functions achieving this lower bound are called *Almost Perfect Nonlinear (APN)*. For example, the cube function over the finite field is always APN [Nyb93]: $x \mapsto x^3, x \in \mathbb{F}_{2^n}$. When n is odd, the cube function is a *permutation* of $\mathbb{F}_{2^n}$ and thus is an *APN permutation*. However, it is not bijective when n is even. The question of existence of APN permutations in even dimensions is a long-standing problem. For $n = 4$ the answer is known to be negative, and for $n = 6$ the positive answer was given by Dillon *et al.* [BDMW10] who explicitly provided a 6-bit APN permutation as a look-up table. In Chapter 5 I describe an interesting decomposition of this function which we found together with my colleagues using S-Box reverse-engineering methods [PUB16]. For even $n \geq 8$ the question is still a big open problem.

Effect of Affine Encodings on the LAT

Compositions of a function with affine mappings have a simple effect on the function's LAT. The following propositions describe the effect separately for addition of constants and composition with linear maps. The constant addition only affects the signs of the LAT coefficients, and the linear encodings shuffle the LAT coefficients in a linear way.

Proposition 2.6. *Let $S \colon \mathbb{F}_2^n \to \mathbb{F}_2^m$ and let $S' \colon \mathbb{F}_2^n \to \mathbb{F}_2^m, S'(x) = S(x \oplus c_x) \oplus c_y$ for some $c_x \in \mathbb{F}_2^n, c_y \in \mathbb{F}_2^m$. Then for any $a \in \mathbb{F}_2^n, b \in \mathbb{F}_2^m$*

$$\mathsf{LAT}_{S'}(a, b) = \mathsf{LAT}_S(a, b)(-1)^{\langle a, c_x \rangle \oplus \langle b, c_y \rangle}.$$

Proposition 2.7. *Let $S \colon \mathbb{F}_2^n \to \mathbb{F}_2^m$ and let $S' \coloneqq B \circ S \circ A$ for some $A \in \mathsf{GL}_n(\mathbb{F}_2), B \in \mathsf{GL}_m(\mathbb{F}_2)$. Then for any $a \in \mathbb{F}_2^n, b \in \mathbb{F}_2^m$*

$$\mathsf{LAT}_{S'}(a, b) = \mathsf{LAT}_S((A^{-1})^\top \times a, B \times b).$$

Part I

Structural and Decomposition Cryptanalysis

In this part, I present the work I have done on structural and decomposition attacks. These attacks aim at determining or distinguishing a particular structure of a cryptographic primitive, which is typically provided as an oracle. Once a structure is established, the cryptanalyst tries to recover its components. For example, the structure may be a Feistel Network with the Feistel round functions being the components. Alternatively, a Substitution-Permutation-Network may be analyzed, where the components are the S-Boxes and the affine mixing layers. There are several reasons for studying structural and decomposition attacks.

First, one may consider a cryptographic primitive, for example, a block cipher, with a publicly known structure but with secret components. For instance, one may replace the components of the AES block cipher - the S-Boxes and the affine layers - by secret ones (see [TKKL15, Gra18]). The description of the secret components thus becomes a part of the key. If the secret components are cryptographically strong, the attacks become harder. In particular, it is harder to attack such primitives in the side-channel setting (though not impossible [RR13]). Furthermore, structural attacks help to understand the security of structures themselves, independently of the specifics of the chosen components. In Chapter 3 I describe distinguishing and decomposition attacks against Feistel Networks. These results are based on the work done together with Léo Perrin [PU16] and partly on the work done together with Léo Perrin and Alex Biryukov [BPU16]. My colleagues also studied the case of Substitution-Permutation-Networks [BKP16].

The second reason comes from the white-box model. The seminal white-box implementations of AES and DES by Chow *et al.* [CEJvO02b, CEJvO02a] are based on the composition of several small components into a single look-up table. Thus, the decomposition attacks pose a direct threat for the security of such implementations. Indeed, multiple decomposition attacks were given [BGEC05, DMWP10, LR13]. Another white-box construction called ASASA was proposed by Biryukov *et al.* [BBK14b]. It is a 2.5-round SPN with secret components. Most ASASA instances were broken in [DDKL15, MDFK15] resulting in a decomposition attack.

The third reason comes from the analysis of S-Boxes. S-Boxes are typically given as lookup tables. Usually, the designers describe the way they generated the S-Box. However, this is not always the case. My colleagues Léo Perrin and Alex Biryukov wrote a seminal work on revealing the secret criteria behind S-Box designs. They called this research direction “S-Box Reverse-Engineering”. In their work, they uncovered possible design criteria of the Skipjack S-Box. They continued developing the S-Box reverse-engineering techniques and found an interesting decomposition of the S-Box used in Russian standard cryptographic primitives. I also contributed to the latter work [BPU16]. Later, we also found another interesting algebraic structure in that S-Box [PU17]. I describe these decompositions in Chapter 4. Furthermore, we found a surprising application of the developed S-Box decomposition techniques. We applied them to find a structure in an S-Box of *mathematical* origin, the 6-bit APN permutation discovered by Dillon *et al.* [BDMW10]. I talk about this result in Chapter 5, which is based on our publication [PUB16].

Chapter 3

Structural Cryptanalysis of Feistel Networks

In this chapter, I describe distinguishing and decomposition attacks against Feistel networks. It is based on a part of the joint work with Alex Biryukov and Léo Perrin [BPU16] and on the joint work with Léo Perrin [PU16]. The first paper describes a structure inside the S-Box used in the Russian cryptographic standards and the method is generalized into a decomposition attack against 3- and 4- round Feistel Networks composed with random affine mappings. The second paper analyzes the method more deeply. It gives many insights into algebraic degeneracies in Feistel Networks and describes how to exploit such artifacts to mount distinguishing and decomposition attacks. This analysis and the generalization is described in this chapter, and the GOST S-Box decomposition is described in Chapter 4.

3.1 Introduction

A Feistel Network (FN) together with Substitution-Permutation Network (SPN) are the two main structures used to design a block cipher. Both are iterated structures in which a simple round function is iterated multiple times. The Feistel Network was invented by Horst Feistel who designed the Lucifer [Sor84] block cipher at IBM. Lucifer was a direct predecessor of the Data Encryption Standard (DES) [Cop94] block cipher which has a 16-round Feistel Network as its structure.

A classical Feistel Network operates on two n-bit branches of the same size. The round function works in the following way. A so-called Feistel function is applied to the right branch and the result is added to the left branch using the XOR or the modular addition. Afterward, the branches are swapped. The swap in the last round is usually omitted. A 3-round Feistel Network is shown in Figure 3.1. The Feistel function is not required to be bijective. In fact, a Feistel Network can be seen as a way to construct a pseudorandom *permutation* from several pseudorandom *functions*. In 1988, Luby and Rackoff [LR88] proved adaptive chosen-plaintext security of a 3-round Feistel Network under the assumption of (pseudo)random Feistel functions. The proof states that any adversary making q queries to the primitive cannot distinguish it from a random permutation with a probability higher than $q^2/2^n$. It follows that the security is guaranteed only as long as q is much smaller than the birthday bound $2^{n/2}$.

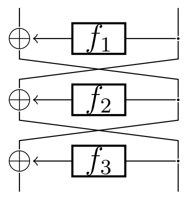


FIGURE 3.1: A 3-round Feistel Network.

A block cipher must have a relatively large block size, $2n \geq 64$. In this case, it is impractical to generate fully random n-bit Feistel functions and store them during encryption. Usually, the Feistel function is chosen to have a simple and efficient structure and is public. However, a secret round key is injected before application of the Feistel function. This construction is called a Key-Alternating Feistel (KAF) cipher. It is much weaker than the ideal one and requires much more rounds in order to achieve strong security. For example, DES has 16 rounds and the recent block cipher Simon [BSS+13] by the NSA has at least 32 rounds in its variants. An analysis of KAF ciphers was done by Lampe *et al.* [LS14], Dinur *et al.* [DDKS15] and more recently by Guo *et al.* [GW18].

From the viewpoint of structural cryptanalysis, it is still important to analyze Feistel Networks with secret round functions. This may have applications in white-box cryptography or S-Box reverse-engineering. Patarin [Pat01,Pat04] first described attacks on generic 5-round Feistel Network. In the seminal S-Box reverse-engineering paper [BP15], Biryukov and Perrin proposed a SAT-solver based heuristic algorithm which seems to be practical for branch sizes of up 7 bits and up to 7 rounds. Biryukov *et al.* [BLP15] described several cryptanalysis methods against generic Feistel Networks with up to 7 rounds, including integral and Yoyo cryptanalysis. More recently, Durak *et al.* [DV18] described decomposition attacks against Feistel Networks with small branch domains based on optimized exhaustive search and the Meet-in-the-Middle technique.

Often the Feistel function has a low algebraic degree for the efficiency reasons. For example, many FN-based ciphers (DES, Camellia) use one SPN round as a Feistel function. The degree of the Feistel function is then upper-bounded by the S-Box size minus one. The same degree bound applies for the inverses of such Feistel Functions. Todo [Tod15] proposed a novel method for finding integral characteristics in general structures, called division property. He evaluated FNs and SPNs based on a degree bound of components. Léo Perrin and I analyzed the algebraic degree of Feistel Networks in [PU16]. In addition, we showed how to cryptanalyze a Feistel Network composed with random affine encodings. Affine encodings are motivated by S-Box reverse-engineering and white-box applications, where such encodings can provide extra security at a low cost for the designer. These results form the plot of this chapter.

3.1.1 Notation

In this chapter, I will use the following definition of a Feistel Network. It includes a bound on the algebraic degree of the Feistel functions as a parameter since proposed attacks exploit low degree or algebraic degeneracy.

Definition 3.1 (Feistel Network). P^r_d *(resp.* F^r_d*) denotes the set of all permutations that can be expressed as an r-round Feistel Network with bijective (resp. unrestricted) Feistel functions $f_1, \ldots, f_r \colon \mathbb{F}_2^n \to \mathbb{F}_2^n$ of an algebraic degree at most d:*

$$\mathsf{P}^r_d \coloneqq \{\mathsf{swap} \circ R_{f_n} \circ \ldots \circ R_{f_1} \mid f_i \colon \mathbb{F}_2^n \to \mathbb{F}_2^n \text{ bijective}\},$$
$$\mathsf{F}^r_d \coloneqq \{\mathsf{swap} \circ R_{f_n} \circ \ldots \circ R_{f_1} \mid f_i \colon \mathbb{F}_2^n \to \mathbb{F}_2^n\},$$
where
$$R_f \colon (\mathbb{F}_2^n)^2 \to (\mathbb{F}_2^n)^2, \quad (a, b) \mapsto (b, a \oplus f(b)).$$

In a few cases, the algebraic degree of the *inverse* of the Feistel function is considered. The upper bound is denoted by d_{-1}.

3.1.2 Contribution

Our work [PU16] has several contributions and I believe that it enriches the toolkit of structural cryptanalysis. I distinguish the following parts:

1. We show an interesting link between the integral cryptanalysis and the LAT modulo 8. This fact does not seem to have direct applications but is interesting from a theoretical viewpoint. It might be useful for locating visual patterns in the LAT for the purpose of S-Box reverse-engineering.

2. We define the High-Degree Indicator Matrix of a vectorial Boolean function. While it simply captures classic integral distinguishers, it has many useful properties and provides more insights into integral cryptanalysis.

3. We study algebraic degree growth in Feistel Networks. As a result, we provide simple closed formulas that give rather good degree upper bounds. Though the algorithmic approach using the division property by Todo [Tod15] provides similar or slightly better results.

4. We propose decomposition attacks on Feistel Networks masked with affine layers. Previously, a similar attack was only described for unmasked 5-round Feistel Networks in [BLP15]. We generalize it for more rounds based on the algebraic degeneracies proved in this work.

The summary of structural attacks against Feistel Networks is given in Table 3.1, including attacks against Feistel Networks whitened with affine encodings.

3.1.3 Outline

This chapter starts with the description of visual patterns in the LAT of random instances of 3- and 4-round Feistel Networks in Section 3.2. These patterns then are explained and linked to the algebraic degeneracies in these structures. The relevant algebraic structure is encoded in a new object called High-Degree Indicator Matrix. In Section 3.3 the algebraic degeneracies are proved and generalized to a larger number of rounds depending on the algebraic degree of Feistel functions. This immediately yields integral distinguishers. In the following Section 3.4 these attacks are extended to Feistel Networks composed with secret affine encodings. Further, I show lower degree algebraic degeneracies in Feistel Networks in Section 3.5. In Section 3.6 I describe how to exploit such weaknesses to mount a round function recovery attack. I discuss the results and conclude in Section 3.7.

3.1.4 Differences with [PU16]

This chapter is a rather significantly reworked version of the paper [PU16] that we wrote together with Léo Perrin. Here I briefly describe the most significant modifications and additions that I have done in this chapter.

Struc.	**Method**	**Power**	**Restrictions**	**Time**	**Data**	**Ref.**
F_d^5	differential	distin.	f_i non-bij.	2^n	2^n	[Pat01]
	SAT-based	recov.	$n \leq 7$	practical	2^{2n}	[BP15]
	yoyo	recov.	–	2^{2n}	2^{2n}	[BLP15]
	guess & det.	recov.	–	$2^{n2^{3n/4}}$	2^{2n}	[BLP15]
P_d^5	imp. diff.	distin.	–	2^{2n}	2^n	[Knu98]
	div. prop.	distin.	–	2^{2n-1}	2^{2n-1}	[Tod15]
	HDIM	distin.	–	2^{2n-1}	2^{2n-1}	Sec. 3.3
	integral	recov.	–	$2^{2.8n}$	2^{2n}	[BLP15]
	imp. monom.	recov.	–	2^{3n}	2^{2n}	Sec. 3.5
F_d^r	HDIM	distin.	$\Lambda_{\mathsf{F}}(r, d) < 2n$	2^{2n-1}	2^{2n-1}	Sec. 3.3
	div. prop.	distin.	algorithmic	2^{2n-1}	2^{2n-1}	[Tod15]
	imp. monom.	recov.	Conj. 3.33	2^{3n}	2^{2n}	Sec. 3.5
P_d^r	HDIM	distin.	$\Lambda_{\mathsf{F}}(r-1, d) < 2n$, $d_{-1} \leq d$	2^{2n-1}	2^{2n-1}	Sec. 3.3
	div. prop.	distin.	algorithmic	2^{2n-1}	2^{2n-1}	[Tod15]
	imp. monom.	recov.	Conj. 3.33	2^{3n}	2^{2n}	Sec. 3.5
$\mathsf{AP}_d^4\mathsf{A}'$	LAT-based	recov.	–	2^{6n}	2^{2n}	[BPU16]
$\mathsf{AF}_d^r\mathsf{A}'$	HDIM	recov.	$\Lambda_{\mathsf{F}}(r+1, d) < 2n$	$n2^{2n}$	2^{2n}	Sec. 3.4
$\mathsf{A}^{-1}\mathsf{F}_d^r\mathsf{A}$	HDIM	recov.	$\Lambda_{\mathsf{F}}(r, d) < 2n$	$n2^{2n}*$	2^{2n}	Sec. 3.4
AF_d^r	HDIM	recov.	$\Lambda_{\mathsf{F}}(r, d) < 2n$	$n2^{2n}$	2^{2n}	Sec. 3.4
$\mathsf{AP}_d^r\mathsf{A}'$	HDIM	recov.	$\Lambda_{\mathsf{F}}(r, d) < 2n$, $d_{-1} \leq d$	$n2^{2n}$	2^{2n}	Sec. 3.4
$\mathsf{A}^{-1}\mathsf{P}_d^r\mathsf{A}$	HDIM	recov.	$\Lambda_{\mathsf{F}}(r-1, d) < 2n$, $d_{-1} \leq d$	$n2^{2n}*$	2^{2n}	Sec. 3.4
AP_d^r	HDIM	recov.	$\Lambda_{\mathsf{F}}(r-1, d) < 2n$, $d_{-1} \leq d$	$n2^{2n}$	2^{2n}	Sec. 3.4

TABLE 3.1: Structural attacks against Feistel Networks with $r \geq 5$ rounds. n is the branch size, d is the degree bound of the Feistel functions, d_{-1} is the degree bound of their inverses. It is known that $\Lambda_{\mathsf{P}}(r, d) \leq \Lambda_{\mathsf{F}}(r, d) \leq d^{\lfloor r/2 \rfloor - 1} + d^{\lceil r/2 \rceil - 1}$. A, A' are secret affine transformations. HDIM, division property, impossible monomial attack are all integral-based attacks. Attacks from this chapter were published in [PU16]. $*$ assuming complexity of solving a system of n^2 quadratic equations is negligible.

1. I redefined and generalized the parametrization of the conditions on the integral distinguishers. I distinguish the case of bijective Feistel functions in a stricter way. Further, instead of using the parameter $\theta(r, d) = d^{\lfloor r/2 \rfloor - 1} + d^{\lceil r/2 \rceil - 1}$ from the original paper which comes from a very basic degree evaluation method, I use the parameters $\Lambda_{\mathsf{P}}(r, d), \Lambda_{\mathsf{F}}(r, d)$ which correspond to exact degree bounds and are hard to evaluate but can be upper bounded using various methods. In this way, improving the upper bounds on the degrees would directly improve the results. In addition, I consider the effect of the degree of the inverse of the Feistel functions.

2. I describe a generalization of the LAT-ANF link to congruences with larger powers of 2. It is a simple corollary from the Poisson Summation formula, which I think is not very well-known or used often. It provides a clear relation between the ANF and the LAT of a Boolean function and gives an insight into the structure of the Walsh transform.

3. I describe a generalization of the HDIM-ANF link, an alternative expression for an arbitrary ANF coefficient. The HDIM expression yields a new method of proving the absence of particular monomials of degree $n - 1$ in a permutation; the generalization yields analogous method for arbitrary monomials.

4. I provide a more rigorous and explicit analysis of the attacks on Feistel Networks masked with affine encodings. I distinguish the cases of linear and quadratic equation systems and analyze the conditions of success. Furthermore, I describe the re-randomization trick which allows attacking arbitrary affine encodings.

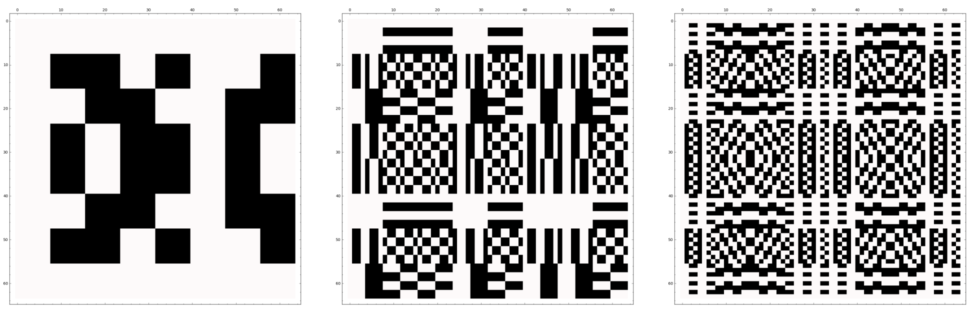
5. I describe the impossible monomial attack in a more concise and accurate way. Furthermore, I provide an algorithm in pseudocode. In addition, I propose a conjecture about the instances of Feistel Networks that can be attacked. I perform an experimental evaluation of the attack and the conjecture.

3.2 High-Degree Indicator Matrix

In [BP15] Perrin and Biryukov suggested looking at the visual representation of the LAT of an S-Box with the goal of finding non-random patterns. The suggested representation is a heatmap of the LAT matrix and was named "a Jackson Pollock representation" of the LAT, after the famous abstract expressionist painter. The success of this method is illustrated in this chapter.

Consider a 4- and 5-round Feistel Network. For a tiny branch size (for example, 3 bits) it is possible to generate the whole codebook and compute the LAT and its visualization. Figure 3.2 shows the Pollock representations of the LAT of Feistel Networks with randomly generated bijective round functions with 3-bit branches (6-bit block size), taken modulo 8.

The images yield a lot of patterns. The patterns are even more clear when observed on multiple random instances of the Feistel Network. In particular, the



(A) $r = 4$ (B) $r = 5$ (C) $r = 6$

FIGURE 3.2: LAT of r-round Feistel Networks (modulo 8). White and black correspond to 0 and 4.

4-round structure always yields LAT consisting of 8×8 single-colored squares. The 5-round structure still has a visible square structure, but not all squares are single-colored. The topmost leftmost square is always white, the topmost (resp. leftmost) squares consist of horizontal (resp. vertical) lines. Furthermore, linear patterns can be noticed: many columns/rows are inverted versions of other columns/rows, and many columns/rows can be expressed as sums of other columns/rows (modulo 8). The 6-round structure still has linear patterns but no clear squared structure.

After studying these patterns, we observed that the LAT modulo 8 is a bilinear form directly related to the monomials of degree $n-1$ in the ANF of the analyzed n-bit permutation. This is formally stated and proved in the following section.

3.2.1 Relation between HDIM, LAT and ANF

Definition 3.2 (HDIM). *Let $S\colon \mathbb{F}_2^n \to \mathbb{F}_2^m$ and $\deg S \le n-1$. The HDIM of S is the $m \times n$ matrix HDIM_S over $\mathbb{F}_2$ given by*

$$\mathsf{HDIM}_S[i,j] \coloneqq \bigoplus_{x \in \mathbb{F}_2^n} \langle e_i, S(x)\rangle \langle e_j, x\rangle \,.$$

Proposition 3.3 (HDIM and ANF). $\mathsf{HDIM}_S[i,j] = 1$ *if and only if the ANF of the i-th coordinate of S contains the monomial $\prod_{k \neq j} x_k = x_1 \ldots x_n / x_j$.*

Proof. The sum over $\mathbb{F}_2^n$ is equal to 1 if and only if the summed expression contains the monomial $x_1 \ldots x_n$. Since no coordinate of S has term of degree n, $\mathsf{HDIM}_S[i,j] = 1$ is equivalent to $\langle e_i, S(x)\rangle$ having the monomial $x_1 \ldots x_n / x_j$. □

Proposition 3.3 shows that a known value of a cell of the HDIM corresponds to a known ANF coefficient, i.e. an integral distinguisher. The following theorem describes the relation between the HDIM and the LAT of a function.

Theorem 3.4 (HDIM and LAT). *Let $S\colon \mathbb{F}_2^n \to \mathbb{F}_2^m, n \ge 3$ be a balanced function. Then*

$$(\mathsf{LAT}_S(a,b) \mod 8)/4 = b^\top \times \mathsf{HDIM}_S \times a.$$

Proof. By linearity of the inner product,

$$b^\top \times \mathsf{HDIM}_S \times a = \bigoplus_{x \in \mathbb{F}_2^n} \langle b, S(x) \rangle \langle a, x \rangle .$$

On the other hand, using $(-1)^z = 1 - 2z$ for $z \in \mathbb{F}_2$ we obtain:

$$\mathsf{LAT}_S(a,b) = \sum_{x \in \mathbb{F}_2^n} (-1)^{\langle a,x \rangle \oplus \langle b, S(x) \rangle} = \sum_{x \in \mathbb{F}_2^n} (1 - 2\langle a, x \rangle)(1 - 2\langle b, S(x) \rangle).$$

Observe that

$$\sum_{x \in \mathbb{F}_2^n} \langle a, x \rangle = 2^{n-1} = \sum_{x \in \mathbb{F}_2^n} \langle b, S(x) \rangle ,$$

where the last equality holds because S is balanced. It follows that

$$\mathsf{LAT}_S(a,b) = 4 \sum_{x \in \mathbb{F}_2^n} \langle a, x \rangle \langle b, S(x) \rangle - 2^n$$

and, for $n \geq 3$, $\mathsf{LAT}_S(a,b) \equiv 4 \sum_{x \in \mathbb{F}_2^n} \langle a, x \rangle \langle b, S(x) \rangle \pmod 8$. □

The HDIM serves as an interesting link between the algebraic normal form and the linear approximation table of a function. It captures all the information in the LAT modulo 8 and explains the (bi)linear patterns. However, the square patterns in Figure 3.2 are artifacts of the 3- and 4-round Feistel Network structure. These patterns have a simple expression in terms of the HDIM. They will be formalized and proved in Section 3.3.

3.2.2 Properties of the HDIM

The HDIM inherits some properties from the LAT. In particular, taking the inverse of a permutation or composing a function with affine mappings has a simple effect on the HDIM.

Proposition 3.5. *Let S be a permutation of $\mathbb{F}_2^n$. Then*

$$\mathsf{HDIM}_{S^{-1}} = \mathsf{HDIM}_S^\top .$$

Proof. It follows from the fact that $\mathsf{LAT}_{S^{-1}} = \mathsf{LAT}_S^\top$. □

Proposition 3.6. *Let $S \colon \mathbb{F}_2^n \to \mathbb{F}_2^m$, and let μ and η be linear permutations of $\mathbb{F}_2^n$ and $\mathbb{F}_2^m$ respectively. Let $T = \eta \circ S \circ \mu$. Then*

$$\mathsf{HDIM}_T = \eta \times \mathsf{HDIM}_S \times (\mu^\top)^{-1}.$$

Proof.

$$\begin{aligned}\mathsf{HDIM}_T[i,j] &= e_i^\top \times \mathsf{HDIM}_T \times e_j = \\ &= \bigoplus_{x\in\mathbb{F}_2^n} \langle e_i, \eta(S(\mu(x)))\rangle \langle e_j, x\rangle = \\ &= \bigoplus_{z\in\mathbb{F}_2^n} \langle e_i, \eta(S(z))\rangle \left\langle e_j, \mu^{-1}\times z\right\rangle = \\ &= \bigoplus_{z\in\mathbb{F}_2^n} \left\langle \eta^\top \times e_i, S(z)\right\rangle \left\langle (\mu^\top)^{-1}\times e_j, z\right\rangle = \\ &= (e_i^\top \times \eta)\times \mathsf{HDIM}_S \times ((\mu^\top)^{-1}\times e_j).\end{aligned}$$

The proposition follows. □

3.2.3 Generalization of the LAT-ANF link

For the rest of the chapter, the link between the HDIM and the LAT will not be used. However, I would like to note a generalization of this link for congruences of the LAT modulo higher powers of 2, for example, modulo 16, 32, etc. The link connects *sums* of the Walsh transform over subspaces with ANF coefficients of lower degree. It is based on the following theorem that relates the sum of a Boolean function f over a linear subspace $V \subseteq \mathbb{F}_2^n$ with the sum of the Walsh transform of f over the orthogonal complement of V. The first quantity is directly related to the ANF of f.

Theorem 3.7 (Poisson Summation, [Lec71, p.147])**.** *Let $f\colon \mathbb{F}_2^n \to \mathbb{F}_2$ and let $V \subseteq \mathbb{F}_2^n$ be a linear subspace. Then*

$$\sum_{a\in V}\mathcal{W}_f(a) = 2^n - 2^{\dim V+1}\sum_{x\in V^\perp} f(x).$$

Proof. Observe that

$$\sum_{a\in V}\mathcal{W}_f(a) = \sum_{a\in V}\sum_{x\in\mathbb{F}_2^n}(-1)^{\langle a,x\rangle\oplus f(x)} = \sum_{x\in\mathbb{F}_2^n}(-1)^{f(x)}\sum_{a\in V}(-1)^{\langle a,x\rangle}.$$

If $x\in V^\perp$, then $\sum_{a\in V}(-1)^{\langle a,x\rangle} = |V|$. Otherwise, $\langle a,x\rangle = 1$ exactly for half of V and therefore, $\sum_{a\in V}(-1)^{\langle a,x\rangle} = 0$. It follows that

$$\sum_{a\in V}\mathcal{W}_f(a) = |V|\sum_{x\in V^\perp}(-1)^{f(x)} = |V|\left(|V^\perp| - 2\sum_{x\in V^\perp} f(x)\right) = 2^n - 2^{\dim V+1}\sum_{x\in V^\perp} f(x).$$

□

Corollary 3.8. *Let $f\colon \mathbb{F}_2^n \to \mathbb{F}_2$ be balanced. For any linear subspace $V \subseteq \mathbb{F}_2^n$*

$$2^n - \sum_{a\in V}\mathcal{W}_f(a) \equiv 2^{\dim V+1}\left(\bigoplus_{x\in\mathbb{F}_2^{V^\perp}} f(x)\right) \pmod{2^{\dim V+2}}.$$

In particular, the link between the LAT and the ANF established using HDIM follows for $n \geq 3$:

$$(\mathcal{W}_f(e_j) \mod 8)/4 = \bigoplus_{x \in \mathbb{F}_2^n, \langle e_j, x \rangle = 0} f(x),$$

where the last expression is the ANF coefficient of the monomial $x_1 \ldots x_n / x_j$.

Example 1. Consider the monomial $x^u = x_3 x_4 \ldots x_n$ of degree $n-2$. The corresponding coefficient a_u in the ANF of f can be expressed as (for $n \geq 4$):

$$a_u = \frac{1}{8}\big((\mathcal{W}_f(e_1) + \mathcal{W}_f(e_2) + \mathcal{W}_f(e_1 + e_2)) \mod 16\big).$$

3.2.4 Generalization of the HDIM-ANF link

The HDIM-ANF link provides an expression for a coefficient of the monomial $x_1 \ldots x_n / x_j$ in the ANF of a balanced Boolean function $f \colon \mathbb{F}_2^n \to \mathbb{F}_2$,

$$a_u = \bigoplus_{x \in \mathbb{F}_2^n} f(x) \cdot \langle e_j, x \rangle,$$

where $1 \leq j \leq n$ and $u \in \mathbb{F}_2^n$ is such that $u_i = 1$ if and only if $i \neq j$. This idea can be generalized for monomials of lower degrees:

Proposition 3.9. *Let $f \colon \mathbb{F}_2^n \to \mathbb{F}_2$ and $u \in \mathbb{F}_2^n$. Then the coefficient a_u of the monomial $x^u := x_1^{u_1} \ldots x_n^{u_n}$ can be computed as:*

$$a_u = \bigoplus_{x \in \mathbb{F}_2^n} f(x) \cdot (\neg x)^{\neg u},$$

where $(\neg x)^{\neg u} := (x_1 \oplus 1)^{u_1 \oplus 1} \ldots (x_n \oplus 1)^{u_n \oplus 1}$.

Proof. The term $(\neg x)^{\neg u}$ is equal to one if and only if $x_i = 0$ for all i such that $u_i = 0$, or, equivalently, $x \preceq u$. It follows that the equation from the proposition is equivalent to

$$a_u = \bigoplus_{x \preceq u} f(x),$$

which is exactly the expression of the ANF coefficient a_u obtained from the Möbius inversion formula. $\square$

Remark 1. The Boolean function f can be replaced by a coordinate $\langle e_i, S \rangle$ of vectorial Boolean function S.

Remark 2. The HDIM expression involves $\langle x, e_i \rangle = x_i^e = x^{\neg u}$. Since f is balanced (i.e. XORs to zero), it is indeed equivalent to $(\neg x)^{\neg u}$. For degrees lower than $n-1$, this is not true in general.

The generalization of the HDIM-ANF link can be used to directly prove a useful general composition bound by Boura and Canteaut [BC13, Corollary 2].

Proposition 3.10. *Let F be a permutation of $\mathbb{F}_2^n$ and let $g\colon \mathbb{F}_2^n \to \mathbb{F}_2$. Then*

$$\deg g \circ F \leq n - \left\lceil \frac{n - \deg g}{\deg F^{-1}} \right\rceil.$$

Proof. By Proposition 3.9, the coefficient a_u of the monomial x^u in the ANF of $g \circ F$ can be computed as

$$a_u = \bigoplus_{x \in \mathbb{F}_2^n} g(F(x)) \cdot (\neg x)^{\neg u} = \bigoplus_{z \in \mathbb{F}_2^n} g(z) \cdot (\neg F^{-1}(z))^{\neg u}.$$

If follows that $a_u = 0$ if

$$\deg \left(g(z) \cdot (\neg F^{-1}(z))^{\neg u} \right) < n,$$

which is definitely true if

$$\deg g + (n - \mathbf{wt}(u)) \cdot \deg F^{-1} < n.$$

Equivalently, $a_u = 0$ if

$$\mathbf{wt}(u) > n - \frac{n - \deg g}{\deg F^{-1}}.$$

It follows that

$$\deg g \circ F \leq n - \frac{n - \deg g}{\deg F^{-1}} \leq n - \left\lceil \frac{n - \deg g}{\deg F^{-1}} \right\rceil.$$

I remark that strict inequality from Corollary 2 from [BC13] is equivalent to this inequality by switching the rounding up to the rounding down. □

3.3 HDIM of Feistel Networks

The Feistel Network is a rather asymmetric and imbalanced structure. After any round, the left branch is a result of more computations, and the right branch is "weaker" in this sense. Often it may occur that, after a particular number of rounds, the left branch has full algebraic degree, while the right branch is still of incomplete degree. This can be seen as the maximum number of rounds available for an integral distinguisher, since after the next round the strong left branch is mixed into the weak right branch, and, in general, we can expect both branches to have full degree. To exploit the imbalance of Feistel Networks, we analyze the degree of the "weak" right branch.

Definition 3.11. *Let $\lambda_{\mathsf{F}}(r, d)$ denote the maximum possible degree of the right output branch of a Feistel Network with r rounds and Feistel functions of degree at most d. Let $\lambda_{\mathsf{P}}(r, d)$ denote the maximum possible degree in the case when* *the Feistel functions are bijective:*

$$\lambda_{\mathsf{F}}(r,d) = \max_{F \in \mathsf{F}_d^r} \deg\left(\mathsf{right} \circ F\right),$$
$$\lambda_{\mathsf{P}}(r,d) = \max_{F \in \mathsf{P}_d^r} \deg\left(\mathsf{right} \circ F\right).$$

Remark 3. The maximum degree of the left branch is equal to the maximum degree of the right branch in the next round, since the branch is transferred untouched.

Remark 4. The exact values of $\lambda_{\mathsf{F}}(r,d), \lambda_{\mathsf{P}}(r,d)$ are hard to compute. However, upper bounds can be computed using different methods. Improving further the upper bounds should lead to strengthening the results from this chapter.

The definition of HDIM leads to an interesting insight into proving absence of particular monomials of maximum degree (i.e. $n-1$ for n-bit permutations), or, equivalently, zeroes in the HDIM itself. The idea is to split the computed function into two roughly equal parts. Then the algebraic/integral distinguisher exists when the *sum* of the degrees of the two parts is less than the block size n. In the original case, the parts are composed and the degrees are roughly *multiplied*, i.e. the distinguisher is found when the *product* of the degrees is less than $n-1$.

In this section the utility functions $\Lambda_{\mathsf{F}}(r,d)$ and $\Lambda_{\mathsf{P}}(r,d)$ will be used to describe the conditions when the integral distinguisher exists.

Definition 3.12. *The functions $\Lambda_{\mathsf{F}}(r,d), \Lambda_{\mathsf{P}}(r,d)$ are defined as follows:*

$$\Lambda_{\mathsf{F}}(r,d) = \lambda_{\mathsf{F}}(\lfloor r/2 \rfloor, d) + \lambda_{\mathsf{F}}(\lceil r/2 \rceil, d),$$
$$\Lambda_{\mathsf{P}}(r,d) = \lambda_{\mathsf{P}}(\lfloor r/2 \rfloor, d) + \lambda_{\mathsf{P}}(\lceil r/2 \rceil, d).$$

The following lemma shows a simple upper bound from the product bound of a composition.

Lemma 3.13. *A Feistel Network with $r \geq 1$ rounds and degree-d round functions has degree at most d^r on the left output branch and degree at most d^{r-1} on the right output branch:*

$$\lambda_{\mathsf{P}}(r,d) \leq \lambda_{\mathsf{F}}(r,d) \leq d^{r-1}.$$

In particular,

$$\Lambda_{\mathsf{P}}(r,d) \leq \Lambda_{\mathsf{F}}(r,d) \leq d^{\lfloor r/2 \rfloor - 1} + d^{\lceil r/2 \rceil - 1}.$$

Remark 5. In the case of Feistel Networks, the block size is assumed to be $2n$, whereas in discussions about general permutations, the block size is n.

The HDIM-based distinguishers that we exhibit in this section have the same structure: if conditions of a distinguisher are satisfied, then the $2n \times 2n$ HDIM has the form $\begin{bmatrix} ? & 0 \\ 0 & 0 \end{bmatrix}$ as a 2×2 block-matrix. Such distinguisher is automatically extended to one more round leading to an HDIM of the form $\begin{bmatrix} ? & ? \\ ? & 0 \end{bmatrix}$. This is formalized in the following definition and a lemma.

Definition 3.14 (Type-I, Type-II Distinguishers). *Let $S\colon \mathbb{F}_2^{2n} \to \mathbb{F}_2^{2n}$. Then*

- *S is said to have the type-I distinguisher if*

$$\mathsf{HDIM}_S[i,j] = 0 \text{ for } n < i \le 2n \textbf{ or } n < j \le 2n;$$

- *S is said to have the type-II distinguisher if*

$$\mathsf{HDIM}_S[i,j] = 0 \text{ for } n < i \le 2n \textbf{ and } n < j \le 2n.$$

Lemma 3.15. *Let $r \ge 1$, $S_r \in \mathsf{F}_d^r$ and $S_{r+1} \in \mathsf{F}_d^{r+1}$ be $2n$-bit permutations such that*

$$S_{r+1} = \mathsf{swap} \circ R_f \circ \mathsf{swap} \circ S_r$$

for some function $f\colon \mathbb{F}_2^n \to \mathbb{F}_2^n$.

If S_r has the type-I distinguisher, then S_{r+1} has the type-II distinguisher.

Proof. Since, $\mathsf{right} \circ S_{r+1} = \mathsf{left} \circ S_r$ the last n rows of $\mathsf{HDIM}_{S_{r+1}}$ are the same as the first n rows of HDIM_{S_r}. □

3.3.1 General Case

The following theorem applies the described ideas to general Feistel Networks.

Theorem 3.16. *Any $S \in \mathsf{F}_d^r$ has the type-I distinguisher if*

$$\Lambda_{\mathsf{F}}(r+1, d) < 2n.$$

Similarly, any $S \in \mathsf{P}_d^r$ has the type-I distinguisher if

$$\Lambda_{\mathsf{P}}(r+1, d) < 2n.$$

Proof. Let $(a,b) \in \mathbb{F}_2^{2n}$ denote the intermediate state of S after $\lfloor r/2 \rfloor$ rounds (see Figure 3.3). Let $x_l(a,b), x_r(a,b)$ denote the input branches of S as functions of a,b, and $y_l(a,b), y_r(a,b)$ the output branches of S as functions of a,b. We now perform the variable replacement in the definition of HDIM:

$$\mathsf{HDIM}_S[i,j] = \bigoplus_{x \in \mathbb{F}_2^{2n}} \langle e_i, S(x) \rangle \langle e_j, x \rangle = \bigoplus_{(a,b) \in \mathbb{F}_2^{2n}} \langle e_i, (x_l, x_r)(a,b) \rangle \langle e_j, (y_l, y_r)(a,b) \rangle .$$

Our goal is to prove a bound on the algebraic degree of the product of the two inner products.

Variables x_l, x_r, y_l, y_r can be computed using a Feistel Network with a,b as inputs: $(x_l, x_r) \circ \mathsf{swap} \in \mathsf{F}_d^{\lfloor r/2 \rfloor}$ and $(y_l, y_r) \in \mathsf{F}_d^{\lceil r/2 \rceil}$. Therefore, they have the following degree bounds:

- $\deg x_l \le \lambda_{\mathsf{F}}(\lfloor r/2 \rfloor + 1, d)$, $\deg x_r \le \lambda_{\mathsf{F}}(\lfloor r/2 \rfloor, d)$,
- $\deg y_l \le \lambda_{\mathsf{F}}(\lceil r/2 \rceil + 1, d)$, $\deg y_r \le \lambda_{\mathsf{F}}(\lceil r/2 \rceil, d)$.

The zeroes in the HDIM required for the type-I distinguisher correspond to products $x_r \cdot y_r$, $x_r \cdot y_l$ and $x_l \cdot y_r$. It is enough to prove the case $n < i \leq 2n$, since the inverse of S is also a Feistel Network and thus the transpose of the HDIM_S will have the same zeroes. The case corresponds to the products $x_r \cdot y_r$ and $x_l \cdot y_r$. It follows that $\mathsf{HDIM}_S[i,j] = 0$ if $n < j \leq 2n$ and $\lambda_{\mathsf{F}}(\lfloor r/2 \rfloor + 1, d) + \lambda_{\mathsf{F}}(\lceil r/2 \rceil, d) < 2n$. The condition is equivalent to $\Lambda_{\mathsf{F}}(r+1, d) < 2n$. □

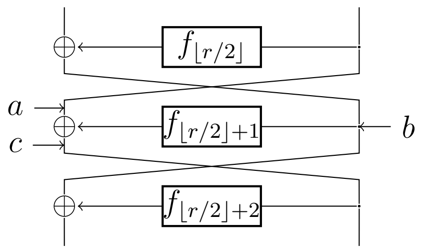

$f_{\lfloor r/2 \rfloor}$
a
$f_{\lfloor r/2 \rfloor+1}$
b
c
$f_{\lfloor r/2 \rfloor+2}$

FIGURE 3.3: The variables a, b and c.

Corollary 3.17. *Any $S \in \mathsf{F}_d^r$ has the type-I distinguisher if*

$$d^{\lfloor r/2 \rfloor} + d^{\lceil r/2 \rceil - 1} < 2n,$$

and the type-II distinguisher if

$$d^{\lfloor r/2 \rfloor - 1} + d^{\lceil r/2 \rceil - 1} < 2n.$$

Proof. Putting the bound from Lemma 3.13 in Theorem 3.16 makes the proof. For the type-II distinguisher the result follows from Lemma 3.15. □

3.3.2 Bijective Feistel Functions

In the case when the Feistel functions are bijective, an additional trick may be used. The intermediate state variables can be chosen in an alternative way by exploiting the fact that the middle Feistel function is invertible. However, in this case we need to know an upper bound on the degree of the *inverse* of the middle Feistel function.

In what follows, the upper bound on the algebraic degree of the inverse of the Feistel function is denoted by d_{-1}.

Theorem 3.18. *Any $S \in \mathsf{P}_d^r$ has the type-I distinguisher if*

$$\max(d, d_{-1}) \cdot \Lambda_{\mathsf{P}}(r-2, d) < 2n.$$

Proof. The proof is analogous to the proof of Theorem 3.16, except that the choice of intermediate variables differs. Instead of choosing both left and right branches of the input of the middle round, it is possible to choose the left branch of the input and the right branch of the output. The variables chosen are (a, c) instead of (a, b) (see Figure 3.3). In this case b can be expressed as $f^{-1}_{\lfloor r/2 \rfloor + 1}(a \oplus c)$, and degree of b as a function of (a, c) is upper bounded by d_{-1}.

Without loss of generality, assume $r \geq 3$. Let $(a_x, b_x) \in \mathbb{F}_2^{2n}$ denote the state before the $\lfloor r/2 \rfloor$-th round, and $)a_y, b_y) \in \mathbb{F}_2^{2n}$ denote the state after the $(\lfloor r/2 \rfloor + 2)$-th round. The following degree bounds hold (every variable is considered as a function of (a, c)):

- $\deg a_x \leq \max(d, d_{-1}), \deg b_x = 1,$
- $\deg a_y \leq \max(d, d_{-1}), \deg b_y = 1,$

The input (x_l, x_r) of S can be computed as a $(\lfloor r/2 \rfloor - 1)$-round Feistel Network composed with the function (a_x, b_x). Similarly, the output (y_l, y_r) of S can be computed as a $(\lceil r/2 \rceil - 2)$-round Feistel Network composed with the function (a_y, b_y). It follows that

- $\deg x_l \leq \max(d, d_{-1}) \cdot \lambda_{\mathsf{P}}(\lfloor r/2 \rfloor, d),$
- $\deg x_r \leq \max(d, d_{-1}) \cdot \lambda_{\mathsf{P}}(\lfloor r/2 \rfloor - 1, d),$
- $\deg y_l \leq \max(d, d_{-1}) \cdot \lambda_{\mathsf{P}}(\lceil r/2 \rceil - 1, d),$
- $\deg y_r \leq \max(d, d_{-1}) \cdot \lambda_{\mathsf{P}}(\lceil r/2 \rceil - 2, d).$

It is easy to verify that the degrees of the products $x_r \cdot y_l$ and $x_r \cdot y_r$ are upper bounded by

$$\max(d, d_{-1}) \cdot (\lambda_{\mathsf{P}}(\lfloor r/2 \rfloor - 1, d) + \lambda_{\mathsf{P}}(\lceil r/2 \rceil - 1, d)) = \max(d, d_{-1}) \cdot \Lambda_{\mathsf{P}}(r - 2, d).$$

Similarly to Theorem 3.16, by the transpose-inverse property of the HDIM, the type-I distinguisher follows if

$$\max(d, d_{-1}) \cdot \Lambda_{\mathsf{P}}(r - 2, d) < 2n.$$

□

Remark 6. When $d_{-1} \leq d$, Theorem 3.18 provides a type-I distinguisher for 1 more round compared to the general Theorem 3.16.

Corollary 3.19. *Any $S \in \mathsf{P}_d^r$ has the type-I distinguisher if*

$$\max(d, d_{-1}) \cdot (d^{\lfloor r/2 \rfloor - 2} + d^{\lceil r/2 \rceil - 2}) < 2n,$$

and the type-II distinguisher if

$$\max(d, d_{-1}) \cdot (d^{\lfloor r/2 \rfloor - 2} + d^{\lceil r/2 \rceil - 3}) < 2n.$$

In particular, any 4-round Feistel Network with bijective round functions has the type-I distinguisher and any 5-round Feistel Network with bijective round functions has the type-II distinguisher.

Remark 7. Note that the results for Feistel Networks with bijective functions are not very useful if a degree bound on Feistel functions is known, but a degree bound on their inverses is not known. In such case only the generic 5-round type-II distinguisher can be obtained.

$(d, 2n)$	Structure	$r_{\max}$	Instance
$(2, 32)$	P_2^r	10	—
	F_2^r	9	SIMON-32 [BSS+13]
$(5, 64)$	P_5^r	7	—
	F_5^r	6	DES [Cop94]
$(31, 64)$	P_{31}^r	5	MISTY1/KASUMI [Mat97]
	F_{31}^r	4	—
$(n-1, 2n)$	P_{n-1}^r	5	—
	F_{n-1}^r	4	—

TABLE 3.2: Maximum number $r_{\max}$ of rounds in Feistel Networks with a type-II distinguisher. It is assumed that $d_{-1} \le d$.

3.3.3 Applications

As an illustration of the theorems, consider the HDIM of random Feistel Networks with 3-bit branches. For particular $S_4, S_5 \colon \mathbb{F}_2^6 \to \mathbb{F}_2^6$, $S_4 \in \mathsf{P}_2^4$ and $S_4 \in \mathsf{P}_2^5$ (the LAT of these functions was shown in Figure 3.2):

$$\mathsf{HDIM}_{S_4} = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}, \ \mathsf{HDIM}_{S_5} = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}. \tag{3.1}$$

It is interesting that from the HDIM we can see that all coordinates of S_5 have full degree $n-1$, but still the structure always has an integral distinguisher because particular monomials are always missing in the ANF.

Table 3.2 shows application of theorems to several concrete parameters.

In [Tod15], Todo proposed *division property*, a method to find integral characteristics. I compared our results with those reported by Todo in Appendix B of [Tod15]. Our HDIM-motivated results (type-II distinguishers) correspond to maximum number of rounds for which an integral characteristic is proven.

- For non-bijective cases, division property is better in 4 targets and provides the same results else. Those targets are (n, d)-Feistel networks $(24, 2), (48, 2), (48, 5), (64, 5)$. Our approach proves a distinguisher for one round less.

- For bijective cases, division property results in one more round than the respective non-bijective case in a few places. Our approach does not exploit this distinction in general and thus is weaker for these cases. However, under the assumption that the degree of the inverses of Feistel functions is upper-bounded by the same value (i.e. $d_{-1} \le d$), Corollary 3.19 provides identical results and even one more round for a three cases: (n, d) Feistel Networks $(32, 5), (32, 7), (64, 7)$. To the best of my knowledge, no known method existed to exploit a bound on the degree of the inverse functions in the division property framework. I describe such method in Section 3.3.4

As the results show, the division property proposed by Todo allows to get slightly stronger integral characteristics than the HDIM-motivated approach, except the cases when the degree of the inverses of Feistel functions is known. The downside of division property is that it requires an algorithmic evaluation for each parameter set, whereas our approach provides a simple closed formula. Furthermore, the degree growth inside the two halves of a primitive is evaluated by a generic bound in our approach. It may be possible to combine our approach with division property or another degree evaluation method to obtain better results. In particular, a recursive approach used in [BKP16] for SPNs may be useful for Feistel Networks as well.

3.3.4 Improving Division Property Propagation

I briefly note a method to improve the division property propagation rule given a bound on the algebraic degree of the inverse function of a permutation. I describe the division property using the equivalent characterization by Boura and Canteaut [BC16]. Recall that the indicator of a multiset is defined as the indicator of the set containing elements from the multiset with odd multiplicities.

Definition 3.20. *A multiset $X \subseteq \mathbb{F}_2^n$ is said to satisfy the division property $\mathcal{D}_k^n$, if*

$$\deg \mathbb{1}_X \leq n - k.$$

The main propagation rule of the division property is as follows (equivalently given in [Tod15] by Todo)

Proposition 3.21. *Let $X \subseteq \mathbb{F}_2^n$ be a multiset satisfying $\mathcal{D}_k^n$. Let F be a permutation of $\mathbb{F}_2^n$. Then the multiset $Y = F(X)$ satisfies the division property*

$$\mathcal{D}_{k'}^n, \textit{ for all } k' \leq \left\lceil \frac{k}{\deg F} \right\rceil.$$

Remark 8. I write inequality instead of original equality, to highlight all division properties that are satisfied, instead of only the strongest one.

I now show that another propagation rule can be obtained, if the degree of F^{-1} is known.

Proposition 3.22. *The multiset $Y = F(X)$ satisfies the division property*

$$\mathcal{D}_{k'}^n, \textit{ for all } k' \leq n - (n-k)\deg F^{-1}.$$

Proof. Without loss of generality, assume that X has no elements with a multiplicity greater than 1. Note that

$$x \in X \Leftrightarrow F(x) \in Y.$$

It can be rewritten as

$$\mathbb{1}_X = \mathbb{1}_Y \circ F.$$

Equivalently,

$$\mathbb{1}_X \circ F^{-1} = \mathbb{1}_Y.$$

It follows that

$$\deg \mathbb{1}_Y \leq \deg \mathbb{1}_X \cdot \deg F^{-1} \leq (n-k) \cdot \deg F^{-1},$$

and thus, $\deg \mathbb{1}_Y \leq n - k'$ for all

$$k' \leq n - (n-k) \deg F^{-1}.$$

□

Using this proposition, the results from [Tod15] can be improved for the case of bijective Feistel functions, assuming that $d_{-1} \leq d$. The improved division property then provides the same or slightly better results than Corollary 3.19 in all cases from [Tod15].

3.4 Feistel Networks with Affine Encodings

In this section, I describe decomposition attacks on Feistel Networks masked with affine layers, which I shall call affine encodings. The motivation for studying this structure comes from the fact that such encodings preserve most of the cryptographic properties (linearity, differential uniformity, algebraic degree) but make it harder to distinguish or decompose the structure. As an evidence, observe that attacks on the ASASA construction [DDKL15, MDFK15] are more involved than the attack on the SASAS construction [BS01]. Thew new attacks also expand the toolkit for the white-box cryptanalysis and S-Box reverse-engineering. I start with a formal definition of the analyzed structure.

Definition 3.23 (Feistel Network with Affine Encodings). $\mathsf{AF}^r_d\mathsf{A}$ *denotes the set of all permutations that can be expressed as an* r*-round Feistel Network with Feistel functions* $f_1, \ldots, f_r \colon \mathbb{F}_2^n \to \mathbb{F}_2^n$ *of algebraic degree at most* d*, and composed with bijective affine mappings from both sides. Furthermore, let* $\mathsf{A}^{-1}\mathsf{F}^r_d\mathsf{A}$ *denote the* $\mathsf{AF}^r_d\mathsf{A}$ *structure where the output affine encoding is the inverse of the input affine encoding; let* AF^r_d *denote the* $\mathsf{AF}^r_d\mathsf{A}$ *structure where the input affine encoding is the identity mapping:*

$$\begin{aligned}
\mathsf{AF}^r_d\mathsf{A} &:= \{\eta \circ R_{f_n} \circ \ldots \circ R_{f_1} \circ \mu \mid f_i \in \mathcal{F}^r_d, \mu, \eta \in \mathsf{GA}_n(\mathbb{F}_2)\}\,, \\
\mathsf{A}^{-1}\mathsf{F}^r_d\mathsf{A} &:= \left\{\mu^{-1} \circ R_{f_n} \circ \ldots \circ R_{f_1} \circ \mu \mid f_i \in \mathcal{F}^r_d, \mu \in \mathsf{GA}_n(\mathbb{F}_2)\right\}, \\
\mathsf{AF}^r_d &:= \{\mu \circ R_{f_n} \circ \ldots \circ R_{f_1} \mid f_i \in \mathcal{F}^r_d, \mu \in \mathsf{GA}_n(\mathbb{F}_2)\}\,,
\end{aligned}$$

where R_{f_i} *is a Feistel round as defined in Definition 3.1, and* $\mathcal{F}^r_d$ *denotes the set of all vectorial Boolean functions of degree at most* d *mapping* $\mathbb{F}_2^n$ *to itself.*

All the attacks in this section are based on the type-I and type-II integral distinguishers from Section 3.3. The constant additions on any of the sides do not change the integral property, in particular the HDIM. Therefore, it is sufficient to consider the case of linear encodings.

The cryptanalyst may compose a given structure $S \in \mathsf{AF}_d^r\mathsf{A}$ with additional random affine or linear encodings μ', η' and still obtain a structure $\eta' \circ S' \circ \mu' \in \mathsf{AF}_d^r\mathsf{A}$. In this way, it is possible to *re-randomize* the initial encoding for any of the three structures. Our attack works for affine encodings satisfying a particular (rather dense) property. The attack therefore applies to arbitrary encodings via the re-randomization. The following definition captures the class of linear permutations that our attack will target.

Definition 3.24. *Let $\mu \in \mathsf{GL}_{2n}(\mathbb{F}_2)$. μ is said to have a 2-UL decomposition, if there exist matrices $a, b, c, d \in \mathbb{F}_2^{n\times n}$ such that*

$$\mu = \begin{bmatrix} \mu_{1,1} & \mu_{1,2} \\ \mu_{2,1} & \mu_{2,2} \end{bmatrix} = \begin{bmatrix} I_{n\times n} & c \\ 0 & I_{n\times n} \end{bmatrix} \circ \begin{bmatrix} b & 0 \\ 0 & d \end{bmatrix} \circ \begin{bmatrix} I_{n\times n} & 0 \\ a & I_{n\times n} \end{bmatrix}.$$

Lemma 3.25. *It is sufficient that $\mu_{2,2}$ is invertible for μ to have a 2-UL decomposition.*

Proof. Note that

$$\begin{bmatrix} I_{n\times n} & c \\ 0 & I_{n\times n} \end{bmatrix} \circ \begin{bmatrix} b & 0 \\ 0 & d \end{bmatrix} \circ \begin{bmatrix} I_{n\times n} & 0 \\ a & I_{n\times n} \end{bmatrix} = \begin{bmatrix} c \times d \times a \oplus b & c \times d \\ d \times a & d \end{bmatrix}.$$

Set

$$\begin{aligned} d &= \mu_{2,2}, \\ a &= d^{-1} \times \mu_{2,1}, \\ c &= \mu_{1,2} \times d^{-1}, \\ b &= \mu_{0,0} \oplus c \times d \times a. \end{aligned}$$

□

The attacks presented in this section recover a partial information about the linear encodings. The underlying structure is not restricted to Feistel Network, it is only required that it has the type-I or type-II distinguisher. However, the partial information recovered is most useful in the Feistel Network case, as it allows to apply the decomposition attack on the unmasked Feistel Network.

3.4.1 Type-I Affine Encodings Recovery

The following theorem describes an attack against a structure with the type-I distinguisher masked with affine encodings. Type-I distinguisher is strong and provides enough equations to recover the required partial information from both sides of the structure.

Theorem 3.26 (Type-I Affine Encodings Recovery)**.** *Let $S\colon \mathbb{F}_2^{2n} \to \mathbb{F}_2^{2n}$ be a permutation that has the type-I distinguisher and let $\mathsf{HDIM}_S = \begin{bmatrix} h_{1,1} & 0 \\ 0 & 0 \end{bmatrix}$. Let*

$\mu, \eta \in \mathsf{GL}_{2n}(\mathbb{F}_2)$ such that μ and η^{-1} both have a 2-UL decomposition. Let $T \coloneqq \eta \circ S \circ \mu$.

Let (a, b, c, d) and (a', b', c', d') be the 2-UL decompositions of μ and η^{-1} respectively. Then, given T, a and a' can be recovered in time $\mathcal{O}(n2^{2n})$ if $h_{1,1}$ is invertible.

Proof. Observe that η can be expressed as

$$\eta = \begin{bmatrix} I_{n\times n} & 0 \\ a' & I_{n\times n} \end{bmatrix} \circ \begin{bmatrix} b'^{-1} & 0 \\ 0 & d'^{-1} \end{bmatrix} \circ \begin{bmatrix} I_{n\times n} & c' \\ 0 & I_{n\times n} \end{bmatrix}.$$

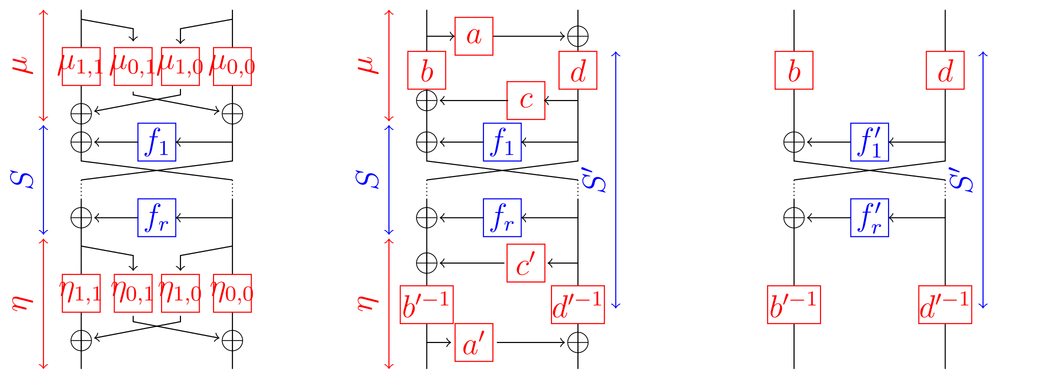
Let $S' \colon \mathbb{F}_2^{2n} \to \mathbb{F}_2^{2n}$ be given by:

$$S' = \begin{bmatrix} b'^{-1} & 0 \\ 0 & d'^{-1} \end{bmatrix} \circ \begin{bmatrix} I_{n\times n} & c' \\ 0 & I_{n\times n} \end{bmatrix} \circ S \circ \begin{bmatrix} I_{n\times n} & c \\ 0 & I_{n\times n} \end{bmatrix} \circ \begin{bmatrix} b & 0 \\ 0 & d \end{bmatrix}.$$

Then T can be expressed as:

$$T = \begin{bmatrix} I_{n\times n} & 0 \\ a' & I_{n\times n} \end{bmatrix} \circ S' \circ \begin{bmatrix} I_{n\times n} & 0 \\ a & I_{n\times n} \end{bmatrix}.$$

The relation between S, S' and T is illustrated in Figure 3.4 where S assumed to be a Feistel Network).



(A) $T = \eta \circ S \circ \mu$. (B) T (alt. representation). (C) S' (alt. representation).

FIGURE 3.4: The target of our attack, its result and its alternative representation. f'_i is affine equivalent to f_i.

Consider the HDIM of S' and S. They are related by Proposition 3.6:

$$\mathsf{HDIM}_{S'} = \begin{bmatrix} b'^{-1} & 0 \\ 0 & d'^{-1} \end{bmatrix} \circ \begin{bmatrix} I_{n\times n} & c' \\ 0 & I_{n\times n} \end{bmatrix} \circ \mathsf{HDIM}_S \circ \begin{bmatrix} I_{n\times n} & 0 \\ c^\top & I_{n\times n} \end{bmatrix} \circ \begin{bmatrix} (b^{-1})^\top & 0 \\ 0 & (d^{-1})^\top \end{bmatrix}.$$

It is easy to verify that

$$\mathsf{HDIM}_{S'} = \begin{bmatrix} b'^{-1} \times h_{1,1} \times (b^{-1})^\top & 0 \\ 0 & 0 \end{bmatrix}.$$

Let $h' = b'^{-1} \times h_{1,1} \times (b^{-1})^\top$. By a similar argument,

$$\mathsf{HDIM}_T = \begin{bmatrix} I_{n\times n} & 0 \\ a' & I_{n\times n} \end{bmatrix} \circ \mathsf{HDIM}_{S'} \circ \begin{bmatrix} I_{n\times n} & a^\top \\ 0 & I_{n\times n} \end{bmatrix} = \begin{bmatrix} h' & h' \times a^\top \\ a' \times h' & a' \times h' \times a^\top \end{bmatrix}.$$

Since $h_{1,1}$ is assumed to be invertible and b, b' are invertible too, h' is invertible. Therefore, a and a' can be easily recovered from HDIM_T in time $\mathcal{O}(n^3)$. The attack complexity is then dominated by the cost of computing HDIM_T, which can be done in $\mathcal{O}(n2^n)$ operations. □

Remark 9. If the rank of $h_{1,1}$ is not full but is high enough, it may still be possible to recover a and a' completely by including the quadratic equations from $(\mathsf{HDIM}_T)_{2,2}$.

The theorem shows that the type-I distinguisher provides $2n^2$ linear equations and n^2 quadratic equations. This is enough to recover $2n^2$ bits of information about the affine encodings. In the case of the type-II distinguisher, only n^2 quadratic equations are available for $2n^2$ unknowns. Still, the method can be applied to simplified structures. One possible scenario is the $\mathsf{A}^{-1}\mathsf{F}_d^r\mathsf{A}$ structure where the output linear layer is the inverse of the input linear layer. In this case though, the cryptanalyst has to solve a system of quadratic equations. Another scenario is the one-sided affine masking, i.e. the AF_d^r structure. In this case linear equations are obtained and the required partial information is recovered.

3.4.2 Type-II Affine Encodings Recovery

Theorem 3.27 (Type-II Affine Encodings Recovery, $\mathsf{A}^{-1}\mathsf{F}_d^r\mathsf{A}$)**.** *Let $S \colon \mathbb{F}_2^{2n} \to \mathbb{F}_2^{2n}$ be a permutation that has the type-II distinguisher and let $\mathsf{HDIM}_S = \begin{bmatrix} h_{1,1} & h_{1,2} \\ h_{2,1} & 0 \end{bmatrix}$. Let $\mu \in \mathsf{GL}_{2n}(\mathbb{F}_2)$ such that μ has a 2-UL decomposition. Let $T := \mu^{-1} \circ S \circ \mu$.*

Let (a, b, c, d) be the 2-UL decomposition of μ. Then, given T, a system of n^2 quadratic equations on a can be obtained.

Proof. Let $S' \colon \mathbb{F}_2^{2n} \to \mathbb{F}_2^{2n}$ be given by:

$$S' = \begin{bmatrix} b^{-1} & 0 \\ 0 & d^{-1} \end{bmatrix} \circ \begin{bmatrix} I_{n\times n} & c \\ 0 & I_{n\times n} \end{bmatrix} \circ S \circ \begin{bmatrix} I_{n\times n} & c \\ 0 & I_{n\times n} \end{bmatrix} \circ \begin{bmatrix} b & 0 \\ 0 & d \end{bmatrix}.$$

Similarly to the proof of Theorem 3.26,

$$\mathsf{HDIM}_{S'} = \begin{bmatrix} b^{-1} & 0 \\ 0 & d^{-1} \end{bmatrix} \circ \begin{bmatrix} I_{n\times n} & c \\ 0 & I_{n\times n} \end{bmatrix} \circ \mathsf{HDIM}_S \circ \begin{bmatrix} I_{n\times n} & 0 \\ c^\top & I_{n\times n} \end{bmatrix} \circ \begin{bmatrix} (b^{-1})^\top & 0 \\ 0 & (d^{-1})^\top \end{bmatrix},$$

$$\mathsf{HDIM}_{S'} = \begin{bmatrix} b^{-1} \times \left(h_{1,1} \oplus c \times h_{2,1} \oplus h_{1,2} \times c^\top\right) \times (b^{-1})^\top & b^{-1} \times h_{1,2} \times (d^{-1})^\top \\ d^{-1} \times h_{2,1} \times (b^{-1})^\top & 0 \end{bmatrix}.$$

Let

$$\mathsf{HDIM}_T = \begin{bmatrix} t_{1,1} & t_{1,2} \\ t_{2,1} & t_{2,2} \end{bmatrix}.$$

Then

$$\mathsf{HDIM}_{S'} = \begin{bmatrix} I_{n\times n} & 0 \\ a & I_{n\times n} \end{bmatrix} \circ \mathsf{HDIM}_T \circ \begin{bmatrix} I_{n\times n} & a^\top \\ 0 & I_{n\times n} \end{bmatrix} =$$
$$= \begin{bmatrix} t_{1,1} & t_{1,2} \oplus t_{1,1} \times a^\top \\ t_{2,1} \oplus a \times t_{1,1} & t_{2,2} \oplus a \times t_{1,2} \oplus t_{2,1} \times a^\top \oplus a \times t_{1,1} \times a^\top \end{bmatrix}.$$

The quadratic equation system follows:

$$(\mathsf{HDIM}_{S'})_{2,2} = t_{2,2} \oplus a \times t_{1,2} \oplus t_{2,1} \times a^\top \oplus a \times t_{1,1} \times a^\top = 0.$$

□

Theorem 3.28 (Type-II Affine Encodings Recovery, AF_d^r)**.** *Let* $S\colon \mathbb{F}_2^{2n} \to \mathbb{F}_2^{2n}$ *be a permutation that has the type-II distinguisher and let* $\mathsf{HDIM}_S = \begin{bmatrix} h_{1,1} & h_{1,2} \\ h_{2,1} & 0 \end{bmatrix}$. *Let* $\mu \in \mathsf{GL}_{2n}(\mathbb{F}_2)$ *such that* μ *has a 2-UL decomposition. Let* $T := \mu^{-1} \circ S$.

Let (a, b, c, d) *be the 2-UL decomposition of* μ. *Then, given* T, a *can be recovered in time* $\mathcal{O}(n2^n)$ *if* $h_{1,2}$ *is invertible.*

Proof. Let $S'\colon \mathbb{F}_2^{2n} \to \mathbb{F}_2^{2n}$ be given by:

$$S' = \begin{bmatrix} b^{-1} & 0 \\ 0 & d^{-1} \end{bmatrix} \circ \begin{bmatrix} I_{n\times n} & c \\ 0 & I_{n\times n} \end{bmatrix} \circ S$$

Then

$$\mathsf{HDIM}_{S'} = \begin{bmatrix} b^{-1} & 0 \\ 0 & d^{-1} \end{bmatrix} \circ \begin{bmatrix} I_{n\times n} & c \\ 0 & I_{n\times n} \end{bmatrix} \circ \mathsf{HDIM}_S,$$
$$\mathsf{HDIM}_{S'} = \begin{bmatrix} b^{-1} \times (h_{1,1} \oplus c \times h_{2,1}) & b^{-1} \times h_{1,2} \\ d^{-1} \times h_{2,1} & 0 \end{bmatrix}.$$

Let

$$\mathsf{HDIM}_T = \begin{bmatrix} t_{1,1} & t_{1,2} \\ t_{2,1} & t_{2,2} \end{bmatrix}.$$

Then

$$\mathsf{HDIM}_{S'} = \begin{bmatrix} I_{n\times n} & 0 \\ a & I_{n\times n} \end{bmatrix} \circ \mathsf{HDIM}_T = \begin{bmatrix} t_{1,1} & t_{1,2} \\ t_{2,1} \oplus a \times t_{1,1} & t_{2,2} \oplus a \times t_{1,2} \end{bmatrix}.$$

It follows that

$$(\mathsf{HDIM}_{S'})_{2,2} = t_{2,2} \oplus a \times t_{1,2} = 0.$$

Note that $t_{1,2} = b^{-1} \times h_{1,2}$ is invertible since $h_{1,2}$ is assumed to be invertible, therefore the linear system has full rank. □

In the next sections, I will describe how to continue the decomposition process of the unmasked Feistel Networks.

3.5 Lower-degree Artifacts in Feistel Networks

In previous section it was shown that the HDIM is a very convenient tool for attacking affine encodings. Affine encodings have rather low entropy and thus provably absent monomials of degree $2n-1$ in a Feistel Network provide enough equations to recover the encodings. However, much more equations are needed to recover a Feistel function. A straightforward direction is to consider lower-degree monomials as well, possibly at the cost of attacking fewer rounds. This has an extra benefit of finding more efficient integral distinguishers, since the data complexity of an integral distinguisher is exponential in the degree of the corresponding absent monomial in the ANF. In this section I use the generalization of the HDIM-ANF relation described in Section 3.2.4 in order to prove absence of lower degree monomials in Feistel networks. The method is quite similar to the method used for proving type-I distinguishers in Section 3.3. The main idea is to replace the sum variable in the expression to an intermediate state and thus split the structure in two halves.

The monomials in the ANF of Feistel Networks can be classified by the degree on the left input branch and on the right input branch. Clearly, all monomials in the same class have equivalent possibility of appearing in the ANF, since such monomials can be interchanged by composing Feistel functions with bit permutations.

Definition 3.29 (((w_l, w_r)-monomials)**.** *Let $u_l, u_r \in \mathbb{F}_2^n$ and let $u := (u_l, u_r) \in \mathbb{F}_2^{2n}$. The monomial x^u is said to be a (w_l, w_r)-monomial if $\mathbf{wt}(u_l) = w_l$ and $\mathbf{wt}(u_r) = w_r$. u is then said to be a (w_l, w_r)-exponent.*

Theorem 3.30. *Let $S \in \mathsf{F}_d^r$ and let $f = \langle e_i, S \rangle, n < i \le 2n$, be any coordinate of the right output branch of S. Let u be a (w_l, w_r)-exponent. Then $\rho_u[f] = 0$ if there exists an integer $r', 0 \le r' < r$ such that*

$$(n - w_l) \cdot \lambda_{\mathsf{F}}(r' + 1, d) + (n - w_r) \cdot \lambda_{\mathsf{F}}(r', d) + \lambda_{\mathsf{F}}(r - r', d) < 2n.$$

Similar result applies for $S \in \mathsf{P}_d^r$ by using $\lambda_{\mathsf{P}}(r, d)$.

Proof. Let $(a, b) \in \mathbb{F}_2^{2n}$ denote the intermediate state of S after r' rounds. Let $(x_l, x_r) \in \mathbb{F}_2^{2n}$ be the two input branches of S as functions of (a, b); $(y_l, y_r) \in \mathbb{F}_2^{2n}$ be the two output branches of S as functions of (a, b). By Proposition 3.9,

$$\rho_u[f] = \bigoplus_{z \in \mathbb{F}_2^{2n}} (\neg x_l)^{\neg u_l} (\neg x_r)^{\neg u_r} \langle e_i, y_r \rangle,$$

where $u_l, u_r \in \mathbb{F}_2^n$ are the two halves of u. (x_l, x_r) can be computed using an r'-round Feistel Network and (y_l, y_r) can be computed using an $(r - r')$-round Feistel Network. Further note that $\mathbf{wt}(\neg u_l) = n - \mathbf{wt}(u_l)$ and $\mathbf{wt}(\neg u_r) = n - \mathbf{wt}(u_r)$. The degree bounds follow:

- $\deg (\neg x_l)^{\neg u_l} \le (n - w_l)\lambda_{\mathsf{P}}(r' + 1, d)$,
- $\deg (\neg x_r)^{\neg u_r} \le (n - w_r)\lambda_{\mathsf{P}}(r', d)$,
- $\deg y_r \le \lambda_{\mathsf{P}}(r - r', d)$.

The theorem follows by summing the degree bounds and comparing to the full degree $2n$. □

A trick for bijective Feistel functions can be applied similarly to Theorem 3.18.

Theorem 3.31. *Let $S \in \mathsf{P}^r_d$ and let $f = \langle e_i, S\rangle, n < i \le 2n$ be any coordinate of the right output branch of S. Let u be a (w_l, w_r)-exponent. Then $\rho_u[f] = 0$ if there exists an integer $r', 0 \le r' < r - 3$ such that*

$$\max(d, d_{-1}) \cdot \big((n - w_l) \cdot \lambda_{\mathsf{P}}(r', d) + (n - w_r) \cdot \lambda_{\mathsf{P}}(r' - 1, d) + \lambda_{\mathsf{P}}(r - r' - 2, d)\big) < 2n.$$

Proof. The variables chosen are (a, c) instead of (a, b) (see Figure 3.3), where (a, b) denotes the intermediate state of S after r' rounds, and $c = f_{r'+1}(b) \oplus a$. In this case b can be expressed as $f^{-1}_{r'+1}(a \oplus c)$, and the degree of b as a function of (a, c) is upper bounded by d_{-1}.

Let x_l, x_r be the two input branches of S as functions of (a, c); y_l, y_r be the two output branches of S as functions of (a, c). Similarly to previous proofs, the following bounds are derived:

- $\deg x_l \le \max(d, d_{-1}) \cdot \lambda_{\mathsf{P}}(r', d)$,
- $\deg x_r \le \max(d, d_{-1}) \cdot \lambda_{\mathsf{P}}(r' - 1, d)$,
- $\deg y_l \le \max(d, d_{-1}) \cdot \lambda_{\mathsf{P}}(r - r' - 1, d)$,
- $\deg y_r \le \max(d, d_{-1}) \cdot \lambda_{\mathsf{P}}(r - r' - 2, d)$.

For Proposition 3.9, the following bounds are needed:

- $\deg (\neg x_l)^{\neg u_l} \le (n - w_l)\lambda_{\mathsf{P}}(r', d)$,
- $\deg (\neg x_r)^{\neg u_r} \le (n - w_r)\lambda_{\mathsf{P}}(r' - 1, d)$,
- $\deg y_r \le \lambda_{\mathsf{P}}(r - r' - 2, d)$.

The theorem follows by summing the degree bounds. □

Corollary 3.32. *Let $S \in \mathsf{P}^4_{n-1}$ and let $f = \langle e_i, S\rangle, n < i \le 2n$ be any coordinate of the right output branch of S. Then the following monomials classes are absent in the ANF of f (in total $2^n + n^2 + n$ monomials):*

(i) $(n - 1, n - 1)$,

(ii) $(n - 1, n)$,

(iii) (n, k) *for any* $0 \le k \le n$.

Proof. Set $r' = 0$ in Theorem 3.31. Note that in this extreme case the term $d_{-1} \cdot (n - w_r) \cdot \lambda_{\mathsf{P}}(r', d)$ for $r' = 0$ can be replaced by $(n - w_r)$, since the right input branch clearly has degree 1 on the chosen variables. For case (i) the condition becomes

$$1 + (n - 1)\,(1 \cdot \lambda_{\mathsf{P}}(1, n - 1) + \lambda_{\mathsf{P}}(1, n - 1)) \le 2n - 1 < 2n.$$

For case (ii) the condition becomes

$$0 + (n-1)\left(1 \cdot \lambda_{\mathsf{P}}(1, n-1) + \lambda_{\mathsf{P}}(1, n-1)\right) \le 2n - 2 < 2n.$$

For case (iii) the condition becomes

$$(n-k) + (n-1)\left(0 + \lambda_{\mathsf{P}}(1, n-1)\right) \le 2n - 1 - k < 2n.$$

$\square$

3.6 Decomposition Attack using Impossible Monomials

In this section I describe how large enough classes of impossible monomials can be used to mount a recovery attack on the last round's Feistel function.

The high level idea is the following. Consider a 5-round $2n$-bit Feistel Network with bijective Feistel functions, i.e. let $S^5 \in \mathsf{P}^5_{n-1}$. Let $S^4 \in \mathsf{P}^4_{n-1}$ be the Feistel Network consisting of the first 4 rounds of S^5 and let $f\colon \mathbb{F}_2^n \to \mathbb{F}_2^n$ be the Feistel function used in the last round. From Theorem 3.31, for any k, any (n,k)-monomial is not present in the ANFs of the right output branch of S^4. However, in the following 5-th round the output of the last Feistel function is xored into this branch and becomes the left output branch of S^5. This result now may or may not contain the (n,k)-monomials. By observing the presence of such monomials in the ANFs of the left branch of S^5, we can deduce some information about the last Feistel function in the form of linear equations. If the number of impossible monomials is large enough, an equivalent of the Feistel function f can be recovered. For an illustration see Figure 3.5, where the 5-th round of a Feistel Network with 3-bit branches is shown. a_u denotes the ANF coefficient of a monomial that is impossible in the right branch of a 4-round Feistel Network.

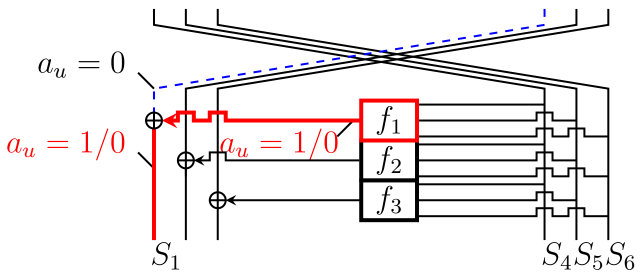


FIGURE 3.5: Impossible monomials in the last round of a 5-round FN S with 3-bit branches. The wire with 4-round impossible monomials is in dashed blue, the path of the observed monomials is highlighted with bold red. a_u is the ANF coefficient of some 4-round impossible monomial.

More formally, observe that by the Feistel structure

$$(\mathsf{left} \circ S^5) \oplus (f \circ \mathsf{right} \circ S^5) = \mathsf{right} \circ S^4.$$

Consider an arbitrary coordinate position $i, 1 \le i \le n$. For any monomial x^u that is impossible in $\mathsf{right} \circ S^4$ (e.g. any (n,k)-monomial),

$$\rho_u \left[\left\langle e_i, \mathsf{left} \circ S^5 \right\rangle\right] \oplus \rho_u \left[\left\langle e_i, f \circ \mathsf{right} \circ S^5 \right\rangle\right] = 0.$$

By decomposing f through its ANF,

$$\rho_u \left[\left\langle e_i, \mathsf{left} \circ S^5 \right\rangle\right] = \bigoplus_{v \in \mathbb{F}_2^n} \rho_u \left[\rho_v \left[\langle e_i, f \rangle\right] (\mathsf{right} \circ S^5)^v\right].$$

Since S^5 is known, this can be considered as a linear equation on the unknown ANF coefficients of $\langle e_i, f\rangle$. In total, there are $2^n - 1$ equations (from 2^n impossible (n,k)-monomials, except the (n,n)-monomial) and $2^n - 1$ unknowns (the constant is excluded). More equations can be obtained by considering the other classes of impossible monomials from Theorem 3.31. Therefore, it can be expected that the system will have (close to) full rank with high probability and the solution will be unique. The algorithm of the attack is given in Algorithm 3.1.

Algorithm 3.1 Feistel Function Recovery Attack

```
Input: the full codebook of a function $S \in \mathsf{F}_d^r, S\colon \mathbb{F}_2^{2n} \to \mathbb{F}_2^{2n}$; a set $U \subseteq \mathbb{F}_2^{2n}\}$.
Output: a function $f\colon \mathbb{F}_2^n \to \mathbb{F}_2^n, \deg f \le d$ if exists, such that
    for all $u \in U$ and for all $i, 1 \le i \le n$, $\rho_u[\langle e_i, \mathsf{right} \circ R_f \circ S\rangle] = 0$.
 1: $V \leftarrow \{v \in \mathbb{F}_2^n \mid 1 \le \mathbf{wt}(v) \le d\}$
 2: $M \leftarrow$ a $|U| \times |V|$ matrix indexed by $U$ and $V$
 3: for all $i \in [1 \ldots n]$ do
 4:     $b^i \leftarrow$ a $|U|$-bit vector indexed by $U$
 5: for all $u \in U$ do
 6:     for all $v \in V$ do
 7:         $M_{u,v} \leftarrow \bigoplus_{x \preceq u} (\mathsf{right} \circ S(x))^v$
 8:     for all $i \in [1 \ldots n]$ do
 9:         $b^i_u \leftarrow \bigoplus_{x \preceq u} \langle e_i, \mathsf{left} \circ S(x)\rangle$
10: for all $i \in \{1, \ldots, n\}$ do
11:     $a \leftarrow$ a solution of $M \times a = b^i$
12:     $f_i \leftarrow (x \mapsto \oplus_{v \in V} a_v x^v)$
13: return $f = (f_1, \ldots, f_n)$
```

3.6.1 On the Assumptions

In the decomposition attack it is assumed that the equation system will have full or close to full rank. Then the correct Feistel function f will be among one of the few system's solutions.

One reason for a possible rank deficiency is that the low-degree monomials in the ANF of the Feistel function f may not generate the high-degree 4-round impossible monomials. In such case the equations generated from the high-degree

4-round impossible monomials provide no information about the low-degree monomials of f. In particular, Theorem 3.18 proves that linear monomials of f can not generate $(n, n-1)$ monomials when composed with the right output branch of a 5-round Feistel Network.

To the best of my knowledge, there are no known ways to prove even a possibility of presence of any highest-degree monomials in, for example, a 5-round Feistel Network. Indeed, in general, proving lower bounds on the algebraic degree is a very difficult problem.

3.6.2 Instantiations

The attack is not restricted to the case of a 5-round Feistel Network with bijective functions. The requirement is to have enough impossible monomials, which can be obtained from Theorems 3.30,3.31 or by another analysis methods. In practice, a cryptanalyst can generate random instances of the analyzed structure and empirically determine all impossible monomial classes with high probability. This analysis will not dominate the complexity.

The described 5-round attack corresponds to the case of the type-II distinguisher. It exploits a large amount of impossible monomials in a 4-round network, i.e. the one that has the type-I distinguisher. I propose a conjecture on the generalization of this rule.

Conjecture 3.33. *Let r be the maximum number of rounds such that all $S \in \mathsf{F}_d^r$ have the type-II distinguisher. Then the impossible monomial attack succeeds with high probability on all $S \in \mathsf{F}_d^r$, i.e. it outputs a negligible number of candidates for the last Feistel function, and the correct one is always among them.*

Experiment. I have implemented the attack in Sage [SD19] and performed a few experiments on small values of the branch size n. For all $n \in \{3, 4, 5, 6, 7\}$ and $d \in \{2, 3, n-1, n\}$, I generated 100 random instances of F_d^r (and P_{n-1}^r) for maximum r such that the structure has the type-II distinguisher. Then for the first $r-1$ rounds I empirically evaluated all impossible monomial classes. Using these classes, I generated the equation system of the impossible monomial attack for each of the 100 instances. I computed the average rank of the system and the system's dimension, i.e. the number of unknowns. In addition, I verified that the actual last round Feistel function satisfies the equations. The results of the experiment are given in Table 3.3.

The results show that the rank is close to the maximum on average. It means that there are only a few solutions on average and the impossible monomial attack succeeds in the analyzed cases. The rank deficiency is larger for cases with $n = 3$ and decreases fast with the growth of n. Furthermore, the results confirm the conjecture on the analyzed cases.

3.6.3 Relation with Integral Attack from [BLP15]

An integral distinguisher was already used to mount a Feistel function recovery attack by Biryukov *et al.* [BLP15]. They show that, for a 5-round Feistel Network, a 4-round integral distinguisher provides a linear equation on the *values*

n	#rounds : avg. rank / dimension				
	$d = 2$	$d = 3$	$d = n - 1$	$d = n$	$d = n - 1$, bij.
3	5 : 3.80/6	3 : 6.62/7	5 : 3.80/6	3 : 6.62/7	5 : 4.21/6
4	5 : 7.63/10	4 : 13.97/14	4 : 13.97/14	3 : 15.00/15	5 : 13.97/14
5	6 : 14.97/15	5 : 24.06/25	4 : 30.00/30	3 : 31.00/31	5 : 30.00/30
6	7 : 20.16/21	5 : 41.00/41	4 : 62.00/62	3 : 63.00/63	5 : 62.00/62
7	7 : 28.00/28	5 : 63.00/63	4 : 126.0/126	3 : 127.0/127	5 : 126.0/126

TABLE 3.3: The maximum number of rounds for type-II distinguisher and the average rank of the equation system in the impossible monomial attack on $2n$-bit Feistel Networks. Evaluated experimentally on 100 random instances per each parameter set.

of the last Feistel function. The impossible monomial attack described in this section considers the same equation system but in the *monomial basis*. That is, the unknown variables are the monomial coefficients in the ANFs of the coordinates of the Feistel function. Since the ANF coefficients can be computed by summing the function over particular sets, this is a change of basis (in other words, the Möbius transform is linear). The advantage of the monomial basis is that an upper bound on the degree of the Feistel functions can be used to decrease the number of unknowns.

3.7 Conclusions

This work started by observing interesting patterns in the LAT modulo 8 of small Feistel Networks. The analysis of the patterns resulted in the definition of High-Degree Indicator Matrix (HDIM). This tool shows a link between the LAT and the highest-degree monomials in the ANF. Furthermore, its properties allow to prove upper bounds on the algebraic degree of cryptographic structures and to prove finer algebraic degeneracies. Though these results do not improve the state of the art, the upper bounds given are expressed in a simple closed formula and there is a room for improvement, e.g. by combining methods. Finally, the most useful application of HDIM is in the cryptanalysis of Feistel Networks masked with secret affine layers. The generalized HDIM-motivated ideas allow to prove lower-degree degeneracies as well, i.e. impossible monomials. I show how they can be used to mount decomposition attacks on Feistel Networks. The results of this chapter together allow to fully decompose affinely-whitened Feistel Networks satisfying the attack conditions. I think it provides many useful tools for S-Box reverse-engineering and white-box analysis toolkit.

The work leaves several open problems:

1. Better degree evaluation. Is it possible to improve the HDIM-motivated method? Is it possible to combine it with other methods, e.g. division property?

2. Proving Conjecture 3.33. Are there always enough impossible monomials to recover the last Feistel function, if the type-II distinguisher applies?

3. In which cases it is possible to decompose Feistel Networks having at least 1 more round than Feistel Networks satisfying the type-II distinguisher?

4. A big open problem: lower bounds in Feistel Networks or Substitution-Permutation Networks. How to prove non-trivial lower bounds on the degree of a structure, i.e. that at least one instance of the structure has high enough degree? Strong lower bounds could shed light on how close are current degree evaluation methods to optimal ones.

Chapter 4

Decompositions of the GOST S-Box

In this chapter, I describe two interesting decompositions of the S-Box used in the recent Russian cryptographic “GOST” standards (Kuznyechik block cipher and Streebog [Fed12] hash function). The S-Box was also used in the first version of the authenticated cipher STRIBOB [Saa14], a candidate of the CAESAR competition; later, the S-Box was replaced. This chapter is based on the joint work with Alex Biryukov and Léo Perrin from EUROCRYPT 2016 [BPU16] and on the joint work with Léo Perrin from TOSC 2016 [PU17]. In the first work we describe a Feistel Network-like decomposition with finite field multiplications and affine whitening layers. In the second work we show that this structure is related to exponentiation/logarithm in the finite field, which was also used in the standard block cipher of Belarus, BelT [Bel11].

4.1 Introduction

S-Boxes play important role in the design of symmetric cryptographic primitives. It is one of the two components of an SPN structure and often S-Boxes are used inside the Feistel functions in Feistel Networks. The main role of S-Boxes is to provide non-linearity and confusion. An S-Box at least should have low linearity, low differential uniformity and high algebraic degree. It is also desirable that the S-Box has good implementation properties: an efficient hardware/bit-slice implementation, small size in order to reduce the memory footprint.

The cryptographic community expects designers to explain all choices done during the design procedure. How the S-Boxes were generated? Do they have an algebraic structure, e.g. an inversion in the finite field? Or do they have a Feistel Network structure? Were they generated at random? If yes, what was the seed used? Which cryptographic properties were optimized and how?

Unfortunately, often the designers describe the S-Box as a look-up table and do not provide any rationale behind its choice. A prominent example is the S-Box of the Skipjack block cipher designed by the American National Security Agency (NSA). Léo Perrin and Alex Biryukov [BP15] attempted to *reverse-engineer* it, i.e. to find the hidden design criteria, an underlying structure or optimization procedure. They succeeded and described a simple optimization method which generates S-Boxes with very close cryptographic properties. The designers of the Russian cryptographic standards did not disclose any rationale behind the S-Box as well, except that it has reasonable cryptographic properties.

The 8-bit S-Box used in the Kuznyechik block cipher and in the Streebog hash function is denoted π in this chapter. The look-up table of $\pi\colon \mathbb{F}_2^8 \to \mathbb{F}_2^8$ is given in Table 4.1. It has linearity equal to 56 and differential uniformity equal to 8. Using methods developed in [BP15], it can be shown that the probability to randomly sample an S-Box with as good differential properties is approximately $2^{-82.69}$. It follows that π has strong resistance against differential cryptanalysis, compared to random S-Boxes. The algebraic degree of all coordinates of π is maximal and equal to 7.

In this chapter I describe two decompositions of π and the way in which they were obtained. A simplified view of the discovered structures of π is given in Figure 4.1. The first decomposition is based on finite field multiplications. It also contains four 4-bit S-Boxes and two whitening (external) linear layers. Interestingly, 16 inputs clearly stand out from the patterns and force the usage of a multiplexer (omitted in the simplified view). The second decomposition is based on a finite field logarithm. It contains only one extra 4-bit S-Box, one whitening linear layer and a simple arithmetic layer.

More recently, my former colleague Léo Perrin studied the logarithm-based decomposition further [Per19]. He shows that the S-Box maps a partition of $\mathbb{F}_2^8$ into multiplicative cosets of $\mathbb{F}_{2^4}^*$ into a partition of $\mathbb{F}_2^8$ into additive cosets of $\mathbb{F}_2^4$. Furthermore, he derives a structure called TKlog that π follows.

	.0	.1	.2	.3	.4	.5	.6	.7	.8	.9	.a	.b	.c	.d	.e	.f
0.	fc	ee	dd	11	cf	6e	31	16	fb	c4	fa	da	23	c5	04	4d
1.	e9	77	f0	db	93	2e	99	ba	17	36	f1	bb	14	cd	5f	c1
2.	f9	18	65	5a	e2	5c	ef	21	81	1c	3c	42	8b	01	8e	4f
3.	05	84	02	ae	e3	6a	8f	a0	06	0b	ed	98	7f	d4	d3	1f
4.	eb	34	2c	51	ea	c8	48	ab	f2	2a	68	a2	fd	3a	ce	cc
5.	b5	70	0e	56	08	0c	76	12	bf	72	13	47	9c	b7	5d	87
6.	15	a1	96	29	10	7b	9a	c7	f3	91	78	6f	9d	9e	b2	b1
7.	32	75	19	3d	ff	35	8a	7e	6d	54	c6	80	c3	bd	0d	57
8.	df	f5	24	a9	3e	a8	43	c9	d7	79	d6	f6	7c	22	b9	03
9.	e0	0f	ec	de	7a	94	b0	bc	dc	e8	28	50	4e	33	0a	4a
a.	a7	97	60	73	1e	00	62	44	1a	b8	38	82	64	9f	26	41
b.	ad	45	46	92	27	5e	55	2f	8c	a3	a5	7d	69	d5	95	3b
c.	07	58	b3	40	86	ac	1d	f7	30	37	6b	e4	88	d9	e7	89
d.	e1	1b	83	49	4c	3f	f8	fe	8d	53	aa	90	ca	d8	85	61
e.	20	71	67	a4	2d	2b	09	5b	cb	9b	25	d0	be	e5	6c	52
f.	59	a6	74	d2	e6	f4	b4	c0	d1	66	af	c2	39	4b	63	b6

TABLE 4.1: The S-Box π in hexadecimal. For example, $\pi(\texttt{C2}) = \texttt{B3}$.

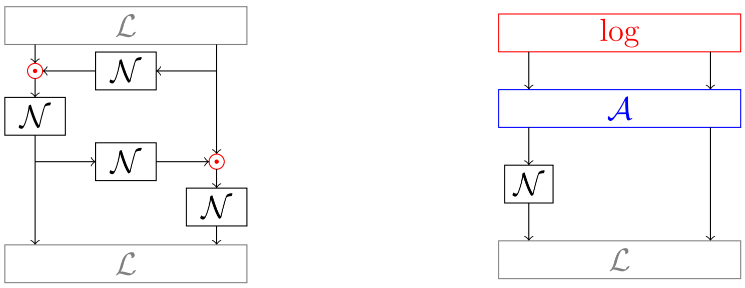


FIGURE 4.1: A simplified view of two decompositions of π. Linear (resp. nonlinear) functions are denoted $\mathcal{L}$ (resp. $\mathcal{N}$). $\odot$ denotes finite field multiplication and *log* is a finite field logarithm. $\mathcal{A}$ denotes a simple integer arithmetic layer.

4.1.1 Outline

Section 4.2 described the first decomposition, and Section 4.3 explains the second decomposition. The results are summarized and discussed in Section 4.4.

4.1.2 Differences with [BPU16, PU17]

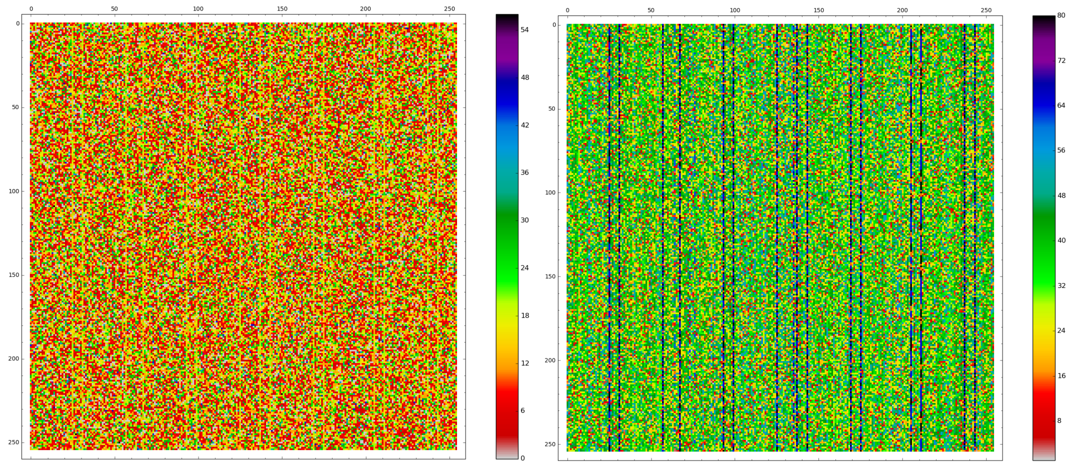
This chapter is a reworked version of the two papers [BPU16, PU17] that we wrote with my colleagues Alex Biryukov and Léo Perrin. In this chapter I kept only results directly related to decompositions of π. The decompositions are kept the same, except that for the first decomposition I performed the analysis for π from the beginning, without decomposing π^{-1} first. In this way T and U are inverted and swapped, compared to [BPU16]. The final decomposition is the same.

4.2 Feistel-like Decomposition based on Finite Field Multiplications

Similarly to Chapter 3, this chapter illustrates the usefulness of the "Jackson Pollock representation" of the LAT of an S-Box. Consider a heatmap of LAT_π shown in Figure 4.2a. It looks rather random overall, except several vertical stripes clearly sticking out. This effect is weaker or stronger depending on the colormap chosen for plotting. By a closer inspection it can be observed that the stripes stick out because of the same color appearing mote often than in another columns. In order to strengthen the effect, I define a *column frequency table.*

Definition 4.1. *Let L be an $n \times m$ matrix. The* column frequency table *of L is the $n \times m$ matrix $\mathsf{CF}(L)$ over $\mathbb{Z}$ given by:*

$$\mathsf{CF}(L)[y,x] := \left|\{y' \mid L[y',x] = L[y,x]\}\right|.$$



(A) LAT_π (B) $\mathsf{CF}(\mathsf{LAT}_\pi)$

FIGURE 4.2: Jackson Pollock representation of the LAT of π and its column frequency table.

The column frequency table of the LAT_π is shown in Figure 4.2b. The same columns are clearly sticking out as in the LAT of π. Let S denote their x-coordinates:

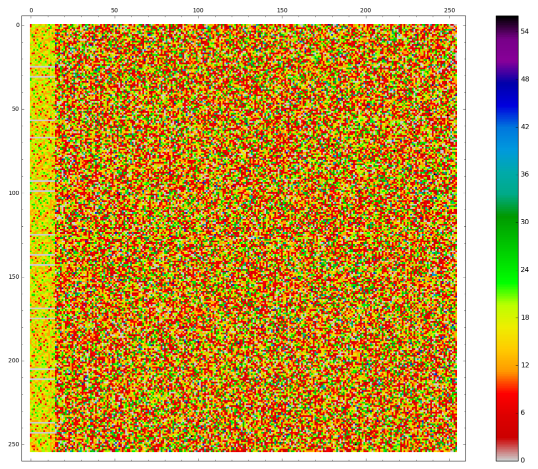
$$S = \{\texttt{00}, \texttt{1A}, \texttt{20}, \texttt{3A}, \texttt{44}, \texttt{5E}, \texttt{64}, \texttt{7E}, \texttt{8A}, \texttt{90}, \texttt{AA}, \texttt{B0}, \texttt{CE}, \texttt{D4}, \texttt{EE}, \texttt{F4}\} \subseteq \mathbb{F}_2^8.$$

Note that 00 was added in order to complete the set to a linear subspace of $\mathbb{F}_2^8$. It follows that we can choose 4 linearly independent coordinates and they will correspond to 4 linearly independent components of π. By composing π with a linear map, the outstanding columns of the LAT_π can be grouped together. Let

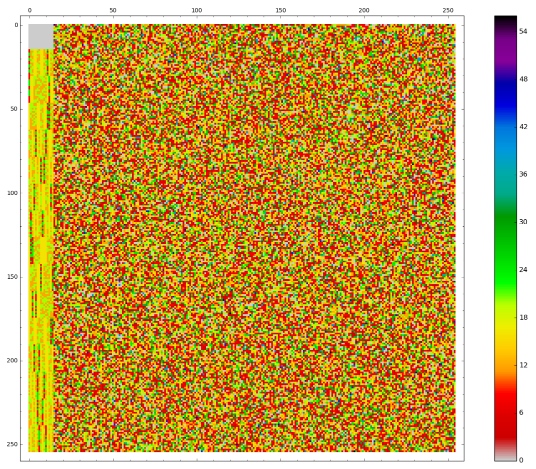
$L \in \mathsf{GL}_8(\mathbb{F}_2)$ be such that

$$L(\texttt{80}) = \texttt{08}, \quad L(\texttt{40}) = \texttt{04}, \quad L(\texttt{20}) = \texttt{02}, \quad L(\texttt{10}) = \texttt{01},$$
$$L(\texttt{08}) = \texttt{8A}, \quad L(\texttt{04}) = \texttt{44}, \quad L(\texttt{02}) = \texttt{20}, \quad L(\texttt{01}) = \texttt{1A}.$$

Let $\pi_1 := L^\top \circ \pi$. The LAT of π_1 is shown in Figure 4.3a. According to Proposition 2.7 from Chapter 2, the outstanding columns are grouped on the left. Furthermore, inside these 16 columns we can now observe similarly outstanding rows. Coincidentally, their coordinates form the same linear subspace S. In order to group the rows on the top, let $\pi_2 := L^\top \circ \pi \circ (L^{-1})^\top$. The LAT of π_2 is shown in Figure 4.3b.



(A) LAT_{π_1}



(B) LAT_{π_2}

FIGURE 4.3: Jackson Pollock representation of the LAT of π_1 and π_2.

4.2.1 TU-decomposition

The LAT of π_2 has interesting artifacts. The special 16 columns now have a visible structure consisting of 16×16 squares. More importantly, the topmost square fully consists of zeroes, i.e. $\mathsf{LAT}_{\pi_2}(a, b) = 0$ for $0 \preceq a, b \preceq \texttt{0F}$. These zeroes can be interpreted as follows: if we fix any linear combination of the 4 rightmost input bits to any constant, then any linear combination of the 4 rightmost output bits is balanced. Following this idea, the following multiset property can be verified: for any $c \in \mathbb{F}_2^4$,

$$\mathsf{right}\,(\pi_2(X)) = \mathbb{F}_2^4, \text{ where } X := \left\{(l, c) \mid l \in \mathbb{F}_2^4\right\}.$$

In other words, there exists 16 permutations $T_\texttt{0}, \ldots, T_\texttt{F}$ of $\mathbb{F}_2^4$ such that for all $l, r \in \mathbb{F}_2^4$

$$\mathsf{right}(\pi_2(l, r)) = T_r(l).$$

Let $U_\texttt{0}, \ldots, U_\texttt{F} \colon \mathbb{F}_2^4 \to \mathbb{F}_2^4$ be such that $U_{T_r(l)}(r) := \mathsf{left}(\pi_2(l, r))$ for all $l, r \in \mathbb{F}_2^4$. Then

$$\pi_2(l, r) = \left(U_{T_r(l)}(r), T_r(l)\right).$$

The high-level decomposition of π_2 into T and U is shown in Figure 4.4 and the look-up tables of T and U are given in Table 4.2. Note that since π_2 and all T_i are permutations, all U_i must be permutations as well. It can be easily verified from the look-up table of U. Due to this bijectivity, T and U can be viewed as mini-block ciphers. Such decomposition into two mini block-ciphers shall be called a *TU-decomposition*. It will prove its usefulness again in Chapter 5.

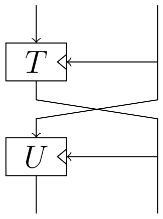


FIGURE 4.4: TU-decomposition of π_2.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
T_0	2	A	3	9	E	5	6	B	0	7	F	D	C	4	8	1
T_1	7	A	6	3	9	D	C	2	0	E	F	B	1	4	8	5
T_2	6	8	9	0	1	7	F	C	5	3	E	D	A	4	2	B
T_3	4	C	6	E	B	7	9	5	1	2	3	0	F	A	D	8
T_4	E	7	1	D	8	2	B	6	5	C	F	3	0	A	4	9
T_5	F	7	2	4	0	6	D	5	3	E	8	9	A	B	1	C
T_6	5	A	3	4	7	8	1	6	D	B	E	2	F	9	C	0
T_7	A	3	B	E	7	6	F	0	C	9	1	8	2	D	4	5
T_8	9	B	F	D	5	7	A	8	C	E	0	2	4	6	3	1
T_9	E	4	0	1	9	7	D	A	F	8	B	5	2	3	C	6
T_A	7	4	9	E	F	2	8	3	D	0	A	1	5	6	B	C
T_B	7	0	2	5	3	B	9	1	8	C	E	A	4	D	F	6
T_C	D	C	4	8	7	3	0	B	F	E	6	A	5	1	2	9
T_D	E	1	F	5	7	D	3	C	6	2	A	9	B	8	0	4
T_E	2	7	8	E	5	0	C	A	B	1	6	D	3	9	F	4
T_F	C	7	4	B	F	1	A	2	6	9	E	5	8	0	D	3

(A) T.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
U_0	C	7	2	8	E	3	F	4	6	D	B	5	9	A	0	1
U_1	C	8	0	A	3	F	5	6	9	1	2	7	D	B	4	E
U_2	C	7	2	8	E	3	F	4	6	D	B	5	9	A	0	1
U_3	C	9	3	D	2	0	4	5	7	A	E	6	8	1	F	B
U_4	8	9	B	5	4	0	2	3	C	F	7	D	E	A	1	6
U_5	8	7	C	B	D	9	5	F	6	0	3	A	4	1	E	2
U_6	8	4	9	0	C	E	F	A	7	1	D	6	B	2	5	3
U_7	8	9	B	5	4	0	2	3	C	F	7	D	E	A	1	6
U_8	E	D	8	7	F	3	C	0	2	4	A	1	6	5	9	B
U_9	E	8	6	9	D	7	5	B	F	C	2	A	3	0	4	1
U_A	E	7	9	5	3	C	1	2	6	B	8	D	4	A	0	F
U_B	E	B	1	F	0	2	6	7	5	8	C	4	A	3	D	9
U_C	A	D	0	4	3	1	E	B	7	5	2	C	8	6	F	9
U_D	A	C	2	D	9	3	1	F	B	8	6	E	7	4	0	5
U_E	A	9	C	3	B	7	8	4	6	0	E	5	2	1	D	F
U_F	A	1	4	E	8	5	9	2	0	B	D	3	F	C	6	7

(B) U.

TABLE 4.2: The mini-block ciphers used to decompose π_2.

Remark 10. It might seem that the TU-decomposition provides little insight into the structure. Indeed, any 8-bit function can be described by two tables of the same size as T and U, for example by considering the left and right halves of the output separately. The only special property that TU-decomposition adds is that each T_i is a permutation (and thus, each U_i). This is a very unlikely event that a random permutation has such decomposition, even if extra linear encodings (such as L in the case of π) are allowed. This property justifies the separation of T and U and their independent analysis.

The decomposition procedure of T and U are described in Section 4.2.3 and Section 4.2.2 respectively.

4.2.2 Decomposition of U

Let $\alpha\colon \mathbb{F}_2^4 \to \mathbb{F}_2^4$ be given by

$$\alpha(x) := U_x(0)$$

and let $U'_0, \ldots, U'_{\mathtt{F}}$ be permutations of $\mathbb{F}_2^4$ given by

$$U'_k(x) := U_k(x) \oplus \alpha(k).$$

It follows that for all $k \in \mathbb{F}_2^4$, $U'_k(0) = 0$. The codebook of U' is given in Figure 4.3a.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
U'_0	0	B	E	4	2	F	3	8	A	1	7	9	5	6	C	D
U'_1	0	4	C	6	F	3	9	A	5	D	E	B	1	7	8	2
U'_2	0	B	E	4	2	F	3	8	A	1	7	9	5	6	C	D
U'_3	0	5	F	1	E	C	8	9	B	6	2	A	4	D	3	7
U'_4	0	1	3	D	C	8	A	B	4	7	F	5	6	2	9	E
U'_5	0	F	4	3	5	1	D	7	E	8	B	2	C	9	6	A
U'_6	0	C	1	8	4	6	7	2	F	9	5	E	3	A	D	B
U'_7	0	1	3	D	C	8	A	B	4	7	F	5	6	2	9	E
U'_8	0	3	6	9	1	D	2	E	C	A	4	F	8	B	7	5
U'_9	0	6	8	7	3	9	B	5	1	2	C	4	D	E	A	F
$U'_{\mathtt{A}}$	0	9	7	B	D	2	F	C	8	5	6	3	A	4	E	1
$U'_{\mathtt{B}}$	0	5	F	1	E	C	8	9	B	6	2	A	4	D	3	7
$U'_{\mathtt{C}}$	0	7	A	E	9	B	4	1	D	F	8	6	2	C	5	3
$U'_{\mathtt{D}}$	0	6	8	7	3	9	B	5	1	2	C	4	D	E	A	F
$U'_{\mathtt{E}}$	0	3	6	9	1	D	2	E	C	A	4	F	8	B	7	5
$U'_{\mathtt{F}}$	0	B	E	4	2	F	3	8	A	1	7	9	5	6	C	D

(A) U'.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	order
M_0	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	1
M_1	0	3	E	D	5	6	B	8	C	F	2	1	9	A	7	4	15
M_2	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	1
M_3	0	C	5	9	2	E	7	B	1	D	4	8	3	F	6	A	15
M_4	0	9	6	F	E	7	8	1	3	A	5	C	D	4	B	2	5
M_5	0	5	3	6	C	9	F	A	2	7	1	4	E	B	D	8	5
M_6	0	E	9	7	3	D	A	4	5	B	C	2	6	8	F	1	3
M_7	0	9	6	F	E	7	8	1	3	A	5	C	D	4	B	2	5
M_8	0	6	D	B	9	F	4	2	E	8	3	5	7	1	A	C	15
M_9	0	D	7	A	6	B	1	C	9	4	E	3	F	2	8	5	15
$M_{\mathtt{A}}$	0	B	A	1	F	4	5	E	7	C	D	6	8	3	2	9	15
$M_{\mathtt{B}}$	0	C	5	9	2	E	7	B	1	D	4	8	3	F	6	A	15
$M_{\mathtt{C}}$	0	A	8	2	B	1	3	9	F	5	7	D	4	E	C	6	5
$M_{\mathtt{D}}$	0	D	7	A	6	B	1	C	9	4	E	3	F	2	8	5	15
$M_{\mathtt{E}}$	0	6	D	B	9	F	4	2	E	8	3	5	7	1	A	C	15
$M_{\mathtt{F}}$	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	1

(B) M.

TABLE 4.3: The mini-block ciphers U' and M.

In [BCBP03], Biryukov *et al.* propose efficient algorithms for checking affine and linear equivalence of permutations. Applying these algorithms to the permutations U'_i shows that all U'_i are pairwise linear equivalent. Furthermore, they differ only by a linear layer in the output. Formally, let $M_0, \ldots, M_{\mathtt{F}}\colon \mathbb{F}_2^4 \to \mathbb{F}_2^4$ be given by

$$M_k(x) := U'_k(x) \circ U_0'^{-1}(x).$$

Then, each M_i is linear. The codebook of M is given in Figure 4.3b.

Remark 11. In the hindsight, it could be trivially checked that $U_i \circ U_j^{-1}$ is linear for all $i, j \in \mathbb{F}_2^4$. However, it is not always clear which properties or relations can be expected. For this reason, the linear/affine equivalence algorithms from [BCBP03] and their improved variants by Dinur [Din18] are very useful tools for S-Box reverse-engineering.

The next step is to observe that the functions M_i have two interesting properties:

1. the functions M_i have orders $1, 3, 5, 15$; those with order 15 generate all M_i;

2. the functions M_i are linearly related: they are contained in linear subspace of dimension 4;

These properties point towards a finite field structure. Let $b := M_0 \oplus M_5$. Then b is *linear-similar* to the multiplication by X in the finite field

$$\mathbb{F}_{2^4} \simeq \mathbb{F}_2[X]/(X^4 + X^3 + 1).$$

By "linear-similar" it is meant that

$$b = l \circ (\cdot \ \odot X) \circ l^{-1}$$

for some linear bijection $l \in \mathsf{GL}_4(\mathbb{F}_2)$, where $(\cdot \ \odot X)$ denotes the multiplication in the finite field by X. In this case, $l = l' = \mathsf{swap2lsb}$, where

$$\mathsf{swap2lsb}\colon \mathbb{F}_2^4 \to \mathbb{F}_2^4, \mathsf{swap2lsb}(x_1, x_2, x_3, x_4) := (x_1, x_2, x_4, x_3).$$

Among all choices of b and the field defining polynomial, this choice results in the simplest mapping l.

Note that similarity is preserved for powers, i.e. $b^i = \mathsf{swap2lsb} \circ (\cdot \ \odot X^i) \circ \mathsf{swap2lsb}$. It follows that for all $k \in \mathbb{F}_2^4$, $\mathsf{swap2lsb} \circ M_k \circ \mathsf{swap2lsb}$ is the finite field multiplication by the power of X equal to the discrete logarithm of M_k base b. More precisely, let $\gamma\colon \mathbb{F}_2^4 \to \mathbb{F}_2^4$ be such that

$$\gamma(k) := (\mathsf{swap2lsb} \circ M_k \circ \mathsf{swap2lsb})(1).$$

Furthermore, let $\beta\colon \mathbb{F}_2^4 \to \mathbb{F}_2^4$ be given by

$$\beta(x) := U'_0(\mathsf{swap2lsb}(x)) \oplus \mathtt{C}.$$

Then U can be decomposed as follows:

$$U_k(x) = \alpha(k) \oplus \beta(\gamma(k) \odot \mathsf{swap2lsb}(x)) \oplus \mathtt{C}.$$

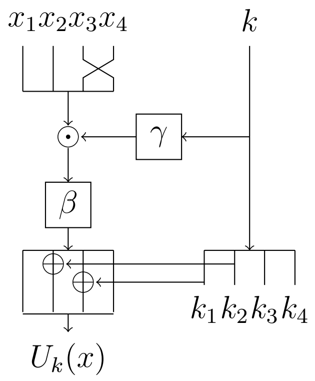
Note that α is affine such that $\alpha \oplus \mathtt{C}$ is linear:

$$\alpha(x_1, x_2, x_3, x_4) = (1, x_2 \oplus 1, x_1, 0).$$

The constant part of α cancels with the constant from β and the linear part can be merged with the outer linear encoding L. The graphical representation of the final decomposition of U and the codebooks of α, β, γ and swap2lsb are given in Figure 4.5.

4.2.3 Decomposition of T

The decomposition of T follows similar path as that of U, but with a couple of differences. all permutations T_i are linearly-equivalent, excluding T_0. The latter function stands out and does not follow any patterns. Furthermore, all $T_i, i \neq 0$ are related only by linear layer in the input. Let $N_1, \ldots, N_{\mathtt{F}}\colon \mathbb{F}_2^4 \to \mathbb{F}_2^4$



	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
α	C	C	C	C	8	8	8	8	E	E	E	E	A	A	A	A
β	0	E	B	4	2	3	F	8	A	7	1	9	5	C	6	D
γ	1	D	1	6	5	3	A	5	E	7	9	6	8	7	E	1
swap2lsb	0	2	1	3	4	6	5	7	8	A	9	B	C	E	D	F

FIGURE 4.5: The decomposition of $U_k(x)$.

be given by

$$N_k(x) \coloneqq T_k^{-1}(x) \circ T_1(x).$$

For any $k \neq 0$, $T_k = T_1 \circ N_k$ and each of N_i is affine. The look-up table of N is given in Table 4.4a.

In order to obtain linear mapping from affine, we detach the constant xor *before* the linear map. It could also be detached *after*, but detaching *before* allows to merge it with the outer linear layer L. Let $\delta\colon \mathbb{F}_2^4 \to \mathbb{F}_2^4$ be given by

$$\delta(k) \coloneqq \begin{cases} 0, & \text{if } k = 0, \\ N_k^{-1}(0), & \text{otherwise.} \end{cases}$$

It turns out that δ is a linear map:

$$\delta(k_1, k_2, k_3, k_4) = (0, k_1 \oplus k_3, 0, k_1 \oplus k_2 \oplus k_3).$$

Let $N'_1, \ldots, N'_{\mathtt{F}}\colon \mathbb{F}_2^4 \to \mathbb{F}_2^4$ be given by

$$N'_k(x) = N_k(x \oplus \delta(k)).$$

Then all N'_k are linear functions, i.e. $N'_k(0) = 0$ for all $k \in \mathbb{F}_2^4, k \neq 0$. The codebook of N' is given in Table 4.4b.

Consider N'_{2} (other choices are possible, but this one leads to simplest linear layers in the decomposition). It is linear-similar to the same field multiplication chosen in the decomposition of U: there exists $\eta \in \mathsf{GL}_4(\mathbb{F}_2)$ such that

$$N'_{\mathsf{2}} = \eta \circ (\cdot \ \odot X) \circ \eta^{-1}.$$

Such η is given by:

$$\eta(x_1, x_2, x_3, x_4) \coloneqq (x_1, x_2 \oplus x_4, x_3 \oplus x_2, x_4).$$

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
N_1	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
N_2	2	E	4	8	C	0	A	6	F	3	9	5	1	D	7	B
N_3	D	6	2	9	B	0	4	F	C	7	3	8	A	1	5	E
N_4	9	0	C	5	E	7	B	2	F	6	A	3	8	1	D	4
N_5	A	0	7	D	8	2	5	F	3	9	E	4	1	B	C	6
N_6	F	1	3	D	0	E	C	2	5	B	9	7	A	4	6	8
N_7	1	3	B	9	0	2	A	8	6	4	C	E	7	5	D	F
N_8	4	B	A	5	F	0	1	E	6	9	8	7	D	2	3	C
N_9	9	D	8	C	4	0	5	1	A	E	B	F	7	3	6	2
$N_{\mathtt{A}}$	0	D	4	9	A	7	E	3	5	8	1	C	F	2	B	6
$N_{\mathtt{B}}$	0	8	7	F	3	B	4	C	E	6	9	1	D	5	A	2
$N_{\mathtt{C}}$	5	6	D	E	0	3	8	B	A	9	2	1	F	C	7	4
$N_{\mathtt{D}}$	9	C	A	F	0	5	3	6	2	7	1	4	B	E	8	D
$N_{\mathtt{E}}$	7	0	E	9	F	8	6	1	B	C	2	5	3	4	A	D
$N_{\mathtt{F}}$	6	0	D	B	A	C	1	7	2	4	9	F	E	8	5	3

(A) N

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
N'_1	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
N'_2	0	C	6	A	E	2	8	4	D	1	B	7	3	F	5	9
N'_3	0	B	F	4	6	D	9	2	1	A	E	5	7	C	8	3
N'_4	0	9	5	C	7	E	2	B	6	F	3	A	1	8	4	D
N'_5	0	A	D	7	2	8	F	5	9	3	4	E	B	1	6	C
N'_6	0	E	C	2	F	1	3	D	A	4	6	8	5	B	9	7
N'_7	0	2	A	8	1	3	B	9	7	5	D	F	6	4	C	E
N'_8	0	F	E	1	B	4	5	A	2	D	C	3	9	6	7	8
N'_9	0	4	1	5	D	9	C	8	3	7	2	6	E	A	F	B
$N'_{\mathtt{A}}$	0	D	4	9	A	7	E	3	5	8	1	C	F	2	B	6
$N'_{\mathtt{B}}$	0	8	7	F	3	B	4	C	E	6	9	1	D	5	A	2
$N'_{\mathtt{C}}$	0	3	8	B	5	6	D	E	F	C	7	4	A	9	2	1
$N'_{\mathtt{D}}$	0	5	3	6	9	C	A	F	B	E	8	D	2	7	1	4
$N'_{\mathtt{E}}$	0	7	9	E	8	F	1	6	C	B	5	2	4	3	D	A
$N'_{\mathtt{F}}$	0	6	B	D	C	A	7	1	4	2	F	9	8	E	3	5

(B) N'.

TABLE 4.4: The keyed permutations ciphers N and N'.

Further, all N'_k turn out to be multiplications by a k-dependent constant in the finite field. Let $\varepsilon\colon \mathbb{F}_2^4 \to \mathbb{F}_2^4$ be given by

$$\varepsilon(k) := \begin{cases} 0, & \text{if } k = 0, \\ \eta^{-1} \circ N'_k \circ \eta(1), & \text{otherwise.} \end{cases}$$

Then ε turns out to be a bijection, and the following holds for $k \neq 0$:

$$\eta^{-1} \circ N'_k \circ \eta(x) = \varepsilon(k) \odot x,$$

ε seems to be a complicated permutation without any pattern. Note that when the inverse of T is computed, the field multiplication becomes the field division and thus, the output of ε is inverted. Denote the composition of the inversion in the finite field with ε by $1/\varepsilon$ (defining $1/0 = 0$). Surprisingly, it is a linear function. Furthermore, when composed with $\mathsf{swap2lsb}$ which appears here from the decomposition of U, it becomes a simple multiplication by a constant in the finite field:

$$1/\varepsilon \circ \mathsf{swap2lsb}(k) = k \odot (X^3 + X^2)$$

for all $k \in \mathbb{F}_2^4$. It follows that

$$\varepsilon(k) = 1/(\mathsf{swap2lsb}(x) \odot (X^3 + X^2)) = X \odot (1/\mathsf{swap2lsb}(k)),$$

where $X \odot (X^3 + X^2) = 1$ in the chosen finite field. The multiplication by constant can be transferred through the main multiplication in N' and merged with T_1.

We obtain that, when $k \neq 0$, T_k can be computed as:

$$T_k(x) = T_1 \circ \eta(\cdot \ \odot X) \left((1/\mathsf{swap2lsb}(k)) \odot \eta^{-1}(x \oplus \delta(k))\right).$$

The addition of $\delta(k)$, η^{-1} and $\mathsf{swap2lsb}$ can be merged with the outer linear layer L. η can be merged with T_1, and $\mathsf{swap2lsb}$ will cancel out when T is merged with U.

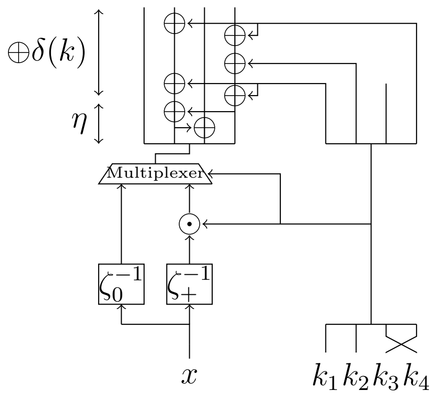
Let

$$\begin{aligned}\zeta_0 &\coloneqq T_0 \circ \eta,\\ \zeta_+ &\coloneqq T_1 \circ \eta \circ (\odot X).\end{aligned}$$

Then

$$T_k(x) = \begin{cases} \zeta_+ \left((1/\mathsf{swap2lsb}(k)) \odot \eta^{-1}(x \oplus \delta(k)) \right), & \text{if } k \neq 0,\\ \zeta_0 \left(\eta^{-1}(x \oplus \delta(k)) \right), & \text{if } k = 0. \end{cases}$$

The final decomposition of T^{-1} and the codebooks of $\delta, \eta, \mathsf{swap2lsb}, \zeta_0, \zeta_+$ and the field inverse $1/x$ are given are shown in Figure 4.6. It uses a multiplexer, which chooses its left input branch if the control branch (i.e., k) is equal to zero, and its right input branch otherwise. The only differences between T and T^{-1} are using the inverses of $\zeta_0, \zeta+$, removing the field inversion, and changing the position of the multiplexer. Note that η is an involution, and the addition of $\delta(k)$ is involution too.



	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
δ	0	0	5	5	1	1	4	4	5	5	0	0	4	4	1	1
η	0	5	2	7	6	3	4	1	8	D	A	F	E	B	C	9
swap2lsb	0	2	1	3	4	6	5	7	8	A	9	B	C	E	D	F
ζ_0	2	5	3	B	6	9	E	A	0	4	F	1	8	D	C	7
ζ_+	7	6	C	9	0	F	8	1	4	5	B	E	D	2	3	A
$\mathcal{I}$	0	1	C	8	6	F	4	E	3	D	B	A	2	9	7	5

FIGURE 4.6: The decomposition of $T_k^{-1}(x)$.

4.2.4 Full Decomposition

The full decomposition of π is obtained from the decompositions of mini-block ciphers T and U. First, let me describe the whitening linear layers.

Recall the whitening linear layer $L^\top$ L was described in the beginning of Section 4.2. Let $L_{in} \in \mathsf{GL}_8(\mathbb{F}_2)$ be given by

$$L_{in} \coloneqq l_{\mathsf{swap2lsb}} \circ l_\eta \circ l_\delta \circ L^\top,$$

where L was described in the beginning of Section 4.2, $l_\eta, l_\delta, l_{\mathsf{swap2lsb}} \in \mathsf{GL}_8(\mathbb{F}_2)$ are given by

$$\begin{aligned} l_\delta(x,k) &:= (x \oplus \delta(k), k), \\ l_\eta(x,k) &:= (\eta(x), \delta(k)), \\ l_{\mathsf{swap2lsb}}(x,k) &:= (x, \mathsf{swap2lsb}(k)). \end{aligned}$$

Let $L_{out} \in \mathsf{GL}_8(\mathbb{F}_2)$ be given by

$$L_{out} := (L^{-1})^\top \circ l_\alpha,$$

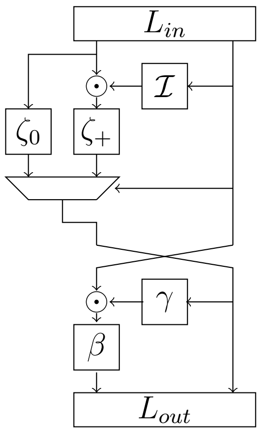
where $l_\alpha \in \mathsf{GL}_8(\mathbb{F}_2)$ is given by

$$l_\alpha(x,k) := (x \oplus \alpha(k), k),$$

The matrix representations of L_{in} and L_{out} are as follows:

$$L_{in} = \begin{bmatrix} 0&0&0&0&1&0&0&0 \\ 0&1&0&0&0&0&0&1 \\ 0&1&0&0&0&0&1&1 \\ 1&1&1&0&1&1&1&1 \\ 1&0&0&0&1&0&1&0 \\ 0&1&0&0&0&1&0&0 \\ 0&0&0&1&1&0&1&0 \\ 0&0&1&0&0&0&0&0 \end{bmatrix}, \quad L_{out} = \begin{bmatrix} 1&0&1&0&0&0&0&0 \\ 0&1&0&0&0&0&0&0 \\ 0&0&0&0&0&0&1&0 \\ 1&0&1&0&1&0&0&1 \\ 1&0&0&0&0&0&0&0 \\ 0&1&0&0&0&1&0&0 \\ 0&0&1&0&1&0&0&0 \\ 0&0&0&1&0&0&0&0 \end{bmatrix}. \qquad (4.1)$$

The final decomposition of π using the linear layers L_{in}, L_{out} and the non-linear components are given in Figure 4.7. An algorithmic representation of the decomposition is shown in Algorithm 4.1.



	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$\mathcal{I}$	0	1	c	8	6	f	4	e	3	d	b	a	2	9	7	5
ζ_0	2	5	3	B	6	9	E	A	0	4	F	1	8	D	C	7
ζ_+	7	6	C	9	0	F	8	1	4	5	B	E	D	2	3	A
$\mathcal{I}$	0	1	C	8	6	F	4	E	3	D	B	A	2	9	7	5
γ	1	D	1	6	5	3	A	5	E	7	9	6	8	7	E	1
β	0	E	B	4	2	3	F	8	A	7	1	9	5	C	6	D

FIGURE 4.7: The decomposition of π. The multiplexer chooses its left input branch if the control branch is equal to zero, and its right input branch otherwise. L_{in} and L_{out} are given in Equation 4.1.

Algorithm 4.1 Computing π: $v = \pi(u)$ using finite field multiplications.

```
1: $(x, k) \leftarrow L_{in}(u)$
2: if $r = 0$ then
3:     $x \leftarrow \zeta_0(x)$
4: else
5:     $x \leftarrow \zeta_+(x \odot \mathcal{I}(k))$                ▷ in $\mathbb{F}_{2^4} \simeq \mathbb{F}_2[X]/(X^4 + X^3 + 1)$
6: $(x, k) \leftarrow (k, x)$
7: $x \leftarrow \beta(x \odot \gamma(k))$                           ▷ in $\mathbb{F}_{2^4} \simeq \mathbb{F}_2[X]/(X^4 + X^3 + 1)$
8: $v \leftarrow L_{out}(x, k)$
9: return $v$
```

4.3 Decomposition based on Finite Field Logarithm

4.3.1 BelT Block Cipher and its S-Box

BelT is a block cipher from the Belarusian cryptographic standard [Bel11]. It uses an 8-bit S-Box $H\colon \mathbb{F}_2^8 \to \mathbb{F}_2^8$ which is given as a look-up table in the standard. The rationale behind this S-Box is not given in the standard, but instead in a separate rationale document by Agievich *et al.* [AGMK02].

Proposition 4.2 (The BelT S-Box Construction, [AGMK02] (translated))**.** *The look-up tables of the S-Box coordinate functions were chosen as different segments of length 255 of different linear recurrences defined by the irreducible polynomial $p(\lambda)$:*

$$p(\lambda) = \lambda^8 + \lambda^6 + \lambda^5 + \lambda^2 + 1.$$

Additionally, a zero element was inserted in a fixed position of each segment.

Agievich also explains in [AA04, AA05] that such a construction is equivalent to an *exponential* function in the finite field.

Definition 4.3. *For a primitive element $w \in \mathbb{F}_{2^n}$ let $x \mapsto w^{(x)}$ be the map from $\mathbb{F}_2^n$ to itself, obtained by raising w to the power given by the integer represented by $x \in \mathbb{F}_2^n$, and representing the result as an element of $\mathbb{F}_2^n$, where the polynomial defining the field should be clear from context.*

The exponential mapping can be turned into a permutation of $\mathbb{F}_2^n$ by letting it map 0 to 0. Let $w \in \mathbb{F}_{2^n}$ be a primitive element. Let exp_w be a permutation of $\mathbb{F}_2^n$ given by:

$$\mathsf{exp}_w(x) :=\mapsto \begin{cases} 0, & \text{if } x = 0, \\ w^{(x)}, & \text{otherwise.} \end{cases}$$

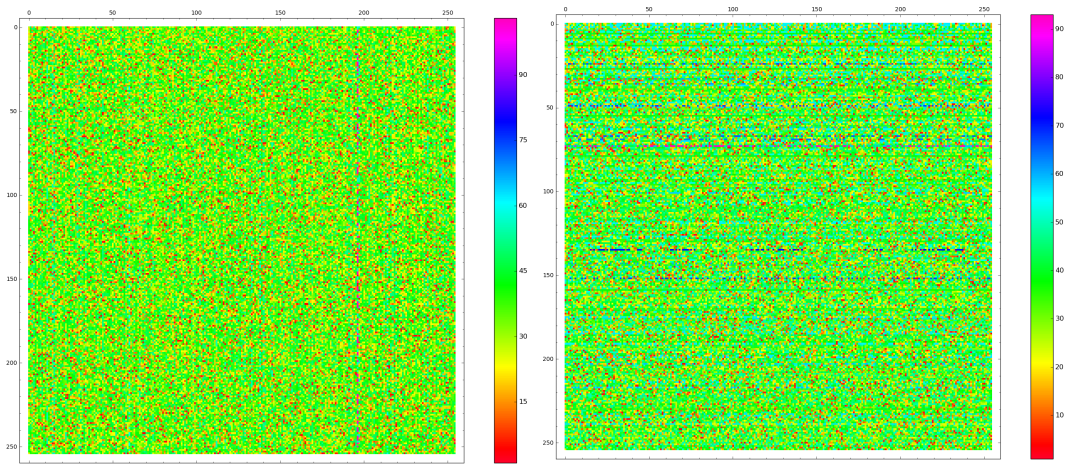
Let $\mathsf{log}_w\colon \mathbb{F}_2^n \to \mathbb{F}_2^n$ denote the functional inverse of exp_w:

$$\mathsf{log}_w := \mathsf{exp}_w^{-1}.$$

Remark 12. In the S-Box H used in BelT, the zero was inserted at $x = \mathtt{0A} \in \mathbb{F}_2^8$ instead of 0.

Figure 4.8 shows the Jackson Pollock representations of the column and row frequency tables of the LAT of H. In the row frequency table several rows stick out, similarly to the special columns in the column frequency table of the LAT of π. This similarity suggests that there might be a relation between H and the inverse of π.

Since this chapter is devoted to π, for a closer analysis of the S-Box used in BelT, I refer to our paper [PU17].



(A) $\mathsf{CF}(\mathsf{LAT}_H)$ (B) $\left(\mathsf{CF}(\mathsf{LAT}_H^\top)\right)^\top$

FIGURE 4.8: Jackson Pollock representation of the column and row frequency tables of the LAT of H.

4.3.2 Exponential Behaviour of π

An exponential function $x \mapsto w^x$ has the following property: for all $x, c \in \mathbb{F}_2^8$

$$w^{(x+c)} = w^{(x)} \odot w^{(c)}.$$

This property can be used to distinguish exponential permutations or functions close to them. However, the integer addition can be partially hidden by a whitening affine layer. Still, a strong property can be observed if the addition is approximated by XOR. Indeed, for a unit vector e_i of $\mathbb{F}_2^n$ and all $x \in \mathbb{F}_2^8$,

$$x \oplus e_i = \begin{cases} x \boxplus e_i, & \text{if } \langle x, e_i \rangle = 0, \\ x \boxminus e_i, & \text{if } \langle x, e_i \rangle = 1. \end{cases}$$

An advantage of this approximation is that the XOR with e_i after a whitening input linear map $L \in \mathsf{GL}_n(\mathbb{F}_2)$ maps back to the XOR with $L^{-1}(e_i)$ before the application of L. And indeed such behaviour can be observed in π! By an exhaustive search over the parameters, the following relations were found in π.

Observation 4.4. *Let* $c \in (12, 26, 24, 30)$. *For any* $i \in [1 \dots 4]$

$$\Pr_{x \in \mathbb{F}_2^8} \begin{bmatrix} \pi^{-1}(x \oplus c_i) = \pi^{-1}(x) \odot X^{2^{i-1}} & or \\ \pi^{-1}(x \oplus c_i) = \pi^{-1}(x) \oslash X^{2^{i-1}}. & \end{bmatrix} = 240/256, \qquad (4.2)$$

	.0	.1	.2	.3	.4	.5	.6	.7	.8	.9	.A	.B	.C	.D	.E	.F
0.	188	189	190	191	192	193	194	195	196	197	198	199	200	201	202	203
1.	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33
2.	143	144	141	142	139	140	137	138	151	152	149	150	147	148	145	146
3.	160	161	158	159	156	157	154	155	168	169	166	167	164	165	162	163
4.	216	215	214	213	220	219	218	217	208	207	206	205	212	211	210	209
5.	97	96	95	94	101	100	99	98	89	88	87	86	93	92	91	90
6.	48	47	50	49	44	43	46	45	40	39	42	41	36	35	38	37
7.	82	81	84	83	78	77	80	79	74	73	76	75	70	69	72	71
8.	172	171	174	173	176	175	178	177	180	179	182	181	184	183	186	185
9.	53	52	55	54	57	56	59	58	61	60	63	62	65	64	67	66
A.	127	126	125	124	123	122	121	120	135	134	133	132	131	130	129	128
B.	246	245	244	243	242	241	240	239	254	253	252	251	250	249	248	247
C.	232	233	230	231	236	237	234	235	224	225	222	223	228	229	226	227
D.	113	114	111	112	117	118	115	116	105	106	103	104	109	110	107	108
E.	221	238	255	0	153	170	187	204	85	102	119	136	17	34	51	68
F.	13	14	15	16	9	10	11	12	5	6	7	8	1	2	3	4

TABLE 4.5: The look-up table of $\mathsf{log}_X \circ \pi^{-1} \circ \alpha^{-1}$ (as integers).

where multiplication $\odot$ *and division* $\oslash$ *are performed in the finite field* $\mathbb{F}_{2^8} \simeq \mathbb{F}_2[X]/(X^8 + X^4 + X^3 + X^2 + 1)$ *and* X *defines a primitive element.*

This strong property suggests that the output side of π^{-1} is not masked by a random linear layer. Otherwise, the multiplication and the division by X^{2^i} would be masked and not triggered by the constant XOR in the input. Therefore, we assume that the output of π^{-1} is the output of an exponential function composed with some simple layer. The simple layer then can be analyzed separately as $\mathsf{log}_X \circ \pi^{-1}$.

4.3.3 Decomposing the Arithmetic Layer

Our hypothesis was that there is a linear whitening layer mapping all c_i to unit vectors. Equation 4.2 suggests that the unit vectors are consecutive powers of 2. Let $\alpha \in \mathsf{GL}_8(\mathbb{F}_2)$ be given by

$$\alpha := \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}^{-1}.$$

It is such that for all $i \in \{1, 2, 3, 4\}$, $\alpha(c_i)$ is the unit vector corresponding to 2^{i-1}. Preimages of the other four unit vectors were chosen randomly to complete the map. The look-up table of $\mathsf{log}_X \circ \pi^{-1} \circ \alpha^{-1}$ is given in Table 4.5.

	.0	.1	.2	.3	.4	.5	.6	.7	.8	.9	.A	.B	.C	.D	.E	.F
0.	188	189	190	191	192	193	194	195	196	197	198	199	200	201	202	203
1.	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33
2.	137	138	139	140	141	142	143	144	145	146	147	148	149	150	151	152
3.	154	155	156	157	158	159	160	161	162	163	164	165	166	167	168	169
4.	205	206	207	208	209	210	211	212	213	214	215	216	217	218	219	220
5.	86	87	88	89	90	91	92	93	94	95	96	97	98	99	100	101
6.	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50
7.	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84
8.	171	172	173	174	175	176	177	178	179	180	181	182	183	184	185	186
9.	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67
A.	120	121	122	123	124	125	126	127	128	129	130	131	132	133	134	135
B.	239	240	241	242	243	244	245	246	247	248	249	250	251	252	253	254
C.	222	223	224	225	226	227	228	229	230	231	232	233	234	235	236	237
D.	103	104	105	106	107	108	109	110	111	112	113	114	115	116	117	118
E.	17	34	51	68	85	102	119	136	153	170	187	204	221	238	255	0
F.	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

TABLE 4.6: The look-up table of $\mathsf{log}_X \circ \pi^{-1} \circ \beta^{-1}$ (as integers).

The rows of Table 4.5 are clearly structured. We observe that each row can be sorted by modifying the linear mapping α (except the zero value). Indeed, let $\beta \in \mathsf{GL}_8(\mathbb{F}_2)$ be given by

$$\beta := \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

Then the look-up table of $\mathsf{log}_X \circ \pi^{-1} \circ \beta^{-1}$ is the same as the look-up table of $\mathsf{log}_X \circ \pi^{-1} \circ \alpha^{-1}$ with sorted rows. It is shown in Table 4.6.

Furthermore, the rows can be reordered by applying a 4-bit nonlinear mapping to the left branch. Let q be a permutation of $\mathbb{F}_2^4$ given by its lookup table

$$\mathsf{LookupTable(q)} := (12, 2, 9, 10, 13, 6, 3, 5, 11, 4, 8, 15, 14, 7, 0, 1).$$

Let q_L be a permutation of $\mathbb{F}_2^8$ made by applying q to the left half of the input:

$$q_L(x, y) := (q(x), y)).$$

Then the look-up table of $\mathsf{log}_X \circ \pi^{-1} \circ \beta^{-1} \circ q_L^{-1}$ has a very simple structure. It is shown in Table 4.7. This structure has a simple arithmetic expression. As a result, an algorithmic decomposition of π^{-1} can be deduced. It is given in Algorithm 4.2.

	.0	.1	.2	.3	.4	.5	.6	.7	.8	.9	.A	.B	.C	.D	.E	.F
0.	17	34	51	68	85	102	119	136	153	170	187	204	221	238	255	0
1.	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
2.	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33
3.	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50
4.	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67
5.	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84
6.	86	87	88	89	90	91	92	93	94	95	96	97	98	99	100	101
7.	103	104	105	106	107	108	109	110	111	112	113	114	115	116	117	118
8.	120	121	122	123	124	125	126	127	128	129	130	131	132	133	134	135
9.	137	138	139	140	141	142	143	144	145	146	147	148	149	150	151	152
A.	154	155	156	157	158	159	160	161	162	163	164	165	166	167	168	169
B.	171	172	173	174	175	176	177	178	179	180	181	182	183	184	185	186
C.	188	189	190	191	192	193	194	195	196	197	198	199	200	201	202	203
D.	205	206	207	208	209	210	211	212	213	214	215	216	217	218	219	220
E.	222	223	224	225	226	227	228	229	230	231	232	233	234	235	236	237
F.	239	240	241	242	243	244	245	246	247	248	249	250	251	252	253	254

TABLE 4.7: The look-up table of $\mathsf{log}_X \circ \pi^{-1} \circ \beta^{-1} \circ q_L^{-1}$ (as integers).

Remark 13. It is also possible to define q such that it moves the row $(17, 34, \ldots, 255, 0)$ to the end of the table. This change results in similar expressions.

Algorithm 4.2 Computing the inverse of π: $y = \pi^{-1}(x)$.

```
1: $(l, r) \leftarrow \beta(x)$
2: $l \leftarrow q(l)$
3: **if** $l = 0$ **then**
4:     $z \leftarrow 17 \times ((r + 1) \mod 16)$          ▷ integer arithmetic
5: **else**
6:     $z \leftarrow 17 \times l + r - 16$          ▷ integer arithmetic
7: $y \leftarrow \mathsf{log}_X(z)$
8: **return** $y$
```

4.3.4 Obtaining a Decomposition of π

Let $\widehat{\pi}$ be the permutation of $\mathbb{F}_2^8$ given by

$$\widehat{\pi} := \mathsf{log}_X \circ \pi^{-1} \circ \beta^{-1} \circ q_L^{-1}.$$

It corresponds to the arithmetic part of the decomposition. Observe that it has a TU-decomposition as π has itself (see Section 4.2.1).

Observation 4.5. *There exist permutations of $\mathbb{F}_2^4$ $T_0, \ldots, T_F$ and $U_0, \ldots, U_F$ such that for all $l, r \in \mathbb{F}_2^4$*

$$\widehat{\pi}(l, r) = U_{T_l(r)}(l), T_l(r).$$

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
T_0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0
T_1	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0
T_2	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0	1
T_3	3	4	5	6	7	8	9	A	B	C	D	E	F	0	1	2
T_4	4	5	6	7	8	9	A	B	C	D	E	F	0	1	2	3
T_5	5	6	7	8	9	A	B	C	D	E	F	0	1	2	3	4
T_6	6	7	8	9	A	B	C	D	E	F	0	1	2	3	4	5
T_7	7	8	9	A	B	C	D	E	F	0	1	2	3	4	5	6
T_8	8	9	A	B	C	D	E	F	0	1	2	3	4	5	6	7
T_9	9	A	B	C	D	E	F	0	1	2	3	4	5	6	7	8
T_A	A	B	C	D	E	F	0	1	2	3	4	5	6	7	8	9
T_B	B	C	D	E	F	0	1	2	3	4	5	6	7	8	9	A
T_C	C	D	E	F	0	1	2	3	4	5	6	7	8	9	A	B
T_D	D	E	F	0	1	2	3	4	5	6	7	8	9	A	B	C
T_E	E	F	0	1	2	3	4	5	6	7	8	9	A	B	C	D
T_F	F	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E

(A) T.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
U_0	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
U_1	1	0	2	3	4	5	6	7	8	9	A	B	C	D	E	F
U_2	2	0	1	3	4	5	6	7	8	9	A	B	C	D	E	F
U_3	3	0	1	2	4	5	6	7	8	9	A	B	C	D	E	F
U_4	4	0	1	2	3	5	6	7	8	9	A	B	C	D	E	F
U_5	5	0	1	2	3	4	6	7	8	9	A	B	C	D	E	F
U_6	6	0	1	2	3	4	5	7	8	9	A	B	C	D	E	F
U_7	7	0	1	2	3	4	5	6	8	9	A	B	C	D	E	F
U_8	8	0	1	2	3	4	5	6	7	9	A	B	C	D	E	F
U_9	9	0	1	2	3	4	5	6	7	8	A	B	C	D	E	F
U_A	A	0	1	2	3	4	5	6	7	8	9	B	C	D	E	F
U_B	B	0	1	2	3	4	5	6	7	8	9	A	C	D	E	F
U_C	C	0	1	2	3	4	5	6	7	8	9	A	B	D	E	F
U_D	D	0	1	2	3	4	5	6	7	8	9	A	B	C	E	F
U_E	E	0	1	2	3	4	5	6	7	8	9	A	B	C	D	F
U_F	F	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E

(B) U.

TABLE 4.8: The mini-block ciphers T, U decomposing $\widehat{\pi}$.

Such T, U are given in Table 4.8. They can also be expressed arithmetically:

$$T_k(x) = \begin{cases} x + k, & \textit{if } k \neq 0, \\ x + k + 1, & \textit{otherwise}, \end{cases} \qquad U_k(x) = \begin{cases} ((x - k - 1) \mod 15) + k + 1, & \textit{if } x \neq 0, \\ k, & \textit{otherwise}. \end{cases}$$

T and U can be inverted separately. By further using the finite field logarithm and inverses of β and q_L, the logarithmic decomposition of π is obtained. The corresponding algorithm is given in Algorithm 4.3 and graphical representation is given in Figure 4.9.

Algorithm 4.3 Computing the S-Box $y = \pi(x)$ using the logarithmic decomposition.

```
$(l, r) \leftarrow \mathsf{log}_X(x)$
$l \leftarrow l - r$
**if** $l = 0$ **then**
    $r \leftarrow r - 1$
**else**
    $l \leftarrow (l + r - 1) \mod 15 + 1$
$r \leftarrow r - l$
$l \leftarrow q^{-1}(l)$
$y \leftarrow \beta^{-1}(l||r)$
**return** $y$
```

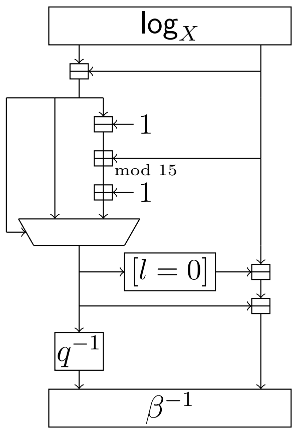


FIGURE 4.9: The logarithmic decomposition of π.

4.4 Discussion and Conclusions

This chapter presented two different decompositions of π, the S-Box used in Russian cryptographic standards. The decompositions show that the S-Box has a strong structure related to the finite field arithmetic. The reasons behind such structure are unclear. It is not known whether a trapdoor can be hidden in such S-Box, such that the block cipher or the hash function using it becomes weaker. A more likely reason is the possibility of having better hardware implementation than for a random S-Box.

The first decomposition, presented in Section 4.2, is based on finite field multiplications forming a Feistel-like 2-round network, several 4-bit S-Boxes and two 8-bit whitening linear layers. The bijectivity is preserved differently in each round. In the first round, a multiplexer is used such that multiplication by 0 is not performed. In the second round, the multiplication is performed by a non-bijective function of the left branch, which is never equal to 0. Such structure was never used before in cryptography.

The second decomposition, presented in Section 4.3, is based on the finite field logarithm, one 4-bit S-Box, one 8-bit whitening linear layer and a simple but strange arithmetic layer, given in Table 4.7. It is almost the identity mapping, except that multiples of 17 are cut out and placed in the beginning, together with 0. This simplicity suggests that indeed π is very closely related to the finite field logarithm. Nevertheless, we could not find a meaningful arithmetic expression or simple circuit for computing it.

The second decomposition is “lighter” then the first one, because it contains less information-heavy elements, as the large part of the complexity is taken away by the finite field logarithm. The relation between the two decompositions is also not clear. The first decomposition can be seen as an implementation of the finite field logarithm using operations in the smaller field, $\mathbb{F}_{2^4}$. This is similar to the Canright’s implementation of the AES S-Box [Can05], the finite field inversion. Note however, that the finite field logarithm itself does not have a TU-decomposition. It is the extra part of π that activates this multiset property. There is also a possibility that both decompositions are a side effect of another algebraic construction.

This chapter shows usefulness of the following S-Box reverse-engineering tools:

1. Jackson-Pollock representation of the LAT for visual patters.
2. TU-decomposition as initial step and high-level decomposition.
3. Affine-equivalence algorithms.

It also shows different methods of simplifying complicated structures and random-looking components. The ways of reasoning employed to obtained the decompositions of π are proved to be useful again in Chapter 5, where it is shown that *mathematical* structures can also be decomposed using S-Box reverse-engineering methods.

Chapter 5

Decomposition of the 6-bit APN Permutation

Almost Perfect Non-linear (APN) permutations are bijective S-Boxes with optimal resistance against differential cryptanalysis. The existence of APN permutations in even dimensions was a long-standing problem until Dillon *et al.* discovered a 6-bit APN permutation [BDMW10, Dil09] and presented it in 2009. Since then, no new APN permutation with even number of bits were discovered, neither their existence was disproved. This question remains a big open problem in the field of Boolean functions. In this chapter, I describe an application of S-Box reverse-engineering methods to the only known APN permutation in even dimension. As a result, a simple algebraic structure of the Dillon's permutation is discovered, which we call the *Butterfly* structure. The result leads to much simpler representations of the permutation, which are further generalized to higher dimensions. Though, no new APN permutations are found.

This chapter highlights the usefulness of the S-Box reverse-engineering methods for obtaining decompositions of S-Boxes.

5.1 Introduction

S-Boxes are used to provide non-linearity in SPN-based block ciphers. They provide basic resistance against linear and differential cryptanalysis, and the rest of the structure ensures that many S-Boxes are activated in a linear or differential trail. The resistance of an S-Box can be quantified. The lower are the *linearity* and the *differential uniformity* of an S-Box, the more resistant it is. Everything else being equal, a stronger S-Box allows to use less rounds in the block cipher using it for the same security level.

The differential uniformity is always even and at least equal to 2. When this bound is achieved, the S-Box is called *Almost Perfect Non-linear (APN)*. The finite field cube function is APN in all field dimensions [Nyb93]. However, it is a permutation only in odd dimensions. This is a problem, since in most cases (e.g. an SPN block cipher) the S-Boxes are required to be bijective. For efficiency reasons, even-dimensional S-Boxes are preferable, especially powers of 2. And this is exactly the case, where the existence of APN functions is not established: bijective S-Boxes in even dimensions with differential uniformity 2, i.e. APN permutations of $\mathbb{F}_2^n$ for n even. For $n = 4$ there exist no APN permutations of $\mathbb{F}_2^n$. For $n = 6$ this question was a long standing problem until Dillon *et al.* presented a 6-bit APN permutation [BDMW10, Dil09] in 2009. Since then, no answers were obtained for even $n \geq 8$, despite many attempts [Göl15, TCT15]. This remains a big open problem in the field of Boolean functions.

The 6-bit APN permutation is found by a computer search, by transforming the 6-bit APN function, called the Kim mapping $\kappa \colon \mathbb{F}_{2^6} \to \mathbb{F}_{2^6}$:

$$\kappa(x) := vx^{24} + x^{10} + x^3,$$

where v is a primitive element of $\mathbb{F}_{2^6}$. Even though the Kim mapping is a trinomial function, the resulting APN permutation is an object without clear structure. For example, its polynomial form contains 52 monomials.

Using the methods of S-Box reverse-engineering described in previous chapters (developed in [BPU16]), I and my coauthors managed to find a simple algebraic structure of the Dillon's APN permutation. We call this structure a "Butterfly" because of its graphical representation the way it changes by particular transformations. The decomposition is established in Theorem 5.9, restated here:

Main theorem (A Family of 6-bit APN Permutations). The 6-bit permutation described by Dillon *et al.* in [BDMW10] is affine equivalent to any involution built using the structure described in Figure 5.1, where $\odot$ denotes

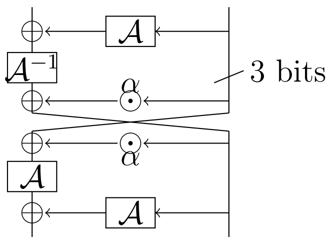


FIGURE 5.1: A family of APN permutations affine equivalent to the Dillon's permutation.

multiplication in the finite field $\mathbb{F}_{2^3}$, $\alpha \neq 0$ is such that $\mathrm{tr}(\alpha) = 0$ and $\mathcal{A}$ denotes any 3-bit APN permutation.

5.1.1 Notations

For any $f\colon \mathbb{F}_2^n \to \mathbb{F}_2^n$ let $P_f\colon \mathbb{F}_2^{2n} \to \mathbb{F}_2^{2n}$ be the parallel application of f given by

$$P_f(x, y) = (f(x), f(y)).$$

For any $a, b \in \mathbb{F}_2^n$ let $X_f\colon \mathbb{F}_2^{2n} \to \mathbb{F}_2^{2n}$ be parallel xor with constants a, b:

$$X_f(x, y) = (x \oplus a, y \oplus b).$$

The finite field trace function is denoted by $\mathsf{Tr}\colon \mathbb{F}_{2^n} \to \mathbb{F}_2$, it is given by

$$\mathsf{Tr}(x) \coloneqq \sum_{e=0}^{n-1} x^e.$$

5.1.2 Outline

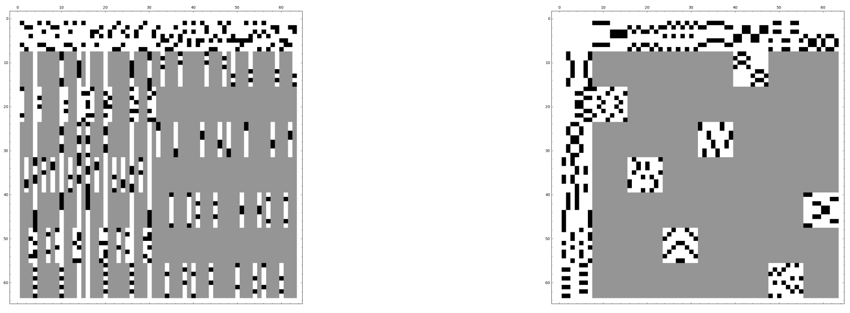
Section 5.2 explains the decomposition process of the APN permutation. In Section 5.3 I describe new properties of the APN permutation that follow from the discovered structure. Section 5.4 studies the flexibility of the structure, i.e. how can we modify the structure while preserving the APN property? In Section 5.5 I show new relations between the APN permutation, the Kim mapping, monomial functions and 3-round Feistel Network structure. Finally, I briefly conclude the chapter in Section 5.6.

5.2 Decomposition of the 6-bit APN permutation

Let S_0 be the APN permutation of $\mathbb{F}_2^6$ proposed in [BDMW10]. The look-up table of S_0 is given in Table 5.1.

	.0	.1	.2	.3	.4	.5	.6	.7	.8	.9	.A	.B	.C	.D	.E	.F
0.	00	36	30	0D	0F	12	35	23	19	3F	2D	34	03	14	29	21
1.	3B	24	02	22	0A	08	39	25	3C	13	2A	0E	32	1A	3A	18
2.	27	1B	15	11	10	1D	01	3E	2F	28	33	38	07	2B	2C	26
3.	1F	0B	04	1C	3D	2E	05	31	09	06	17	20	1E	0C	37	16

TABLE 5.1: The Dillon APN permutation S_0 in hexadecimal (e.g. $S_0(\texttt{10}) = \texttt{3B}$).



(A) LAT_{S_0} (B) $\mathsf{LAT}_{\eta^\top \circ S_0}$

FIGURE 5.2: The absolute LAT of S_0 and $\eta^\top \circ S_0$. White, grey and black pixels correspond to 0, 8 and 16 respectively.

5.2.1 TU-decomposition

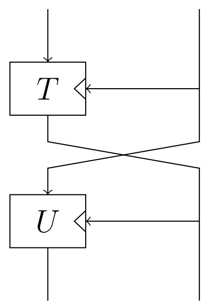
The first step to the decomposition of S_0 completely resembles the reverse-engineering of the GOST S-Box. We start by looking at the visualization of the LAT of S_0 (see Figure 5.2a). In the same way, there are 7 special columns and their indices (i.e., the linear masks defining the coordinates of S_0) together with 00 form a 3-dimensional linear subspace $V \subseteq \mathbb{F}_2^6$:

$$V = \{\texttt{00}, \texttt{04}, \texttt{0A}, \texttt{0E}, \texttt{10}, \texttt{14}, \texttt{1A}, \texttt{1E}\} = \operatorname{span}(\texttt{04}, \texttt{0A}, \texttt{210}).$$

Following the same path as in Chapter 4, we compose S_0 with a linear map to "move" the special lines to the left. Let $\eta \in \mathsf{GL}_6(\mathbb{F}_2)$ be such that

$$\eta(\texttt{01}) := \texttt{04}, \eta(\texttt{02}) := \texttt{0A}, \eta(\texttt{04}) := \texttt{10},$$
$$\eta(\texttt{08}) := \texttt{01}, \eta(\texttt{10}) := \texttt{02}, \eta(\texttt{20}) := \texttt{20},$$

where the first 3 values correspond to the basis of V and the last 3 values were chosen arbitrarily to complete the map to bijection. As a result, the special columns are aligned to the left in the LAT of $\eta^\top \circ S_0$, see Figure 5.2b. Furthermore, a clear square-based structure emerged in the LAT. Note that in the case of the GOST S-Box, the lines of the LAT had also to be reordered, and the inverse of the same linear mapping accidentally could be used to do it. Here, only the output of the S-Box has to be composed with a linear map. Indeed, already a white 8×8 square is observed in the top-left corner. In the decomposition of the GOST S-Box, it suggested multiset properties which led to the high-level TU-decomposition. The same decomposition is obtained for



(A) TU-decomposition

	0	1	2	3	4	5	6	7
T_0	0	6	4	7	3	1	5	2
T_1	7	5	1	6	4	2	0	3
T_2	4	3	2	0	5	6	1	7
T_3	3	5	2	1	4	6	7	0
T_4	1	2	0	6	4	3	7	5
T_5	6	5	2	4	7	0	1	3
T_6	5	2	6	4	0	3	1	7
T_7	2	0	1	6	5	3	4	7

(B) T

	0	1	2	3	4	5	6	7
U_0	0	3	6	4	2	7	1	5
U_1	7	4	0	2	3	6	1	5
U_2	1	4	2	6	3	0	5	7
U_3	7	2	5	1	3	0	4	6
U_4	7	3	4	1	0	2	6	5
U_5	3	7	1	4	2	0	5	6
U_6	1	3	7	4	6	2	5	0
U_7	4	6	3	0	5	1	7	2

(C) U

FIGURE 5.3: The TU-decomposition of $\eta^\top \circ S_0$.

S_0 as well.

Proposition 5.1 (TU-decomposition of $\eta^\top \circ S_0$)**.** *There exist 16 permutations $T_0, \dots, T_7, U_0, \dots, U_7$ of $\mathbb{F}_2^3$ such that for all $l, r \in \mathbb{F}_2^3$*

$$\eta^\top \times S_0(l,r) = \big(U_{T_r(l)}(r), T_r(l)\big)\,.$$

The codebooks of the keyed permutations T and U are given in Figure 5.3.

The algebraic degrees of the functions $(x,k) \mapsto T_k(x)$ and $(x,k) \mapsto T_k^{-1}(x)$ are 3 and 2 respectively. For U, the respective degrees are 2 and 3. This observation suggests that U and T^{-1} should be easier to decompose and that these keyed permutations may be related. We applied the linear equivalence algorithm from [BCBP03] to the mappings $(x,k) \mapsto (T_k^{-1}(x), k)$ and $(x,k) \mapsto (U_k(x), k)$ and found that they are linearly related.

Proposition 5.2 (Linear Equivalence of T^{-1} and U)**.** *Let T, U be permutations of $\mathbb{F}_2^6$ given by*

$$T(x,k) := (T_k(x), k), U(x,k) := (U_k(x), k).$$

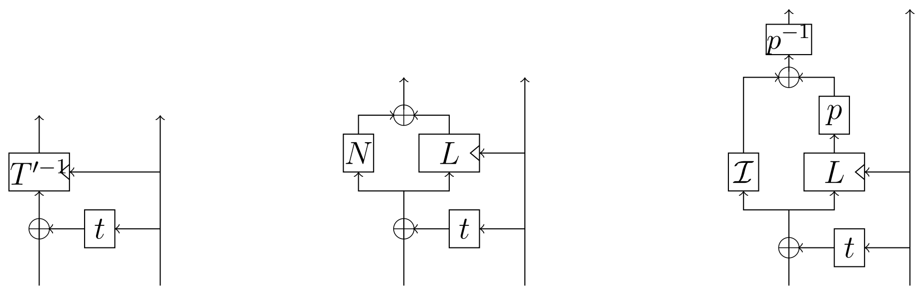
Let $M_U, M'_U \in \mathsf{GL}_8(\mathbb{F}_2)$ be given by

$$M_U := \begin{bmatrix} 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix}, \quad M'_U := \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix}.$$

Then

$$U = M'_U \circ T^{-1} \circ M_U.$$

Since T^{-1} and U are linear-equivalent, it is enough to decompose one of them. Since the linear layer $\eta^\top$ was applied at the output side, it could "obfuscate" some algebraic structure in U. Therefore, we choose to decompose T^{-1}. Afterwards, the two decompositions will be joined and the linear layer in between will be simplified.



(A) Detaching a linear Feistel round. (B) Splitting T'^{-1} into N and L. (C) Simplifying N into $\mathcal{I}$ and linear functions.

FIGURE 5.4: Simplifying the keyed permutation T'^{-1}.

	0	1	2	3	4	5	6	7	Interpolation polynomial
$T_0'^{-1}$	0	5	7	4	2	6	1	3	$w^3x^6 + w^1x^5 + w^3x^4 + w^6x^3 + w^1x^2 + \ 0\ x$
$T_1'^{-1}$	0	3	1	4	7	5	2	6	$w^3x^6 + w^1x^5 + w^7x^4 + w^6x^3 + w^2x^2 + w^1x$
$T_2'^{-1}$	0	4	5	7	3	6	2	1	$w^3x^6 + w^1x^5 + \ 0\ x^4 + w^6x^3 + \ 0\ x^2 + \ 0\ x$
$T_3'^{-1}$	0	2	3	7	6	5	1	4	$w^3x^6 + w^1x^5 + w^1x^4 + w^6x^3 + w^4x^2 + w^1x$
$T_4'^{-1}$	0	2	5	1	7	4	6	3	$w^3x^6 + w^1x^5 + w^3x^4 + w^6x^3 + \ 0\ x^2 + w^6x$
$T_5'^{-1}$	0	4	3	1	2	7	5	6	$w^3x^6 + w^1x^5 + w^7x^4 + w^6x^3 + w^4x^2 + w^5x$
$T_6'^{-1}$	0	3	7	2	6	4	5	1	$w^3x^6 + w^1x^5 + \ 0\ x^4 + w^6x^3 + w^1x^2 + w^6x$
$T_7'^{-1}$	0	5	1	2	3	7	6	4	$w^3x^6 + w^1x^5 + w^1x^4 + w^6x^3 + w^2x^2 + w^5x$

TABLE 5.2: The codebook and polynomial representation of each $T_k'^{-1}$.

5.2.2 Decomposition of T^{-1}

Step 1. Similarly to the decomposition process of the GOST S-Box, we "detach" a Feistel function from T in order to obtain a new keyed permutation T' such that $T'_k(0) = 0$ for all $k \in \mathbb{F}_2^3$. Detaching such function at the input and at the output of T^{-1} leads to the Feistel functions t and $\hat{t}$ respectively, a permutations of $\mathbb{F}_2^3$ given by

$$t(k) := T_k(0), \quad \hat{t}(k) := T_k^{-1}(0).$$

t is linear and $\hat{t}$ has algebraic degree 2. We conclude that detaching t at the input of T^{-1} leads to simpler decomposition. Let $T'_0, \dots, T'_7$ be permutations of $\mathbb{F}_2^3$ given by, for any $k \in \mathbb{F}_2^3$,

$$T'_k(x) := T_k(x) \oplus t(k).$$

The result of this step is shown in Figure 5.4a.

Step 2. Due to the algebraic nature of the S-Box, we may expect an algebraic structure. We consider the field

$$\mathbb{F}_{2^3} \simeq \mathbb{F}_2[w]/(w^3 + w + 1),$$

The primitive element w will be used to express field elements. As was noted in Chapter 2, from now on the isomorphism between $\mathbb{F}_2^3$ and $\mathbb{F}_{2^3}$ will be implicit.

We now apply Lagrange interpolation method to each permutation $T_k'^{-1}$ to obtain its polynomial representation, see Table 5.2. It is clear that the coefficients of the nonlinear part (i.e. the coefficients of x^3, x^5, x^6) are independent of k for each $T_k'^{-1}$. This leads to the following proposition, illustrated in Figure 5.4b.

Proposition 5.3. *Let N be a permutation of $\mathbb{F}_2^3$ given by its polynomial representation in $\mathbb{F}_{2^3}$: $N(x) := w^3x^6 + w^1x^5 + w^6x^3$. Let $L_0, \dots, L_7$ be permutations of $\mathbb{F}_2^3$ given by*

$$L_k(x) := T_k'^{-1}(x) \oplus N(x).$$

Then for any $k \in \mathbb{F}_2^3$, L_k is linear.

Step 3. We continue by simplifying the nonlinear function N. We do so by composing a linear bijection of our choice with N and L_k. Its inverse will then merge to the outer linear layer. It turns out that N can be transformed into the finite field inverse function, which we denote $\mathcal{I}\colon \mathbb{F}_2^3 \to \mathbb{F}_2^3$. Its polynomial representation is

$$\mathcal{I}(x) = x^6.$$

Proposition 5.4. *Let $p \in \mathsf{GL}_3(\mathbb{F}_2)$ be given by its $\mathbb{F}_{2^3}$-polynomial*

$$p(x) := w^2x^4 + x^2 + x.$$

Then

$$p(N(x)) = x^6 = \mathcal{I}(x)$$

is the inversion in the finite field, and also for all $k \in \mathbb{F}_2^3$

$$p(L_k(x)) = l_2(k+w)x^2 + l_4(k+w)x^4,$$

where $l_2, l_4 \in \mathsf{GL}_3(\mathbb{F}_2)$ are given by

$$l_2(x) := wx^4 + w^2x^2 + x,$$
$$l_4(x) := x^4 + w^4x^2 + wx.$$

Note that $k+w$ was used because $L_2 = 0$ (w corresponds to 2), so that l_2 and l_4 are linear. The composition of p with N and L_k is illustrated in Figure 5.4c.

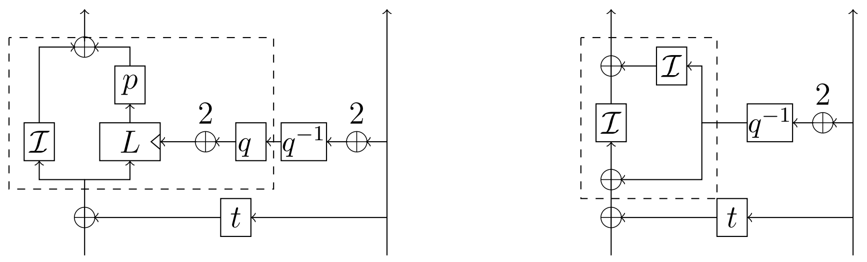
Step 4. The next step is to simplify the linear bijections l_2 and l_4. By composing l_2 with an arbitrary linear bijection $q \in \mathsf{GL}_3(\mathbb{F}_2)$, an arbitrary linear bijection may be obtained. However, l_4 has to be composed with the same linear map q. Therefore, q should simplify both l_2 and l_4. By exhaustive search of q, we found q such that $l_2(q(x)) = x^4$, $l_4(q(x)) = x^2$. These expressions lead to a simple expression for $p \circ L_k$, as it is shown by the following proposition.

Proposition 5.5. *Let $q \in \mathsf{GL}_3(\mathbb{F}_2)$ be given by its $\mathbb{F}_{2^3}$-polynomial:*

$$q(x) := w^3x^4 + w^5x^2 + w^3x.$$

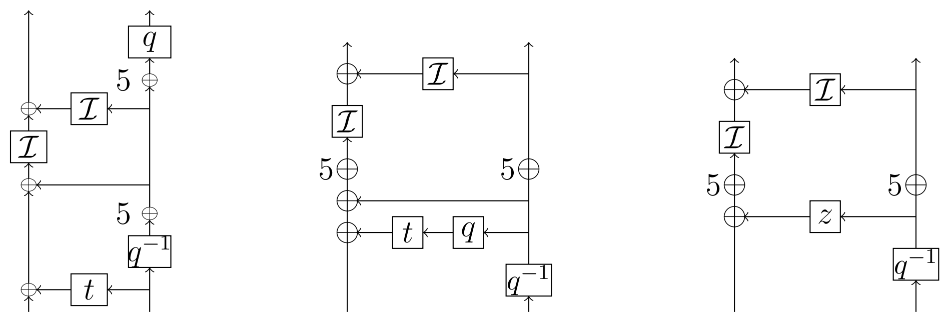
Then, for all $x \in \mathbb{F}_{2^3}$,

$$p(T_k'^{-1}(x)) = x^6 + x^2k'^4 + x^4k'^2 = (x+k')^6 + k'^6, \tag{5.1}$$



(A) Using $k' = q(k) \oplus 2$. (B) Using Equation 5.1.

FIGURE 5.5: Simplifying $p \circ L$ (and $p^{-1} \circ T'^{-1}$). The dashed area corresponds to the application of Equation 5.1.



(A) (B) (C)

FIGURE 5.6: Merging affine mappings in the decomposition of T^{-1}.

where $k' = q^{-1}(k + w)$.

Proof. Recall that

$$p(T_k'^{-1}(x)) = x^6 + l_2(k + w)x^2 + l_4(k + w)x^4,$$

where

$$l_2(k + w) = l_2(q(q^{-1}(k + w))) = (q^{-1}(k + w))^4 = k'^4.$$

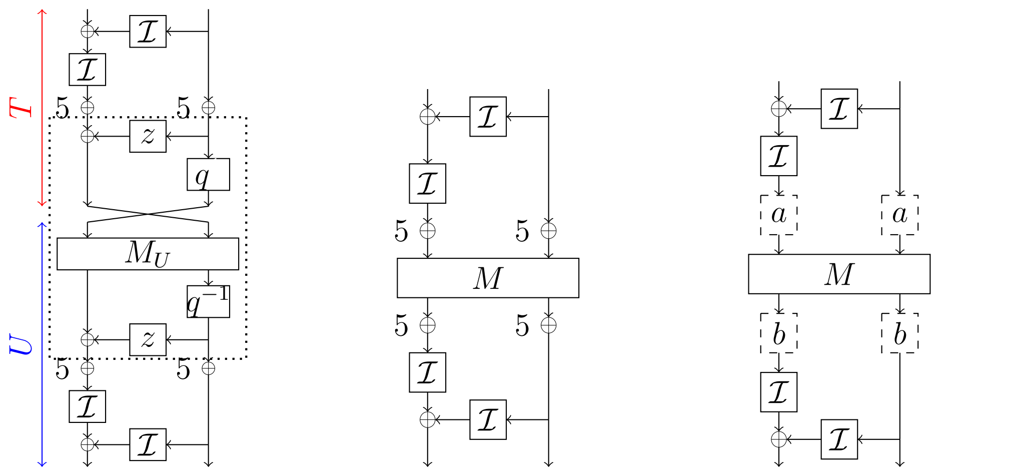
Similarly, $l_2(k + w) = k'^2$. □

The graphical representation of the effect of this proposition is shown in Figure 5.5.

Step 5. The final step is to simplify the affine layers. The application of q' and the addition of w can be moved from the Feistel function into the input and output linear layers of T'^{-1}. The output affine layer of T'^{-1} can be omitted, since it corresponds to an S-Box affine-equivalent to S_0. The simplification process is illustrated in Figure 5.6.

Proposition 5.6 (Decomposition of T'^{-1})**.** *For all* $x, k \in \mathbb{F}_{2^3}$,

$$T_k^{-1}(x) = p^{-1}(k'^6 + (x + k'')^6),$$



(A) Joining the decompositions of T and U.
(B) Merging linear layers.
(C) Allowed transformations.

FIGURE 5.7: Simplifying the middle affine layer. The linear mappings in the dotted area in Figure 5.7a form the linear layer M.

where

$$k' = q^{-1}(k) \oplus w^6,$$
$$k'' = z(q^{-1}(k)) \oplus w^6,$$
$$z \in \mathsf{GL}_3(\mathbb{F}_2), z(y) := t(q(y)) + y. \quad q \in \mathsf{GL}_3(\mathbb{F}_2), q(y) := w^3y^4 + w^5y^2 + w^3y.$$

Proof. Recall that

$$T_k^{-1}(x) = T_k'^{-1}(x \oplus t(k)).$$

Together with Proposition 5.5, it is enough to have

$$k'' = k' \oplus t(k).$$

This is true, because

$$k' = q^{-1}(k \oplus w) = q^{-1}(k) \oplus w^6,$$
$$k'' = z(q^{-1}(k)) \oplus w^6 = t(k) \oplus q^{-1}k \oplus w^6.$$

□

5.2.3 Combining T and U

We can now obtain a decomposition of full S_0. Recall that

$$U = M_U' \circ T^{-1} \circ M_U.$$

The decomposition of T follows from the decomposition of T^{-1} by inverting q in the middle. We omit all outer affine maps and merge the inner linear maps into one linear transformation of $\mathbb{F}_2^6$. The resulting structure is given in Figure 5.7a, Figure 5.7b.

Proposition 5.7. *The APN permutation S_0 is affine-equivalent to the structure given in Figure 5.7b. Formally, let V be the permutation of $\mathbb{F}_2^3 \times \mathbb{F}_2^3$ given by*

$$V(x, k) = (\mathcal{I}(x \oplus \mathcal{I}(k)), k),$$

and let $M \in \mathsf{GL}_6(\mathbb{F}_2)$ be given by its matrix:

$$M = \begin{bmatrix} 1 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 \end{bmatrix}.$$

Then the permutation S_M of $\mathbb{F}_2^6$ defined as $S_M := V^{-1} \circ X_{5,5} \circ M \circ X_{5,5} V$ is APN and is affine-equivalent to S_0.

Proof. The proposition follows form the TU-decomposition of S_0, decomposition of T^{-1} and linear relation between T^{-1} and U. Furthermore, M is obtained through composition including linear maps q, q^{-1}, z and M_U. □

We further studied how the middle affine layer can be changed while preserving the APN property. It turns out that, for affine layers $a, b \colon \mathbb{F}_2^3 \to \mathbb{F}_2^3$, applying a to both branches before M and applying b to both branches after M always leads to an APN permutation affine-equivalent to S_0. This is formally stated and proved in Section 5.4.1.

We observe that removing the constant addition (i.e. $X_{5,5}$) from the structure does not break the APN property. Therefore, it is left to simplify the linear map M. By exhaustive search over linear maps $a, b \in \mathsf{GL}_3(\mathbb{F}_2)$ we found that M can be transformed into a 2×2 matrix over $\mathbb{F}_{2^3}$.

Proposition 5.8. *Let V be defined as in Proposition 5.7. Let $M' \in \mathsf{GL}_2(\mathbb{F}_{2^3})$ be given by:*

$$M' := \begin{bmatrix} w, w^6, \\ 1, w \end{bmatrix}.$$

Then the permutation $S_{M'}$ of $\mathbb{F}_2^6$ defined by

$$S_{M'} := V^{-1} \circ M \circ V$$

is APN.

Proof. Let $a, b \in \mathsf{GL}_3(\mathbb{F}_2)$ be given by $\mathbb{F}_{2^3}$-polynomials:

$$a(x) = wx^4 + wx^2 + w^2x,$$
$$b(x) = wx^4 + w^3x^2 + wx,$$

Then,

$$M' = P_b \circ M \circ P_a.$$

According to Theorem 5.10, $S_{M'}$ is APN. □

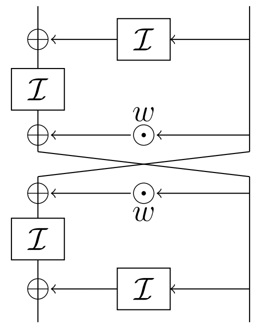


FIGURE 5.8: The APN involution $S_{\mathcal{I}}$.

Observe that M' happens to be an involution, making $S_{M'}$ an involution too due to its symmetric structure. Furthermore, M' can be decomposed as a two-round Feistel Network with multiplication by w as the Feistel function. The following theorem finalizes our decomposition, which is illustrated in Figure 5.8.

Theorem 5.9 (Decomposition of S_0). *Let W, swap be permutations of $\mathbb{F}_2^3 \times \mathbb{F}_2^3$ given by their bivariate $\mathbb{F}_{2^3}$-polynomials:*

$$W(x,k) = ((x+wk)^6 + k^6, k), \quad \mathsf{swap}(x,k) = (k,x).$$

Then the permutation $S_{\mathcal{I}}$ of $\mathbb{F}_2^6$ given by

$$S_{\mathcal{I}} \coloneqq W \circ \mathsf{swap} \circ W^{-1}$$

is an APN involution and is affine-equivalent to S_0:

$$S_0(x) = B(S_{\mathcal{I}}(A(x) \oplus \mathsf{9}) \oplus \mathsf{4},$$

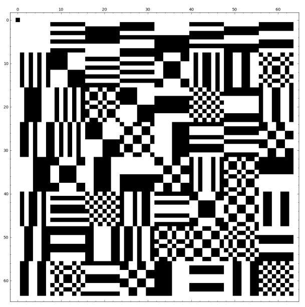
where $A, B \in \mathsf{GL}_6(\mathbb{F}_2)$ are given by

$$A = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 \end{bmatrix}, \quad B = \begin{bmatrix} 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 \end{bmatrix}.$$

5.3 Properties of the Decomposition

5.3.1 Cryptographic Properties

The decomposition uncovers an interesting property of the 6-bit APN permutation S_0: it is affine-equivalent to a 6-bit APN involution $S_{\mathcal{I}}$. The DDT and the LAT of the involution $S_{\mathcal{I}}$ are illustrated in Figure 5.9 (the DDT of $\mathsf{swap} \circ S_{\mathcal{I}} \circ \mathsf{swap}$ is illustrated, because it has clearer structure). $S_{\mathcal{I}}$ has differential uniformity 2 and its linearity is 16. The left and right halves of the output of $S_{\mathcal{I}}$ have algebraic degree 4 and 3 respectively.


(A) DDT of $\mathsf{swap} \circ S_{\mathcal{I}} \mathsf{swap}$ (white: 0, black: 2).

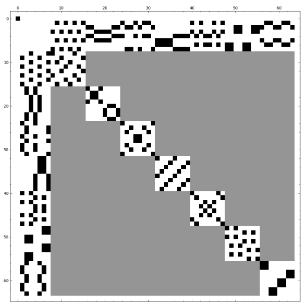

(B) LAT of $S_{\mathcal{I}}$ (white: 0, grey: 8, black: 16).

FIGURE 5.9: The DDT and the LAT of $S_{\mathcal{I}}$.

We used the algorithm from [BCBP03] to find all pairs of affine self-equivalence mappings, i.e. maps $A, B \in \mathsf{GA}_6(\mathbb{F}_2)$ such that $S_{\mathcal{I}} = B \circ S_{\mathcal{I}} \circ A$. In [BCBP03] it was suggested as a measure of symmetry of the permutation. The number of such pairs is invariant under affine-equivalence. Therefore, the decomposition is not necessary to count them. On the other hand, the decomposition shows that these maps have a simple expression. Let $(a, b) \otimes (c, d) := (ac, bd)$ denote the component-wise $\mathbb{F}_{2^3}$-multiplication. Then, for each $\lambda \in \mathbb{F}_{2^3}, \lambda \neq 0$ the following holds for all $x, y \in \mathbb{F}_{2^3}$:

$$S_{\mathcal{I}}(\lambda x, \lambda^{-1} y) = (\lambda, \lambda^{-1}) \otimes S_{\mathcal{I}}(x, y).$$

That is, multiplying the input halves by λ and λ^{-1} is equivalent to multiplying the output halves by λ and λ^{-1}. In Section 5.5 it is shown that this property is similar to a property that the Kim mapping has.

5.3.2 Univariate Representations

In this section I show that there exist 6-bit APN permutations with simpler univariate polynomials, than a random permutation or the Dillon's APN permutation has. These results are based on interpolating the involution $S_{\mathcal{I}}$ in $\mathbb{F}_{2^6} \simeq \mathbb{F}_{2^3} \times \mathbb{F}_{2^3}$ using different field basis. This is done by composing $S_{\mathcal{I}}$ with linear maps corresponding to the basis change. All polynomial presented in this section are defined over $\mathbb{F}_{2^6} \simeq \mathbb{F}_2[v]/(v^6 + v^4 + v^3 + v + 1)$, where v is primitive.

Single polynomial. In [BDMW10], the APN permutation was given as a univariate polynomial over $\mathbb{F}_{2^6}$ with 52 nonzero coefficients. Our decomposition allows to obtain an APN permutation from 25 monomials. The permutation s of $\mathbb{F}_{2^6}$ given by

$$\begin{aligned} s(x) = {} & x^{58} + x^{51} + x^{44} + x^{37} + v^{27}x^{36} + v^{38}x^{32} + x^{30} \\ & + v^{53}x^{28} + v^{7}x^{25} + v^{51}x^{24} + x^{23} + v^{53}x^{21} + v^{7}x^{18} + v^{24}x^{17} \\ & + v^{7}x^{16} + v^{46}x^{14} + v^{7}x^{11} + v^{4}x^{10} + x^{9} + v^{22}x^{8} + v^{46}x^{7} \\ & + v^{3}x^{4} + v^{50}x^{3} + v^{56}x^{2} + v^{52}x \end{aligned}$$

is APN.

Composition of 2 polynomials. Dillon *et al.* also represented S_0 as the composition $S_0 = f_2 \circ f_1^{-1}$, where polynomials f_1 and f_2 contain 18 monomials each. Using our decomposition, we found more compact polynomials. Let f'_1, f'_2 be permutations of $\mathbb{F}_{2^6}$ given by

$$f'_1(x) = v^{11}x^{34} + v^{53}x^{20} + x^8 + x,$$
$$f'_2(x) = v^{28}x^{48} + v^{61}x^{34} + v^{12}x^{20} + v^{16}x^8 + x^6 + v^2x.$$

Then $f'_2 \circ f'^{-1}_1$ is an APN permutation.

Composition of 3 polynomials. Finally, the representation becomes even simpler if 3 functions are used in the composition. Let i, m be permutations of $\mathbb{F}_{2^6}$ given by

$$i(x) = v^{21}x^{34} + x^{20} + x^8 + x, \quad m(x) = v^{52}x^8 + v^{36}x.$$

Then $i \circ m \circ i^{-1}$ is an APN permutation. Similarly, let i', m' be permutations of $\mathbb{F}_{2^6}$ given by

$$i'(x) = v^{37}x^{48} + x^{34} + v^{49}x^{20} + v^{21}x^8 + v^{30}x^6 + x, \quad m'(x) = x^8.$$

Then $i' \circ m' \circ i'^{-1}$ is also an APN permutation. These decompositions are obtained by interpolating parts of the decomposition separately. i and i' correspond to the part with the inverses and m, m' correspond to the central linear layer.

5.4 Modifying Components

In this Section I study flexibility of the decomposition and of the discovered structure. What can be changed without breaking the APN property?

5.4.1 Propagation of Affine Mappings through the Components

Recall the decomposition from Proposition 5.7 (Figure 5.7b). It is easy to check experimentally, that changing the xor constant or removing it does not break the APN property. Furthermore, the resulting permutation is affine-equivalent to S_0. It is not trivial to see how the constant xor goes through the nonlinear finite field inversions to the outside affine-equivalence maps. Furthermore, we observed experimentally, that changing the finite field inverse to an arbitrary 3-bit APN permutation (on the left branch in the first half of the structure its inverse must be used to preserve symmetry) leads to affine-equivalence with S_0 again. This property will be explained further in Section 5.4.3. From this observations we deduced the following theorem.

Theorem 5.10. *Let V be the permutation of $\mathbb{F}_2^3 \times \mathbb{F}_2^3$ given by*

$$V(x, k) := (\mathcal{I}(x \oplus \mathcal{I}(k)), k).$$

Let $L \in \mathsf{GL}_6(\mathbb{F}_2)$ be given by a 2×2 block matrix:

$$L := \begin{bmatrix} L_{1,1} & L_{1,2} \\ L_{2,1} & L_{2,2} \end{bmatrix},$$

where $L_{i,j} \in \mathbb{F}_2^{3\times 3}$ and $L_{1,2} \in \mathsf{GL}_3(\mathbb{F}_2)$.

Then, for any $a_l, a_r, b_l, b_r \in \mathbb{F}_2^3, A, B \in \mathsf{GL}_3(\mathbb{F}_2)$, the permutations S, S' of $\mathbb{F}_2^6$ given by

$$S := V^{-1} \circ L \circ V,$$
$$S' := V^{-1} \circ X_{b_l,b_r} \circ P_B \circ L \circ P_A \circ X_{a_l,a_r} \circ V,$$

are affine-equivalent. The structures of S, S' are illustrated in Figure 5.10.

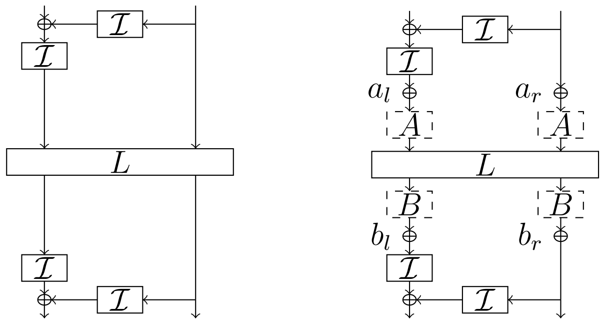


FIGURE 5.10: Affine-equivalent structures.

Proof. **Part 1: constants.** Consider the case when A and B are identity mappings. Let $a'_r, b'_r \in \mathbb{F}_2^3$ be given by

$$a = a_r \oplus L_{1,2}^{-1}(L_{1,1}(a_l) \oplus b_l),$$
$$b = b_r \oplus L_{2,1}(a_l) \oplus L_{2,2}(L_{1,2}^{-1}(L_{1,1}(a_l) \oplus b_l)).$$

It is easy to verify that $L(0, a) \oplus (0, b) = L(a_l, a_r) \oplus (b_l, b_r)$. Therefore, it is enough to consider the constants on the right branches:

$$X_{b_l,b_r} \circ L \circ X_{a_l,a_r} = X_{0,b} \circ L \circ X_{0,a}.$$

This transformation is illustrated in Figure 5.11a and Figure 5.11b.

It is left to show that the constant propagates through the Feistel function, the field inverse. Indeed, observe that

$$\mathcal{I}(x \oplus b) = (x + b)^6 = x^6 + b^2x^4 + b^4x^2 + b^6 = \mathcal{I}(x) \oplus i_b(x),$$

where $i_b \in \mathsf{GA}_3(\mathbb{F}_2)$ is an affine map. It can be also explained by the fact that $\mathcal{I}x$ is quadratic and therefore, all its derivatives are linear. i_b can be considered as an additional Feistel round, which is a part of the outer affine mapping. This transformation is shown in Figure 5.11c and Figure 5.11d. It can be equivalently applied to the other part of the structure to propagate the constant a outside the structure.

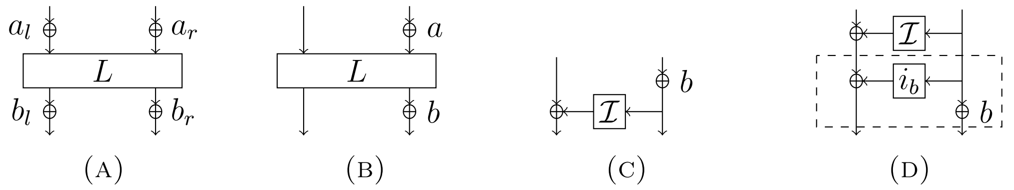


FIGURE 5.11: Propagating the constant XOR from the central layer to the outside affine layers.

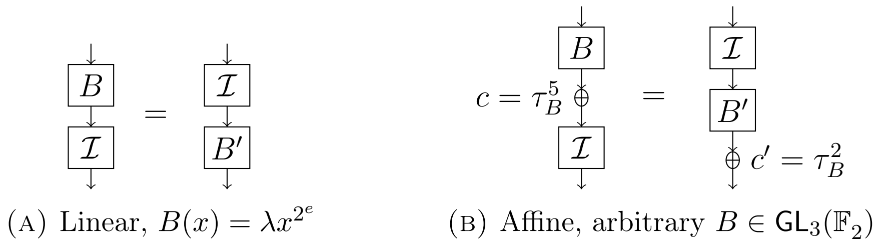


FIGURE 5.12: Propagation of linear/affine maps through the finite field inverse (only $\mathbb{F}_{2^3}$).

I conclude that arbitrary constants can be added around the central linear layer, without changing the affine-equivalence class.

Part 2: linear layers. Consider first the propagation of a linear map $L \in \mathsf{GL}_3(\mathbb{F}_2)$ through the finite field inverse $\mathcal{I}$. It can be verified exhaustively that the only linear maps that propagate through the inverse (i.e. a linear map B such that $\mathcal{I} \circ B = B' \circ \mathcal{I}$ and $B' \in \mathsf{GA}_6(\mathbb{F}_2)$) are the maps of the form $x \mapsto \lambda x^{2^e}$, where $\lambda \in \mathbb{F}_2^3, e \in \{0, 1, 2\}$. Indeed, $\mathcal{I}(\lambda x^{2^e}) = \lambda^6 (x^6)^{2^e} = \lambda^6 \mathcal{I}(x)^{2^e}$. This propagation is illustrated in Figure 5.12a.

How do other linear maps propagate then? It turns out that an addition of constant is needed in order to propagate an arbitrary linear map. Interestingly, this phenomenon seems to work only in $\mathbb{F}_{2^3}$. For example, in $\mathbb{F}_{2^4}, \mathbb{F}_{2^5}$ the only affine mappings that propagate through the field inverse into an affine mapping are those of the form $x \mapsto \lambda x^{2^e}$. The following observation was deduced and verified experimentally. Its effect is illustrated in Figure 5.12a.

Observation 5.11. *Let $B \in \mathsf{GL}_3(\mathbb{F}_2)$ be given by its $\mathbb{F}_{2^3}$ polynomial:*

$$B(x) := \lambda_4 x^4 + \lambda_2 x^2 + \lambda_1 x.$$

Let

$$\begin{aligned} \tau_B &:= \lambda_1 \lambda_2 \lambda_4 \in \mathbb{F}_{2^3}, \\ c &:= (\tau_B)^5 \in \mathbb{F}_{2^3}, \\ c' &:= (\tau_B)^2 \in \mathbb{F}_{2^3}, \\ B' &:= c' \oplus (\mathcal{I} \circ (B \oplus c) \circ \mathcal{I}), B' \colon \mathbb{F}_{2^3} \to \mathbb{F}_{2^3}. \end{aligned}$$

Then, B' is linear, i.e. $B' \in \mathsf{GL}_3(\mathbb{F}_2)$. By construction, it is such that

$$\mathcal{I} \circ (B \oplus c) = (B' \oplus c') \circ \mathcal{I}.$$

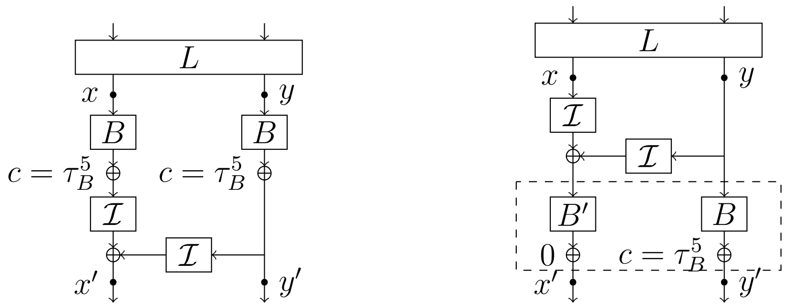


FIGURE 5.13: Propagation of affine mappings through the inverses. The dashed area contains the outer affine parts.

Furthermore, such c is uniquely determined by the mapping B. That is, for any other $\hat{c} \neq c$, the mapping

$$\mathcal{I} \circ (B \oplus \hat{c}) \circ \mathcal{I}$$

is never affine.

This observation sheds light on how arbitrary linear maps propagate through the inverse function. Note that the linear map

$$L' := P_B \circ L \circ P_A \in \mathsf{GL}_6(\mathbb{F}_2)$$

satisfies the conditions of this theorem, namely that the top-right 3×3 submatrix of L' is invertible. Therefore, the constants a_l, a_r, b_l, b_r can be arbitrarily modified. Let us change them to $\tau_A^5, \tau_A^5, \tau_B^5, \tau_B^5$ respectively. Let $x, y \in \mathbb{F}_2^3$ denote the output of the central linear layer L, and let $x', y' \in \mathbb{F}_2^3$ denote the output of the map S'. The placement of these variables is shown in Figure 5.13. Observe that

$$\begin{aligned} y' &:= B(y) \oplus \tau_B^5, \\ x' &:= \mathcal{I}(B(x) \oplus \tau_B^5) \oplus \mathcal{I}(B(y) \oplus \tau_B^5). \end{aligned}$$

By Observation 5.11, there exists $B' \in \mathsf{GL}_6(\mathbb{F}_2)$ such that

$$x' = B'(\mathcal{I}(x)) \oplus \tau_B^2 \oplus B'(\mathcal{I}(y)) \oplus \tau_B^2 = B'(\mathcal{I}(x) \oplus \mathcal{I}(y)).$$

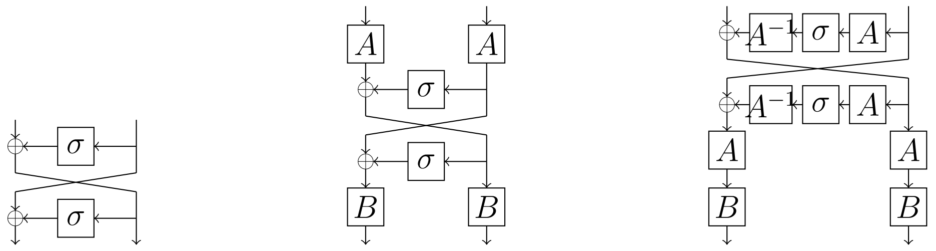
The propagation of the linear map A to the input affine layer is symmetric. This concludes the proof.

□

5.4.2 Modifying the Central Linear Layer

In Theorem 5.9 it was shown the permutation

$$S_{\mathcal{I}} := V^{-1} \circ M \circ V,$$



(A) The linear layer from the decomposition (B) Applying arbitrary linear bijections A and B (C) Moving the linear functions A down

FIGURE 5.14: Propagation of the linear function A through the middle linear layer.

where $M := \begin{bmatrix} w & w^6 \\ 1 & w \end{bmatrix} \in \mathsf{GL}_2(\mathbb{F}_{2^3})$, is APN. Such M has a 2-round Feistel network structure:

$$M(l, r) = (r + w(l + wr), l + wr).$$

The next proposition shows which Feistel functions can be used instead of multiplication by w.

Proposition 5.12. *Let $M_\sigma \in \mathsf{GL}_6(\mathbb{F}_2)$ be a 2-round Feistel network with $\sigma \in \mathsf{GL}_3(\mathbb{F}_2)$ as the Feistel function. Its 2×2 block matrix is*

$$M_\sigma = \begin{bmatrix} \sigma & I_{3\times 3} \oplus \sigma^2 \\ I_{3\times 3} & \sigma \end{bmatrix}.$$

Then the permutation S_σ of $\mathbb{F}_2^6$ given by

$$S_\sigma := V^{-1} \circ M_\sigma \circ V$$

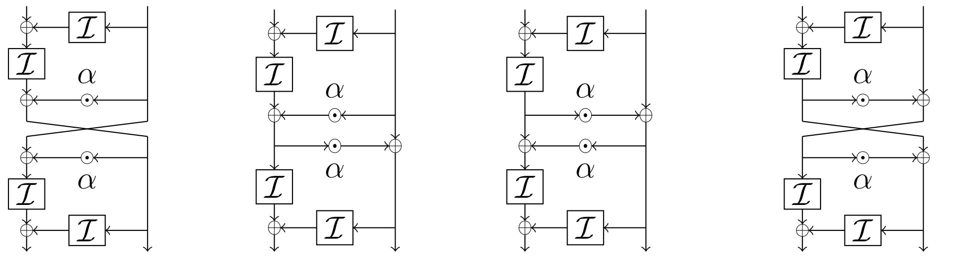
is APN if and only if σ is similar to the matrix of multiplication by w in $\mathbb{F}_{2^3}$. In particular, σ can be set to the matrix of multiplication by $c \in \mathbb{F}_{2^3}$ such that $c \neq 0$ and $\mathsf{Tr} c = 0$, independently of the choice of the field defining polynomial.

Proof. By Theorem 5.10, M_σ with $\sigma(x) = wx$ can be composed with P_A and P_B for arbitrary $A, B \in \mathsf{GL}_3(\mathbb{F}_2)$. Observe that P_A propagates through the Feistel network (see Figure 5.14:

$$P_B \circ M_\sigma \circ P_A = P_{B\times A} \circ M_{A^{-1}\times\sigma\times A}.$$

Setting $B = A^{-1}$ proves that σ can be replaced by any similar linear mapping. The fact that using another maps for σ does not lead to an APN permutation can be verified experimentally exhaustively.

Since multiplication by w works, it follows that multiplication by w^2 and w^4 work too, as all these are similar linear maps. In the finite field $\mathbb{F}_{2^3} \simeq \mathbb{F}_2[w']/(w'^3 + w'^2 + 1)$, such constants form the set $\{w'^3, w'^5, w'^6\}$. Observe that all this elements can be identified unambiguously with $\mathsf{Tr} c = 0, c \neq 0$. There are no other irreducible polynomials of degree 3 over $\mathbb{F}_2$. $\square$



(A) No swaps (B) Swap after (C) Swap before (D) Both swaps

FIGURE 5.15: Four APN permutations from different affine-equivalence classes, obtained by adding swaps before and/or after the central linear layer.

Besides modifying the Feistel function and composing P_B from one side, for arbitrary $B \in \mathsf{GL}_3(\mathbb{F}_2)$, there are another ways of modifying the central linear layer without breaking the APN property. We performed an exhaustive search over all invertible matrices $L \in \mathsf{GL}_6(\mathbb{F}_2)$, optimized by exploiting the equivalence classes given by Theorem 5.10. By analyzing the results, we deduced that the branch swap function swap can be appended and/or prepended to the linear layer L without breaking the APN property. However, on the contrast with transformations from Theorem 5.10, inserting these swaps results in APN permutations from different affine-equivalence classes. Though, all such APN permutations lie in the same CCZ-equivalence class. The 4 resulting affine-inequivalent APN permutation classes are represented in Figure 5.15.

Observation 5.13. *Let $M \in \mathsf{GL}_6(\mathbb{F}_2)$ be defined as in Theorem 5.9. Then the permutation $S_{M'}$ of $\mathbb{F}_2^6$ given by*

$$S_{M'} := V^{-1} \circ M' \circ V$$

is APN if

$$M' \in \{M, \mathsf{swap} \circ M, M \circ \mathsf{swap}, \mathsf{swap} \circ M \circ \mathsf{swap}\}.$$

5.4.3 Modifying the Inverse Mapping

In previous sections I showed how flexible is the linear layer in the decomposition. It is left to show how the nonlinear part, i.e. the finite field inverse function, can be modified without breaking the APN property. The following proposition shows that, in fact, any 3-bit APN permutation can be used instead.

Proposition 5.14. *Let A, B be 3-bit APN permutations. Let V_A, V_B be permutations of $\mathbb{F}_2^6$ defined by*

$$V_A(x,k) := (A^{-1}(x \oplus B(k)), k),$$
$$V_B(x,k) := (B^{-1}(x \oplus B(k)), k).$$

Let M be any linear map from Observation 5.13. Then the permutation $S_{A,B}$ of $\mathbb{F}_2^6$ given by

$$S_{A,B} = V_B^{-1} \circ M \circ V_A$$

is APN.

Proof. This proposition relies on the fact that all 10752 APN permutations of $\mathbb{F}_2^3$ are all pairwise affine-equivalent, established experimentally. By Theorem 5.10, we can compose M with P_C for arbitrary $C \in \mathsf{GA}_3(\mathbb{F}_2)$. Let $\mu, \eta \in \mathsf{GA}_3(\mathbb{F}_2)$ be such that

$$B = \eta \circ \mathcal{I} \circ \mu.$$

It follows that

$$P_\eta \circ V^{-1} \circ P_\mu(x,k) = \big(\eta \circ \mathcal{I} \circ \mu(x) \oplus \eta \circ \mathcal{I} \circ \mu(k), \eta \circ \mu(k)\big) = R \circ V_B^{-1},$$

where $R \in \mathsf{GA}_6(\mathbb{F}_2)$, $R(x,k) := (x, \eta \circ \mu \circ (k))$. By applying the same method to V_A, we show that $S_{A,B}$ is affine-equivalent to $S_\mathcal{I}$ and is an APN permutation. □

5.5 Relations with other Maps

5.5.1 Butterfly Structure

The structure discovered in the 6-bit APN permutation can be naturally generalized to arbitrary dimensions. In order to keep the algebraic properties, we restrict the nonlinear components to monomial functions in the finite field.

Definition 5.15 (Butterfly Structure)**.** *Let n be an integer, $n \geq 3$, let $\alpha \in \mathbb{F}_{2^n}$, e be an integer such that $x \mapsto x^e$ is a permutation of $\mathbb{F}_{2^n}$. Let $r_{e,\alpha}, R_{e,\alpha}$ be defined as*

$$\begin{aligned} &r_{e,\alpha} \colon \mathbb{F}_{2^n} \times \mathbb{F}_{2^n} \to \mathbb{F}_{2^n}, \\ &r_{e,\alpha}(x,k) = (x + \alpha k)^e + k^e, \\ &R_{e,\alpha} \colon \mathbb{F}_{2^n} \times \mathbb{F}_{2^n} \to \mathbb{F}_{2^n} \times \mathbb{F}_{2^n}, \\ &R_{e,\alpha}(x,k) = (r_{e,\alpha}(x,k), k). \end{aligned}$$

We call Butterfly Structures *the mappings of $\mathbb{F}_{2^n} \times \mathbb{F}_{2^n}$ to itself defined as follows:*

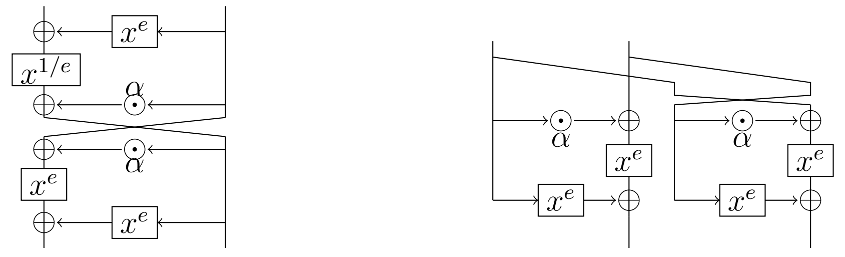
- *the* Open Butterfly *with branch size n, exponent e and coefficient α is the permutation denoted H_e^α defined by:*

$$\mathsf{H}_e^\alpha = R_{e,\alpha} \circ \mathsf{swap} \circ R_{e,\alpha}^{-1}.$$

- *the* Closed Butterfly *with branch size n, exponent e and coefficient α is the function denoted V_e^α defined by:*

$$\mathsf{V}_e^\alpha(x,y) = \big(r_{e,\alpha}(x,y), r_{e,\alpha}(y,x)\big).$$

The butterfly structure was generalized and studied in consequent works [LTYW18, CDP17, FFW17]. Many instances of this structure are proved to be differentially 4-uniform, but no new APNs were found. Recently, Canteaut *et al.*



(A) Open (bijective) butterfly H_e^α. (B) Closed (non-bijective) butterfly V_e^α.

FIGURE 5.16: The two kinds of butterfly structure.

prove [CPT18] that the generalized butterfly structure is never APN for $n > 6$. In this chapter I analyze only relations between the butterfly structure and other mappings in $\mathbb{F}_2^6$.

Proposition 5.16. *For all n, e, α, the structures $H := \mathsf{H}_e^\alpha$ and $V := \mathsf{V}_e^\alpha$ are CCZ-equivalent.*

Proof. Let Γ_H and Γ_V be the graphs of H and V respectively. Let $R := R_{e,\alpha}, r := r_{e,\alpha}$. Observe that

$$\Gamma_V = \left\{ \big(x, y, r(x, y), r(y, x)\big) \mid x, y \in \mathbb{F}_{2^3} \right\},$$
$$\Gamma_H = \left\{ \big(r(y, x), x, r(x, y), y\big) \mid x, y \in \mathbb{F}_{2^3} \right\}.$$

Clearly, the graphs differ by a simple reordering of the 3-bit nibble and thus are linear-equivalent. □

5.5.2 Relations with the Kim Mapping

Recall the Kim mapping $\kappa\colon \mathbb{F}_{2^6} \to \mathbb{F}_{2^6}$:

$$\kappa(x) := vx^{24} + x^{10} + x^3,$$

where v is a primitive element of $\mathbb{F}_{2^6}$. It is CCZ-equivalent to the Dillon's APN permutation. It turns out that the Kim mapping is actually affine-equivalent to the closed butterfly V_6^w, the closed version of $S_\mathcal{I} = \mathsf{H}_6^w$. This equivalence sheds light on the structure of the CCZ-transformation applied by Dillon *et al.* to the Kim map in order to obtain the APN permutation. Indeed, it can be seen as "opening" the closed butterfly V_6^w, in a particular field basis. Note that Proposition 5.16 shows that V_6^w (a quadratic APN function) can be CCZ-transformed into H_6^w (a degree-4 APN permutation) simply by reordering nibbles in the function's graph.

Observation 5.17. *The functions V_6^w and κ are affine-equivalent:*

$$\kappa = B \circ \mathsf{V}_6^w \circ A,$$

where $A, B \in \mathsf{GL}_6(\mathbb{F}_2)$ are given by

$$A = \begin{bmatrix} 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \end{bmatrix}, \quad B = \begin{bmatrix} 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 & 1 \end{bmatrix}.$$

An anonymous reviewer of [PUB16] pointed out that the Kim mapping has the following property: for all $\lambda, x \in \mathbb{F}_{2^3}$

$$\kappa(\lambda x) = \lambda^3 \kappa(x).$$

The closed butterfly inherits this linear self-equivalence property, which is then expressed in a bivariate way. The following proposition describes this expression and generalizes the similar property of $S_{\mathcal{I}}$ observed in Section 5.3.

Proposition 5.18 (Linear Self-Equivalence of Butterflies)**.** *For any n, e, α, the closed butterfly V_e^α satisfies the following property: for all $\lambda, x, y \in \mathbb{F}_{2^n}$*

$$\mathsf{V}_e^\alpha(\lambda x, \lambda y) = (\lambda^e, \lambda^e) \otimes \mathsf{V}_e^\alpha(x, y).$$

Furthermore, the open butterfly H_e^α, when it is well-defined, satisfies the following property: for all $\lambda, x, y \in \mathbb{F}_{2^n}$

$$\mathsf{H}_e^\alpha(\lambda^e x, \lambda y) = (\lambda^e, \lambda) \otimes \mathsf{H}_e^\alpha(x, y).$$

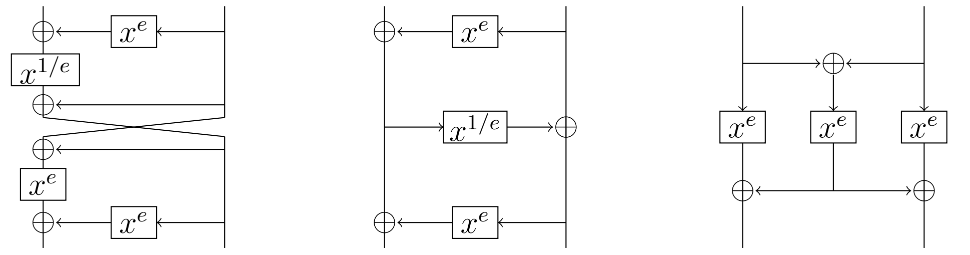
Proof. The propagation of multiplications by λ can be easily traced through the structures. □

5.5.3 Relation with a 3-round Feistel Network

Consider butterfly structures with $e = 1$. The open butterfly with $e = 1$ (Figure 5.17a) is functionally equivalent to a 3-round Feistel network with Feistel functions $x^e, x^{1/e}, x^e$ (Figure 5.17b). The closed butterfly with $e = 1$ (Figure 5.17c) is an interesting structure similar to the Lai-Massey structure.

Definition 5.19 (The F^e structure)**.** *Let $n \geq 1$, and let e be an integer such that x^e is a permutation of $\mathbb{F}_{2^n}$. The structure F^e is defined as a $2n$-bit 3-round Feistel network with round functions $x^e, x^{1/e}, x^e$, where $1/e$ is the multiplicative inverse of e modulo $2^n - 1$.*

In [LW14] the authors notice that the 6-bit structure F^3 is CCZ-equivalent to the monomial mapping x^5 of $\mathbb{F}_{2^3}$ to itself. We noticed that the closed butterfly V_5^1 is affine-equivalent to the monomial mapping x^5. From CCZ-equivalence of open and closed butterflies, we obtain a full proof of CCZ-equivalence of the monomial mapping x^5 and the Feistel network F^3. We generalize these observations in the following theorem.



(A) Open butterfly H_e^1 (B) $\mathsf{F}^e = \mathsf{H}_e^1$ (C) Closed butterfly V_e^1

FIGURE 5.17: The equivalence between H_e^1 and F^e.

Theorem 5.20. *Let $n \geq 3$ be an odd integer and $e = 2^{2k} + 1$ for some positive integer k. Then the closed $2n$-bit butterfly V_e^1 is linear-equivalent to the monomial mapping $x \mapsto x^e$ of $\mathbb{F}_{2^{2n}}$.*

Corollary 5.21. *Let $n \geq 3$ be an odd integer and $e = 2^{2k} + 1$ for some $k \in \mathbb{Z}_+$, such that the monomial $x \mapsto x^e$ defines a permutation of $\mathbb{F}_{2^{2n}}$. Then the $2n$-bit Feistel Network F^e is CCZ-equivalent to this permutation.*

Proof. Let us represent an element x of $\mathbb{F}_{2^{2n}}$ by a linear polynomial $x = au + b$ over $\mathbb{F}_{2^n}$ with multiplication modulo the irreducible polynomial $u^2 + u + 1$. Note that $u^2 = u + 1, u^4 = u, \ldots, u^{2^{2k}} = u, u^{2^{2k}+1} = u + 1$. Then, by linearity of $x \mapsto x^{e-1}$:

$$\begin{aligned} x^e = (au+b)^e = (au+b)^{e-1}(au+b) &= a^e u^e + a^{e-1}u^{e-1}b + aub^{e-1} + b^e \\ &= (a^e + a^{e-1}b + ab^{e-1})u + a^e + b^e \\ &= (b^e + (a+b)^e)u + a^e + b^e. \end{aligned}$$

Note that $(au+b) \mapsto ((a+b)u+a)$ is a linear map. Therefore $(au+b) \mapsto (au+b)^e$ is linear-equivalent to

$$(a^e + (a+b)^e)u + b^e + (a+b)^e.$$

This expression is exactly the same as in the closed butterfly:

$$\mathsf{V}_e^1(a,b) = (a^e + (a+b)^e, b^e + (a+b)^e)).$$

Therefore, F^e is linear-equivalent to H_e^1. Finally, H_e^1 is CCZ-equivalent to V_e^1, whenever $x \mapsto x^e$ defines a permutation. □

5.6 Conclusions

In this chapter I described a decomposition of the 6-bit APN permutation, which we obtained together with my colleagues Léo Perrin and Alex Biryukov. The discovered structure is simple and algebraic: it is based on the finite field arithmetics. We generalized this structure to larger dimensions, though no new APN permutations were found. The decomposition also shed more light on the process used to obtain it by Dillon *et al.*. Furthermore, many new interesting properties and relations with other structures were observed.

I would like to highlight main tools used to obtain the decomposition and study it:

1. TU-decomposition. This tool is the most effective way to obtain a high-level decomposition when it is possible.

2. Affine- and linear- equivalence algorithm from [BCBP03]. It helps to discover relations between different components and between parts of a single component, e.g. between single permutations inside a keyed permutation. The algorithm is also useful to find affine and linear- self-equivalence mapping pairs.

3. Polynomial interpolation in the finite field. This tool was particularly useful due to the mathematical nature of the analyzed object.

4. Algebraic degree evaluation. In the decomposition process it makes sense to choose steps that result in components of lower algebraic degree.

Unfortunately, no new APN permutations in even dimensions were found, even though many natural generalizations emerged. Recently, Canteaut *et al.* proved that a generalized butterfly structure is not APN for $n > 6$. Therefore, the big APN problem is still unsolved:

Do there exist APN permutations of $\mathbb{F}_{2^n}$ for even $n \geq 8$?

Part II

Nonlinear Invariant Cryptanalysis

In this part, I present the work I have done on cryptanalysis techniques based on nonlinear invariants and, in particular, invariant subspaces. These techniques were applied recently to several lightweight block ciphers [LAAZ11, TLS16] and are studied actively [BCLR17, Bey18, WYWP18, BCL18].

First, I contributed to the analysis of the permutation used in the NORX authenticated encryption, a third-round candidate of the CAESAR competition [Com19]. It is a joint work with Alex Biryukov and Vesselin Velichkov, available as a report [BUV17]. We found symmetries on different levels of the structure and verified the absence of nonlinear invariants of low degree in the G function used in NORX8. One of the symmetries was independently discovered in [CFG+17] and was used to attack the previous version of NORX.

Second, together with Christof Beierle and Alex Biryukov, we developed theoretical aspects of nonlinear invariants with respect to linear layers [BBU18]. We studied whether quadratic invariant attacks from [TLS16] can be generalized to higher degrees. As one of our results, we prove that there exist no bijective linear maps that preserve cubic invariants of the same form as in [TLS16]. We also show that such *expanding* linear maps exist and study the minimum expansion rate. This work is currently available as a report [BBU18].

Chapter 6

Analysis of the NORX Permutation

In this chapter, I describe an analysis of the core permutation F of NORX [AJN16], one of the fifteen authenticated encryption algorithms that have reached the third round of the CAESAR competition [Com19]. I show that it has rotational symmetries on different structure levels. This yields simple distinguishing properties for the permutation, which propagate with very high probability or even probability one. The stronger symmetry was independently discovered by Chaigneau *et al.* in [CFG+17] and was used to attack a previous version of NORX. The latest version of NORX is not susceptible to the attack. I describe three attacks on slightly modified variants of NORX exploiting the discovered symmetries.

I also propose an algorithm to prove absence of low-degree nonlinear invariants based on the cycle decomposition of a permutation. I use it to prove that there are no nonlinear invariants of a low degree in the 32-bit permutation G used in NORX8.

6.1 Introduction

CAESAR is a finished competition of authenticated ciphers aiming to select a portfolio of ciphers suitable for different usage scenarios. NORX [AJN16] is one of the fifteen candidates that have reached the third round. NORX is based on the Monkey Duplex [BDPVA12a, BDPVA12c] construction which is a sponge mode tailored for authenticated encryption schemes.

In this chapter, I report on some non-random properties of the NORX permutation. More specifically, I show that it exhibits some rotational symmetries on different structure levels. They yield simple distinguishing properties for the permutation, which propagate with very high probability or even probability one.

6.1.1 Outline

The rest of the chapter is organized as follows. I begin by briefly outlining the NORX algorithm in Section 6.2. In Section 6.3, I describe rotational symmetric properties in its core permutation, both at the state and at the word level and show attacks on slightly modified versions of NORX exploiting these properties. In Section 6.4, I propose an algorithm to search for nonlinear invariants of low-degree from the cycle decomposition of a permutation and apply the algorithm to the permutation G used in NORX8.

6.2 Description of NORX

NORX has a sponge structure and is based on the monkeyDuplex construction. It uses ARX-like (Addition/Rotation/XOR) operations. More specifically, it is inspired by the ChaCha stream cipher, where the addition operation is replaced by its 1-st order approximation: $x \oplus y \oplus ((x \& y) \ll 1)$.

The original submission [AJN16] proposes versions of NORX with 32- and 64-bit words called respectively NORX32 and NORX64. Subsequently two more versions were proposed, with 8- and 16-bit words called resp. NORX8 and NORX16 [AJN15]. The word size is denoted by w. The internal state of all NORX variants is composed of 16 words organized as a 4×4 matrix.

The basic building block of NORX is a permutation F of $\mathbb{F}_2^b \simeq (\mathbb{F}_2^w)^{16}$, where $b = r + c = 16w$ is called the *width*, r is the *rate* and c is the *capacity*. F is also called a *round*, and F^l is an l-fold iteration of F. The recommended instances of NORX use $l = 4$ or $l = 6$ rounds. The initialization phase is always followed by a data processing phase and as a result the state effectively goes through F^{2l} before any absorption. NORX allows parallelization but we consider only the sequential construction (the parameter $p = 1$). The parameter combinations of the NORX variants are given in Table 6.1. The full scheme is depicted on Figure 6.1.

F is composed of the *column step* denoted by $F_{col}\colon (\mathbb{F}_2^w)^{16} \to (\mathbb{F}_2^w)^{16}$ followed by the *diagonal step* denoted by $F_{diag}\colon (\mathbb{F}_2^w)^{16} \to (\mathbb{F}_2^w)^{16}$:

$$F = F_{diag} \circ F_{col}.$$

word size (w)	rounds (l)	rate (r)	capacity (c)	state size (b)	nonce (N)	key (K)	tag (t)
8	4 or 6	40	88	128	32	80	80
16	4 or 6	128	128	256	32	96	96
32	4 or 6	768	256	1024	128	128	128
64	4 or 6	1536	512	2048	256	256	256

TABLE 6.1: Parameters of the NORX instances.

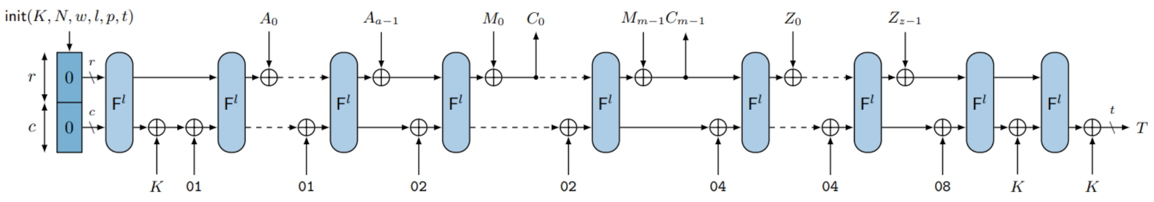


FIGURE 6.1: The NORX v3.0 AE scheme with parallelization parameter $p = 1$. K and N denote a key and a nonce resp., A and Z denote a header and a trailer resp., M_i and C_i denote plaintext and ciphertext blocks resp., T is the authentication tag. (credits: NORX specification [AJN16])

Let $G(a, b, c, d)$ be the permutation of $(\mathbb{F}_2^w)^4$ represented by the circuit shown in Figure 6.2. The column step F_{col} applies G in parallel to each of the 4 columns. The diagonal step F_{diag} applied G in parallel to each of the 4 main diagonals. These steps are illustrated in Figure 6.3.

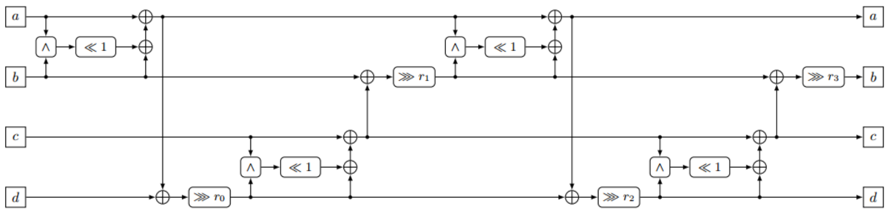


FIGURE 6.2: The G circuit used in NORX. (credits: NORX specification [AJN16])

The security of each of the four versions of NORX is limited by the key size and the tag size. The designers require unique nonces and abort on verification failure. In addition, at most 2^e messages are allowed to be processed with a single key, where e is equal to 24, 32, 64, 128 respectively for NORX8, NORX16, NORX32, NORX64.

For a more detailed description of NORX I refer the reader to the specification [AJN16].

6.3 Rotational Invariants in NORX

In this section I describe rotational symmetries in the permutation F of NORX. They exist both on the word level (inherited from G) and on the state level (structural).

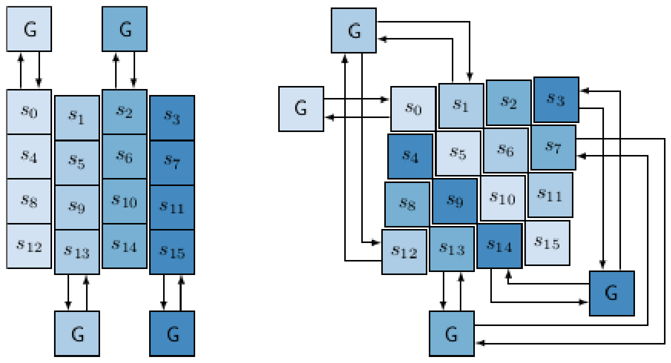


FIGURE 6.3: The G circuit applied to the columns (left) and diagonals (right) of the state. (credits: NORX specification [AJN16])

6.3.1 State Invariants

We can see a 4x4 NORX state S as a list of 4 columns: $S = (c_1, c_2, c_3, c_4)$.

Definition 6.1 (Columns Rotation)**.** *For an integer n denote by R_n the function rotating of the columns left by n positions. For example $R_1(c_1, c_2, c_3, c_4) = (c_2, c_3, c_4, c_1)$ for arbitrary $c_1, c_2, c_3, c_4 \in (\mathbb{F}_2^w)^4$.*

The following proposition shows that the permutation F is column rotation-symmetric.

Proposition 6.2. *The permutations R_n and F^l commute for any integers n and $l \geq 1$:*

$$F^l \circ R_n = R_n \circ F^l.$$

Proof. Clearly, the rotation of columns does not affect the column step F_{col}, since it transforms each column separately: $F_{col} \circ R_n = R_n \circ F_{col}$. Such rotations do not break the diagonals as well, because the diagonals are simply reordered. Therefore, $F_{diag} \circ R_n = R_n \circ F_{diag}$. It follows that F commutes with R_n and thus F^l commutes with R_n too. □

Definition 6.3. *A state $s \in (\mathbb{F}_2^w)^{16}$ is said to be* column n-rotation invariant *if*

$$R_n(s) = s.$$

Let $s \in (\mathbb{F}_2^w)^{16}$ be a column n-rotation invariant state for a fixed positive integer n. Observe that

$$R_n(F(s)) = F(R_n(s)) = F(s),$$

i.e. $F(s)$ is also column n-rotation invariant. It follows that the property of a state being column n-rotation invariant is an invariant of the round function F. It is easy to see that this invariant corresponds to an invariant subspace.

Proposition 6.4. *For a fixed integer $n, 1 \leq n \leq 3$, the set of all column n-rotation invariant states is a linear subspace of $(\mathbb{F}_2^w)^{16}$. For $n = 1$ or $n = 3$ this is the same subspace of dimension $4w$, for $n = 2$ the invariant subspace has dimension $8w$.*

Proof. If $n = 1$ or $n = 3$, then for any $c_1, c_2, c_3, c_4 \in (\mathbb{F}_2^w)^4$

$$(c_1, c_2, c_3, c_4) = (c_2, c_3, c_4, c_1)$$

and it follows that all columns are equal: $c_1 = c_2 = c_3 = c_4$. There are 2^{4w} out of 2^{16w} such states. The designers of NORX noted these states in [AJN14]. A constraint $c_i = c_j$ consists of $4w$ linear equations $c_{i,y,x} \oplus c_{j,y,x} = 0$, where $1 \le y \le 4, 1 \le x \le w$. Therefore, these constraints define a linear subspace of dimension $16w - 3 \cdot 4w = 4w$.

If $n = 2$, then for any $c_1, c_2, c_3, c_4 \in (\mathbb{F}_2^w)^4$

$$(c_1, c_2, c_3, c_4) = (c_3, c_4, c_1, c_2)$$

and it follows that the two pairs of columns are equal: $c_1 = c_3$ and $c_2 = c_4$. There are 2^{8w} out of 2^{16w} such states. Similarly, these constraints define a linear subspace of dimension $8w$. $\square$

Hitting such a special state even for the case $n = 2$ is not easy under the NORX security claims. However, 2^{8w} is a more serious fraction of states than the 2^{4w} weak states which were known to the designers. To illustrate possible dangers of such properties, I refer to the forgery attack [CFG+17] on the previous version of NORX exploiting this invariant, and I also describe two hypothetical attacks on NORX8 [AJN15], a NORX version with 8-bit words for low-end devices. I remark that NORX8 is not a part of the CAESAR submission.

The fist attack shows a weak-key set, which could be exploited if the domain separation constants were rotation-invariant. The weak-key set is relatively small, 2^{32} keys out of 2^{80}. The second attack is a state/key recovery attack in a known plaintext scenario. It succeeds with probability 2^{-64} for each two consequent known-plaintext blocks, and the total time complexity is 2^{72} to recover an 80-bit key. Note that the designers restrict the data per single key to 2^{24} message blocks, therefore, the attack can break a concrete key with probability only 2^{-40}.

Both attacks are independent of the number of rounds l used in the permutation.

6.3.2 Hypothetical Weak-key Attack on NORX8 Initialization

The initial state of NORX8 is given by

$$\begin{pmatrix} n_1 & n_2 & n_3 & n_4 \\ k_1 & k_2 & k_3 & k_4 \\ k_5 & k_6 & k_7 & k_8 \\ k_9 \oplus w & k_{10} \oplus l & u_{15} \oplus p & u_{16} \oplus t \end{pmatrix} \in (\mathbb{F}_2^8)^{16}, \tag{6.1}$$

where n_i and k_i denote bytes of the nonce and the key respectively, u_i are constants and w, l, p, t are constants encoding parameters of NORX. It is possible to construct valid initial states with two equal halves, i.e. a column 2-rotation invariant state. Indeed, let us fix the four key bytes (k_3, k_4, k_7, k_8) arbitrarily and let us choose the two nonce bytes (n_3, n_4) arbitrarily. Then we can set the left half of the state equal to the right half, i.e.

$$\begin{aligned}(n_1, n_2) &= (n_3, n_4),\\ (k_1, k_2) &= (k_3, k_4),\\ (k_5, k_6) &= (k_7, k_8),\\ (k_9, k_{10}) &= (u_{14} \oplus p \oplus w, u_{15} \oplus t \oplus l).\end{aligned}$$

There are 2^{32} weak keys out of 2^{80} and 2^{16} nonces that result in such a weak state. The column 2-rotation invariant of such state is preserved through arbitrary number of rounds of F. However, after the first F^l rounds the domain separation constant will be added to the last word of the state (see Figure 6.4). This constant is not column 2-rotation invariant and therefore it will break the property. Therefore, we consider a slightly modified version of NORX8 where the domain separation constant is column 2-rotation invariant. For example, the original constant may be added not only to the last word, but to all words of the state or to all words in the last row. In such case the invariant is preserved through the next F^l rounds and the rate part of the state is then observed by an adversary. This leads to a simple distinguisher: the adversary simply compares the left and right halves of the exposed part of the state. In NORX8 the rate part consists of only 5 bytes. It allows to check only the topmost 4 words with error probability 2^{-16}. By using a few more encryptions with another weak nonces the error probability can be decreased to negligible.

I remark that the weak key space is very small and the attack requires symmetric domain separation constants. On the other hand, it is powerful in that it is independent of the number of rounds. The attack illustrates possible dangers of having such strong invariants in the permutation.

6.3.3 State Recovery Attack on NORX8

The column 2-rotation invariant can be used to mount a state/key recovery attack on NORX8, though exceeding the data usage limit defined by the designers.

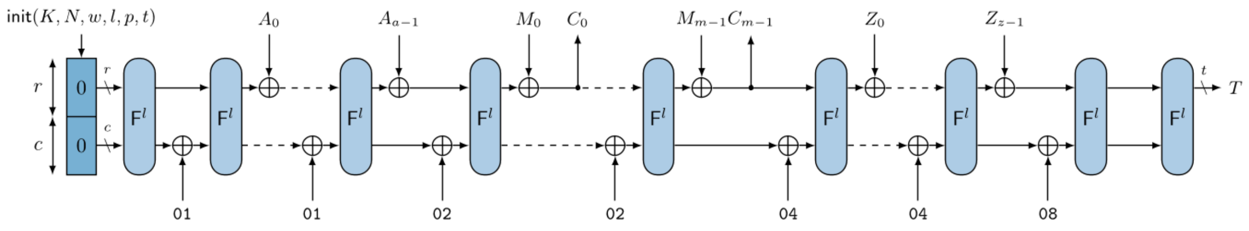


FIGURE 6.4: The NORX v2.0 AE scheme with parallelization parameter $p = 1$. NORX8 and NORX16 follow this scheme. (credits: NORX specification [AJN16])

Assume that we have a two-block known-plaintext message. That is, we know the rate part before and after a call to the NORX8 core permutation F^l. Denote the input rate part by a and the output rate part by b. Recall that the rate in NORX8 is 40 bits, which is five 8-bit words. With probability 2^{-16} we will observe $a_1 = a_3, a_2 = a_4$. Then there are two cases:

1. The whole state is column rotation-2 invariant. The probability of this is equal to $2^{-6 \cdot 8} = 2^{-48}$, given the observed rate part. Indeed, a uniformly random state is column 2-rotation invariant with probability 2^{-64}. In this case the output state will be also column rotation-2 invariant with probability 1 and we will observe $b_1 = b_3, b_2 = b_4$.

2. The whole state is not column rotation-2 invariant. Then with probability 2^{-16} we will observe $b_1 = b_3, b_2 = b_4$ as a false positive.

As a result, when we observe both $a_1 = a_3, a_2 = a_4$ and $b_1 = b_3, b_2 = b_4$, the probability of the state being column rotation-2 invariant is equal to 2^{-32} and in the other cases it is a false positive. In the first case the state before the call to F^l contains 5 unknown words $x_1, \ldots, x_5 \in \mathbb{F}_2^8$:

$$\begin{pmatrix} a_1 & a_2 & a_3 = a_1 & a_4 = a_2 \\ a_5 & x_1 & a_5 & x_1 \\ x_2 & x_3 & x_2 & x_3 \\ x_4 & x_5 & x_4 & x_5 \end{pmatrix}.$$

We can exhaustively check all 2^{40} possibilities for $x_1, \ldots, x_5$ by encrypting through F^l and obtaining extra filter with probability 2^{-24} from b. The remaining 2^{16} candidates can be checked by decrypting the state up to the initial state and matching the constants and further verifying the tag.

As a result, with probability 2^{-64} two consequent known-plaintext blocks allow to recover the full state and the secret key. The initial filter has strength 2^{-32} and the time complexity of checking a block pair is 2^{40}. Note that the designers set a limit to 2^{24} data, therefore the attack succeeds for a concrete key only with probability around 2^{-40}.

6.3.4 Word Invariants

A similar rotational symmetry exists on the word level too. Let G' be the permutation of $(\mathbb{F}_2^w)^4$ to itself obtained from G by replacing the four left shift operations by left rotations.

Proposition 6.5. *$G' = G$ is conditioned by 4 bit equations, where each equation holds with probability* $3/4$.

Proof. The left shift by one inserts a zero in the least significant bit of the result. If the most significant bit of the input is equal to 0, then the left shift is equivalent to the left rotation. There are 4 left shifts in G, each yields such bit equation. The input of a left shift in G is simply an AND of two state bits, which are uniformly distributed. □

Observation 6.6. *Experimentally, it is observed that* $\Pr[G' = G]$ *is close to* $2^{-1.82}$*, where the input is sampled uniformly at random, for all word sizes* $w \in \{8, 16, 32, 64\}$.

Note that this observation shows the effect of dependency of the four quarter-steps in G. The probability that all these bits are equal to zero can be estimated as $(3/4)^4 \approx 2^{-1.66}$. However, the actual probability is lower due to the dependency of the equations.

Definition 6.7. *Let $r_n\colon \mathbb{F}_2^w \to \mathbb{F}_2^w$ be the mapping which rotates a word left by n bits and let $\mathbf{r}_n\colon (\mathbb{F}_2^w)^4 \to (\mathbb{F}_2^w)^4$ be defined as*

$$\mathbf{r}_n(a,b,c,d) \coloneqq (r_n(a), r_n(b), r_n(c), r_n(d)).$$

Proposition 6.8. *For any integer $n, 1 \le n < w$, $\mathbf{r}_n$ commutes with G':*

$$G' \circ \mathbf{r}_n = \mathbf{r}_n \circ G'.$$

Furthermore, $\mathbf{r}_n$ commutes with G conditioned by 8 bit equations, each holding with probability $3/4$.

Proof. First, it is easy to verify that all operations in G' commute with $\mathbf{r}_n$. For a binary operation to commute it is required that $\mathbf{r}_n$ applied to both inputs is equivalent to $\mathbf{r}_n$ applied to the output.

The second claim follows by applying Proposition 6.5 to the equation $G' \circ \mathbf{r}_n = \mathbf{r}_n \circ G'$ two times. □

Observation 6.9. *Experimentally, it is observed that* $\Pr[G' \circ \mathbf{r}_n = \mathbf{r}_n \circ G']$ *varies from $2^{-3.84}$ to $2^{-3.59}$ depending on the word size and rotation amount n. The rotation amounts corresponding to the smallest probabilities are 1 and $w-1$.*

Similarly to the column n-rotation invariant, define the word n-rotation invariant.

Definition 6.10. *A columns $c \in (\mathbb{F}_2^w)^4$ (resp. a state $s \in (\mathbb{F}_2^w)^{16}$) is said to be word n-rotation invariant if for each its word c_i (resp. s_i) the following holds:*

$$r_n(c_i) = c_i \quad (\textit{resp. } r_n(s_i) = s_i).$$

Proposition 6.11. *The set of all word n-rotation invariant states is a linear subspace of dimension $16 \cdot gcd(n,w)$.*

Proof. It is easy to see that a word $v \in \mathbb{F}_2^w$ is word n-rotation invariant if and only if it is made of $w/gcd(n,w)$ copies of the same vector $u \in \mathbb{F}_2^{gcd(n,w)}$. Clearly, all such words form a linear subspace of $\mathbb{F}_2^w$ of dimension $gcd(n,w)$. As there are 16 words in the state, the proposition follows. □

Note that the property of a state or column being invariant requires only one approximation of G by G', i.e. it is approximately twice as more probable than the commutation.

Proposition 6.12. *Let $c \in (\mathbb{F}_2^w)^4$ be a word n-rotation invariant column. Then*

$$\Pr[\mathbf{r}_n(G(c)) = G(c)] \ge \Pr[G(c) = G'(c)],$$

where the probabilities are taken over c sampled uniformly at random from the set of all word n-rotation invariant columns.

Proof. Consider the following equation:

$$\mathbf{r}_n(G(c)) \approx \mathbf{r}_n(G'(c)) = G'(\mathbf{r}_n(c)) = G'(c) \approx G(c).$$

The two approximations are applied to the same input: $G(c) \approx G'(c)$, therefore the equation holds with probability at least $\Pr[G(c) = G'(c)]$. □

Experimentally, no difference is observed on $\Pr[G(c) = G'(c)]$ when c is sampled uniformly at random from $(\mathbb{F}_2^w)^4$ and when it is sampled uniformly at random from the set of all word n-rotation invariant columns. Therefore, it can be expected that a word n-rotation invariant is preserved through F with a probability approximately $(2^{-1.82})^8 = 2^{-14.56}$. The commutation of F and the n-rotation of each word can be expected to happen with probability approximately $(2^{-3.59})^8 = 2^{-28.72}$ if $1 < n < w - 1$.

It is worth noting that the word n-rotation invariants can be seen as probabilistic invariant subspaces of F.

6.3.5 Hypothetical Attack on NORX128 v2.0

As the probability of $\mathbf{r}_n$ commuting with G' does not seem to depend on the word size, the distinguishing property is stronger for instances with larger words and key size. I consider an existential forgery attack similar to the one proposed in [CFG+17]. Similarly, I consider NORX v2.0 since NORX v3.0 breaks the attack by injecting the key in the finalization stage.

Consider the forgery attack scenario. The finalization stage of NORX consists of 8 iterations of F. Let us assume that the words in the rate part of the state before the finalization are $w/2$-rotation invariant. This happens with probability 2^{-2w}. Then we can attempt a forgery by rotating each word in the last ciphertext block by $w/2$. Then, with probability approximately $(2^{-3.59})^{64} = 2^{-229.76}$ we expect the rotation to commute with the finalization:

$$F^8(\mathbf{r}_n(s)) = \mathbf{r}_n(F^8(s)),$$

where s is the state before the finalization stage. Since the tag is obtained by truncating the final state and we have observed the tag in the first encryption, we can expect the new tag to be equal to the word $w/2$-rotated version of the original tag.

For NORX64, the probability of the rate to be 32-rotation invariant is equal to 2^{-128}. Unfortunately, the attack's success probability then is worse than for a generic attack (i.e. 2^{-256}). For this reason, I suggest to increase the word size even more and to consider NORX128, a generalization of NORX64 by increasing the word size to 128 bits. In this hypothetical cipher, the full attack success probability is approximately $2^{-256} \cdot 2^{-229.76} < 2^{-512}$, i.e. it is better than a generic attack.

This attack on the hypothetical instance of NORX shows the possibility of exploiting the word-level symmetries as well. The attack does not apply directly to main instances of NORX.

6.4 Proving Absence of Low Degree Invariants

Recently, a nonlinear invariant attack was introduced by Todo *et al.* in [TLS16]. They show that, for any SPN-based cipher, if there exists a quadratic invariant for the S-Box and the linear layer is orthogonal, then it is possible to construct a nonlinear invariant for the full round of the cipher. Together with Christof Beierle and Alex Biryukov, we studied and generalized such linear layers in [BBU18]. I describe the results in Chapter 7. In this chapter, I keep the focus on invariants of nonlinear functions.

In [TLS16] it was noted that all nonlinear invariants of an S-Box can be obtained from its cycle decomposition. Indeed, an invariant must take a constant value on each cycle. If an S-Box S has t cycles, then there are precisely 2^t invariants corresponding to 2^t possible assignments of the invariant values to all t cycles. A special kind of invariants was considered in [TLS16], where the output value of the invariant is allowed to be a negation of the input value., i.e. a function g such that $g(S(x)) = g(x) \oplus 1$. This case is only possible if all cycles of S have even length.

Quadratic invariants are interesting because they are preserved by an orthogonal linear layer. However, one can argue that any low-degree invariant is an interesting property, even if no orthogonal layer is used in the analyzed cipher. First, the algebraic degree can be seen as a measure of simplicity. Second, invariants may be used to generate equation systems of a cipher, and low-degree equations are generally easier to solve. Third, low-degree invariants are also an evidence that the analyzed component is non-ideal.

6.4.1 Low Degree Invariants from Cycle Decomposition

A naive approach

Let S be a permutation of $\mathbb{F}_2^n$ with t cycles. The most straightforward approach of finding all invariants $g\colon \mathbb{F}_2^n \to \mathbb{F}_2$ of S of degree at most d is to generate all 2^t invariants and check their algebraic degree. Generating the ANF of an n-bit Boolean function requires $n2^n$ operations. As a result, the complexity of this approach is $n2^{t+n}$. It is possible to compute a single ANF coefficient of a monomial degree $d+1$ in 2^{d+1} evaluations of the function. If the coefficient is equal to one, then the considered invariant has degree at least $d+1$ and can be excluded. Otherwise, other monomials must be checked. In the best case (when the first chosen coefficient is equal to 1), the complexity is 2^{t+d+1}.

An improved approach

The method can be improved by replacing the enumeration of all 2^t invariants by solving a linear algebra problem. Consider an affine subspace A of dimension $d+1$. Any possible invariant g of degree at most d must sum to zero over this subspace:

$$\bigoplus_{x \in A} g(x) = 0.$$

As g must be constant on each cycle of S, g can be defined by t unknowns - values of the invariant on each of the cycles. Let $c_1, \ldots, c_t \subseteq \mathbb{F}_2^n$ be cycles of S. Denote by $g_1, \ldots, g_t \in \mathbb{F}_2$ the unknowns, where g_i corresponds to the value of g on all elements from c_i. It follows that the sum of g over A can be expressed as a linear combination of g_i:

$$\bigoplus_{x \in A} g(x) = \bigoplus_{1 \le i \le t} \lambda_i g_i = 0, \text{ where}$$
$$\lambda_i = |A \cap c_i| \mod 2.$$

The improved approach is to generate enough linear equations on the unknowns g_i and solve the system.

Complexity. Generating one equation requires 2^{d+1} operations, assuming that the map $x \mapsto (i : x \in c_i)$ is efficient. This assumption can be implemented by a precomputation, which requires $\mathcal{O}(2^n)$ memory. This is a downside of this approach. $\mathcal{O}(t)$ equations are needed, and can be generated in time $\mathcal{O}(t2^{d+1})$. Solving the linear equation system requires $\mathcal{O}(t^3)$. Thus, the final time complexity is $\mathcal{O}(t2^{d+1} + t^3)$. In particular, all degrees (lower bounds) of invariants can be computed in time $\mathcal{O}(t2^n + t^3 n)$ using memory $\mathcal{O}(2^n)$. On practice, the algorithm is reasonably efficient for S-Boxes of sizes up to 32 bits and with a non-extremely large number of cycles. A parallelization is possible but requires a significant amount of memory.

Remark 14. Generating equations requires special care. If we choose a random affine subspace of dimension d, it is likely that only large cycles will be covered by the generated equation and we obtain no constraints on small cycles. Therefore, when generating an affine subspace, we ensure that multiple elements from the subspace lie in distinct cycles. This can be done by choosing basis vectors accordingly. For example, we can choose a cycle uniformly at random and then choose an element of the cycle uniformly at random.

Remark 15. The algorithm has a one-sided error possibility. If the generated linear system has no solutions (besides the constant ones $g = 0$ and $g = 1$), then surely there are no invariants of degree at most d. In this way, the algorithm provides a tool for *provable* security against low-degree invariants.

However, a solution to the system is not guaranteed to have degree at most d, only that it sums to 0 on generated affine spaces of dimension $d + 1$. I argue that we do not restrict the affine subspaces to be cosets of cubes, and an unrestricted affine subspace corresponds to a monomial coefficient in a random basis. In this way, even an invariant with a sparse ANF can be disproved to have a low degree with high probability, since in a random basis it may have a non-sparse ANF.

The pseudo-code of the proposed algorithm is given in Algorithm 6.1.

6.4.2 Cycle Decomposition of G from NORX8

States consisting of four equal columns lie on the cycles of F_{col} and F_{diag} functions that correspond directly to cycles of the G function. Indeed, such states

Algorithm 6.1 Low-Degree Invariants from Cycle Decomposition.

```
Input: partition $c_1, \dots, c_t \subseteq \mathbb{F}_2^n$ of $\mathbb{F}_2^n$ corresp. to the cycles of $S\colon \mathbb{F}_2^n \to \mathbb{F}_2^n$;
       an integer $d, 1 \le d \le n-1$.
Output: an invariant $g\colon \mathbb{F}_2^n \to \mathbb{F}_2$ of $S$ with possibility that $\deg g \le d$;
        or No invariants with $\deg g \le d$.
 1: $E \leftarrow$ an empty equation system
 2: for $i \in \{0, \dots, t+\epsilon\}$ do
 3:     $u \leftarrow$ a random element from $c_{1+(i \bmod t)}$
 4:     $V \leftarrow$ a random linear subspace of $\mathbb{F}_2^n$ of dimension $d+1$
        such that several $v \in V$ belong to distinct cycles.
 5:     $A \leftarrow u \oplus V$
 6:     for $j \in \{1, \dots, t\}$ do
 7:         $\lambda_j \leftarrow 0$
 8:     for $x \in A$ do
 9:         $\lambda_k \leftarrow \lambda_k \oplus 1$, where $k$ is such that $x \in c_k$
10:     $E \leftarrow E \cup (\bigoplus_{1 \le k \le t} \lambda_k g_k = 0)$
11: if $E$ has non-trivial solutions then
12:     $(g_1, \dots, g_t) \leftarrow$ a non-trivial solution of $E$
13:     $g \leftarrow (x \mapsto (g_i$ such that $x \in c_i))$
14:     return $g$
15: else
16:     return No invariants with $\deg g \le d$
```

always consist of four copies of a single column and applying F_{col} or F_{diag} to such state is equivalent to applying G to the corresponding column. For instance, it is possible to enumerate all cycles of the G function for NORX8, where G permutes $(\mathbb{F}_2^8)^4$. All these cycles of G can be transformed into cycles of F_{col} or F_{diag} by simply making 4 copies of the column. These cycles then are also cycles of F, except that all even cycles will split into two cycles each, because we need to consider cycles of G^2. I provide the cycle decomposition of $G\colon (\mathbb{F}_2^8)^4 \to (\mathbb{F}_2^8)^4$ from NORX8 in Table 6.2 by providing starting points and lengths of the cycles.

6.4.3 Low-Degree Invariants in G from NORX8

I applied the proposed approach to the function G from NORX8 permuting $\mathbb{F}_2^{32}$. In the previous section I described the cycle decomposition of G. There are 22 cycles and thus there are 2^{22} invariants of G. Note that there are cycles with odd length, therefore there exist no “switching” invariants of G.

The algorithm used around 36 gigabytes of memory and ran for 25 hours on a single 3.5GHz core. It generated 100 linear equations over 22 unknowns for each subspace dimension in $[1 \dots 32]$. Solving the systems took negligible time.

The results are as follows. For all dimensions $d \le 31$, there are no non-trivial invariants of degree $d-1$. For the dimension $d = 32$, there a space of dimension 21 of invariants of degree $d - 1 = 31$. It follows that the other coset of this space has invariants of degree 32. That is, a half of the invariants have degree 32 and the other half are of degree 31. Note that these are the maximum possible

Starting point	Cycle length	Starting point	Cycle length
(00, 00, 00, 01)	3294443807	(00, 00, 2B, 65)	399843
(00, 00, 00, 08)	621984749	(00, 02, 1C, 06)	52972
(00, 00, 00, 56)	212798071	(00, 00, 5C, 28)	23344
(00, 00, 00, 07)	56236016	(00, 00, 00, D5)	8301
(00, 00, 00, 06)	55712043	(00, 00, B8, D2)	6339
(00, 00, 00, 02)	21461014	(00, 05, 94, D3)	2124
(00, 00, 02, 29)	9062510	(01, 66, 26, D2)	848
(00, 00, 04, 52)	7374122	(00, 4E, 63, C1)	595
(00, 00, 00, 46)	7328319	(00, 9D, 2B, C3)	137
(00, 00, 08, 4E)	5608893	(03, 4F, 69, 6C)	78
(00, 00, 01, F7)	2463170	(00, 00, 00, 00)	1

Table 6.2: Cycles of G from NORX8. Starting points are of the form $(a, b, c, d) \in (\mathbb{F}_2^8)^4$ (see Figure 6.2).

degrees of invariants, since any pair of invariants of degree 32 must sum to an invariant of lower degree.

Furthermore, I searched for low-degree invariants of iterated G, i.e. G^l for $1 \leq l \leq 16$ and for particular values of l that lead to an increased number of cycles. Each cycle of G of length c splits into $gcd(l, c)$ cycles of G^l of lengths $c/gcd(l, c)$. Thus, the number of cycles grows and the number of invariants too. For example, G^8 has 54 cycles and thus, 2^{54} invariants. The results show that for $1 \leq l \leq 16$ and for $l \in \{24, 30, 32, 36, 51, 59\}$, all non-trivial invariants of G^l have degree at least 30. More detailed results are given in Table 6.3. Evaluation of G^l for a single l took the time proportional to the number of cycles in G^l.

I also verified correctness of the algorithm on a toy S-Box permutation of $\mathbb{F}_2^{16}$ with an invariant of degree 11. The algorithm successfully recovered this invariant.

I conclude that invariants of the mapping $G\colon (\mathbb{F}_2^8)^4 \to (\mathbb{F}_2^8)^4$ from NORX8 have maximum degree. Furthermore, invariants of G iterated for several rounds have also close to maximum degree. I remark that this observation does not rule out a possibly simple structure or property of those invariants. Indeed, the invariant subspace of dimension 2^{8w} shown in Section 6.3 corresponds to an invariant function of quite large degree $8w$. On the other hand, a low-degree invariant could correspond to a bit-level property of G, whereas the invariant subspace from Section 6.3 corresponds to a high-level structural property of the NORX's permutation.

function	$d \leq 29$	$d \leq 30$	$d \leq 31$	$d \leq 32 = \#cycles$
G^1	1	1	21	22
G^2	1	2	31	32
G^3	1	5	37	38
G^4	1	9	41	42
G^5	1	2	33	34
G^6	1	19	51	52
G^7	1	2	33	34
G^8	1	21	53	54
G^9	1	23	55	56
G^{10}	1	19	51	52
G^{11}	1	1	21	22
G^{12}	1	33	65	66
G^{13}	1	2	33	34
G^{14}	1	17	49	50
G^{15}	1	17	49	50
G^{16}	1	45	77	78
G^{24}	1	45	77	78
G^{30}	1	39	71	72
G^{32}	1	45	77	78
G^{36}	1	69	101	102
G^{51}	1	85	117	118
G^{59}	1	163	195	196

TABLE 6.3: Upper-bounds on dimensions of the spaces of the invariants of degree at most d of the function G from NORX8 iterated multiple times.

Chapter 7

Nonlinear Invariant-Preserving Linear Layers

In this chapter, I show a study of linear layers that preserve all low-degree invariants of a particular form. It is a generalization of *orthogonal* linear layers, which preserve all quadratic invariants, exploited in [TLS16]. Our main result is that for cubic invariants, there is no such *bijective* linear map that preserves all of them. However, we exhibit such *expanding* linear maps. We study the minimum expansion rate of these maps. This is a joint work with Christof Beierle and Alex Biryukov and it is currently in a process of submission to a Boolean functions journal.

7.1 Introduction

After the introduction of *linear cryptanalysis* in [Mat93] as a powerful method to attack symmetric cryptographic primitives, people started studying how to generalize this method in order to exploit *nonlinear approximations* for cryptanalysis, see, e.g., [HKM95] and [KR96]. While it might be easier to find a nonlinear approximation over parts of the primitive, e.g., over an S-box of small size, a crucial problem in nonlinear cryptanalysis is to find nonlinear approximations that hold true for the *whole round function* of the primitive. An example that exploits nonlinear approximations that are preserved over the whole round function is *bilinear cryptanalysis* over Feistel ciphers [Cou04].

More recently, an interesting solution for the above problem was described by Todo, Leander and Sasaki in [TLS16] for round functions that can be described in terms of an LS-design [GLSV14]. Let one round of a substitution-permutation cipher operating on n S-boxes of t-bit length be given as depicted in Figure 7.1 and let the linear layer $L^{(t)}\colon \mathbb{F}_2^{nt} \to \mathbb{F}_2^{nt}$ only XOR the outputs of the S-boxes, i.e., each $(y_1, \ldots, y_n)$ for $y_j \in \mathbb{F}_2^t$ is mapped to $(z_1, \ldots, z_n)$ where $z_j = \sum_{i=1}^n \alpha_{i,j} y_i$ for particular $\alpha_{i,j} \in \mathbb{F}_2$. In that case, $L^{(t)}$ can be defined by t parallel applications of the matrix L given by

$$L = \begin{bmatrix} \alpha_{1,1} & \alpha_{1,2} & \cdots & \alpha_{1,n} \\ \alpha_{2,1} & \alpha_{2,2} & \cdots & \alpha_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ \alpha_{n,1} & \alpha_{n,2} & \cdots & \alpha_{n,n} \end{bmatrix} .$$

Todo *et al.* observed that if L is orthogonal, then for *any* t-bit Boolean function f of algebraic degree less than or equal to 2 and for any $y_1, \ldots, y_n \in \mathbb{F}_2^t$ it is

$$f(y_1) + f(y_2) + \cdots + f(y_n) = f(z_1) + f(z_2) + \cdots + f(z_n) . \tag{7.1}$$

This fact was used to successfully cryptanalyze the block ciphers Midori, Scream and iScream in a weak key setting. Indeed, if f is any invariant function for the S-box S, i.e., if for all $x \in \mathbb{F}_2^t$, $f(x) = f(S(x))$, and if $\deg(f) \leq 2$, one obtains an invariant function for the whole round according to Equation 7.1.

An interesting question is whether the property of L being orthogonal is also necessary for Equation 7.1 to hold for all f with degree upper-bounded by 2. More generally, we would like to understand the necessary and sufficient properties of the linear layer that preserve such invariants in the case when $\deg(f) \leq d$ for $d > 2$. Although the existence of a non-trivial[1] linear layer for which Equation 7.1 holds for all f with $\deg f \leq d$ is totally unclear, such a construction would be of significant interest. On the one hand, it would deepen the knowledge on how to design strong symmetric cryptographic primitives and to avoid possible attacks and could on the other hand be useful in order to design symmetric *trapdoor ciphers* to be used as public-key schemes, see, e.g., [RP97, PG97, BBF16]. The idea would be to hide a nonlinear approximation as the trapdoor information. If the linear layer is designed such that it

[1]By *non-trivial*, we mean that the matrix of L is not a permutation matrix.

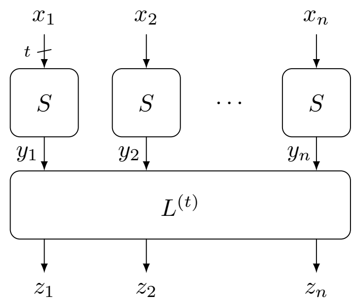


FIGURE 7.1: The round function of a substitution-permutation cipher based on an LS-design.

preserves *all invariants* of a special form, e.g., all functions of degree at most d, the specification of the linear layer would not leak more information on the particular invariant and thus on the trapdoor. There could also be applications besides cryptography, so the above problem might be of independent interest.

7.1.1 Our Contribution

In this work we answer the above question and consider the case of $L \in \mathbb{F}_2^{n\times m}$, i.e., the number of outputs might be different than the number of inputs. We precisely characterize the matrices that preserve *all* invariants of the form similar as given in Equation 7.1, i.e.,

$$f(y_1) + \cdots + f(y_n) = f(z_1) + \cdots + f(z_m) + f(0) \cdot (m + n \mod 2) \ , \tag{7.2}$$

where the degree of f is upper bounded by d. We call such matrices *degree-d sum-invariant.* We show that such matrices correspond to n-bit Boolean functions of degree at most $n - d - 1$ which admit no linear annihilators. We call the *supports* of such Boolean functions *degree-d zero-sum sets of rank n.* This characterization is obtained in Proposition 7.6, Proposition 7.8 and Proposition 7.13. Our results imply that $m \geq n$ and, for the case of $d = 2$, the property of L being (semi-)orthogonal is not only sufficient, but also necessary. Moreover, we obtain an interesting characterization of orthogonal matrices over $\mathbb{F}_2$, i.e., $L \in \mathbb{F}_2^{n\times n}$ is orthogonal if and only if in every $2 \times 2n$ submatrix of $\left[\begin{array}{c|c} I_n & L \end{array}\right]$, each column occurs an even number of times.

Besides showing the link between degree-d zero-sum sets and degree-d sum-invariant matrices, we study degree-d zero-sum sets of full rank in more detail. We are in particular interested in the smallest of such sets. Let $F(n, d)$ denote the minimum number of elements in a degree-d zero-sum set of rank n. The following theorem summarizes our main results.

Theorem 7.1. *Let $n, d \in \mathbb{Z}_+$ with $n > d \geq 1$. Then the following properties of $F(n, d)$ hold.*

(i) $F(n, d) = \min\{\mathbf{wt}(g) \mid g \in \mathcal{BF}_{n,n-d-1}\backslash\{0\}$ *with g having at most 1 affine annihilator*$\}$.

(ii) $F(n, 1) = n + 2 - (n \mod 2)$ *and, for $n = 4$ or $n > 5$,* $F(n, 2) = 2n$.

As exceptions, $F(3,2) = 8$ *and* $F(5,2) = 12$*.*

(iii) $F(d+1,d) = F(d+2,d) = 2^{d+1}$*. Moreover,* $F(d+3,d) = 3 \cdot 2^d$ *and* $F(2d+4,d) = 2^{d+2}$*. For* $d+4 \leq n \leq 2d+3$*,*

$$F(n,d) = 2^{2d-n+4}(2^{n-d-2}-1)\ .$$

(iv) for any fixed d*, the sequence* $F(n,d)$ *is increasing, i.e.,* $F(n+1,d) \geq F(n,d)$*.*

(v) for $n_1, n_2 > d$*, the inequality*

$$F(n_1+n_2,d) \leq F(n_1,d) + F(n_2,d)$$

holds. Moreover, for $d \geq 2$*, it is*

$$F(n+d,d-1) \leq F(n,d) \leq 2F(n-1,d-1)\ .$$

The last inequality implies that, for $n \geq 4$*,* $F(n,3) \geq 2n+6$*.*

We prove the above values by providing a construction of the corresponding zero-sum sets (resp. Boolean functions). In case where we only prove an upper bound, we provide a construction that meets this bound. Table 7.1 shows the values and bounds for $F(n,d)$ for $n \leq 30$ and $d \leq 10$.

The last inequality in Theorem 7.1 implies that any degree-d sum-invariant matrix $L \in \mathbb{F}_2^{n\times n}$ for $d \geq 3$ must be a permutation matrix, i.e. an invertible matrix with exactly n ones. In other words, the observation of Todo *et al.* cannot be extended for higher-degree invariants without L being expanding.

7.1.2 Organization

In Section 7.2, I fix notation specific to this chapter. I also recall the observations made in [TLS16] with regard to orthogonal matrices and the preservation of degree-2 invariants. For motivating the remainder of the chapter, I directly present an example construction of an expanding linear mapping that preserves higher-degree invariants.

In Section 7.3, I show equivalent characterizations of degree-d zero-sum sets and explain the links between degree-d sum-invariant matrices and degree-d zero-sum sets.

Minimal degree-d zero-sum sets are studied in Section 7.4, where Theorem 7.1 is proven. I further summarize the implications to degree-d sum-invariant matrices in Section 7.5. Finally, the chapter is concluded in Section 7.6.

7.2 Preliminaries

In this chapter I use a specific notation to simplify expressions. A vector $v \in \mathbb{F}_2^n$ is considered a row vector. Any matrix $L \in \mathbb{F}_2^{n \times m}$ defines a linear mapping $\varphi\colon \mathbb{F}_2^n \to \mathbb{F}_2^m,\ x \mapsto xL$.

7.2.1 Higher-Order Derivatives, Affine Equivalence and Algebraic Immunity of Boolean Functions

Boolean functions have several applications in cryptography, e.g., for designing stream ciphers. In order to resist algebraic attacks, the notion of algebraic immunity was introduced in 2004 as follows.

Definition 7.2 (Algebraic immunity [MPC04])**.** *Let $f\colon \mathbb{F}_2^n \to \mathbb{F}_2$. An n-bit Boolean function $g \neq 0$ is called an* annihilator *of f, if $fg = 0$. The set of annihilators of f together with $g = 0$ form a vector space, denoted by $\mathrm{AN}(f)$. We denote by $\mathrm{AN}_d(f)$ the subspace of annihilators of f with algebraic degree at most d together with the zero-function. The* algebraic immunity *of f, denoted $\mathrm{AI}(f)$, is defined as the minimum k for which $\mathrm{AN}_k(f) \cup \mathrm{AN}_k(f+1) \neq \{0\}$.*

An important concept for Boolean function is the notion of affine equivalence.

Definition 7.3 (Domain Affine Equivalence)**.** *Two Boolean functions $f, g\colon \mathbb{F}_2^n \to \mathbb{F}_2$ are called* domain affine equivalent *if there exists a linear bijection $\varphi\colon \mathbb{F}_2^n \to \mathbb{F}_2^n$ and a vector $c \in \mathbb{F}_2^n$ such that $g = f \circ (\varphi + c)$. If $c = 0$, f and g are called* linear equivalent*.*

I remark that, in the literature, *domain affine equivalence* of Boolean functions is called simply *affine equivalence*. I specify the term to avoid ambiguity, as, for example, g and $g \oplus 1$ are not domain affine equivalent in general. It is well known that the weight, the algebraic degree and the dimensions of the annihilator spaces (and thus the algebraic immunity) are invariant under domain affine equivalence.

7.2.2 Orthogonal Matrices and Preservation of Nonlinear Invariants

In [TLS16], Todo, Leander and Sasaki introduced the *nonlinear invariant attack* and successfully distinguished the block ciphers Midori, Scream and iScream from a random permutation for a significant fraction of weak keys. For an n-bit permutation $G\colon \mathbb{F}_2^n \to \mathbb{F}_2^n$, the main idea consists in finding a non-constant n-bit Boolean function f and a constant $\varepsilon \in \mathbb{F}_2$ such that

$$\forall x \in \mathbb{F}_2^n\colon f(x) = f(G(x)) + \varepsilon.$$

Such a function f is called an *invariant* for G. In order to find an invariant for the cipher, Todo *et al.*. observed that if $L \in \mathbb{F}_2^{n \times n}$ is an orthogonal matrix, i.e., if $\langle xL, yL\rangle = \langle x, y\rangle$ for all $x, y \in \mathbb{F}_2^n$, then for all Boolean functions $f \in \mathcal{BF}_{t,2}$ it is

$$\forall X \in \mathbb{F}_2^{t\times n}\colon \bigoplus_{i=1}^{n} f\big((X^\top)_i\big) = \bigoplus_{j=1}^{n} f\big(((XL)^\top)_j\big). \tag{7.3}$$

In other words, *any* Boolean function $f\colon \mathbb{F}_2^t \to \mathbb{F}_2$ of algebraic degree at most 2 gives rise to an invariant over the linear layers of Midori, Scream and iScream of the form $(x_1, \ldots, x_n) \mapsto f(x_1) + \ldots f(x_n)$, where n denotes the number of S-boxes, t denotes the bit length of the S-box and $x_i \in \mathbb{F}_2^t$.

We illustrate this from a slightly different point of view on the example of the linear layer used in Midori (see [BBI+15]), which is defined by the following matrix:

$$L = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix}. \tag{7.4}$$

It is easy to see that L is orthogonal. Thus, according to Equation 7.3, for *any* $f \in \mathcal{BF}_{t,2}$ and all $x_1, x_2, x_3, x_4 \in \mathbb{F}_2^t$, the following equation holds:

$$\begin{aligned}&f(x_1) + f(x_2) + f(x_3) + f(x_4) = \\ &f(x_2 + x_3 + x_4) + f(x_1 + x_3 + x_4) + f(x_1 + x_2 + x_4) + f(x_1 + x_2 + x_3).\end{aligned}$$

Consider an alternative way of proving this. The arguments of f form an affine subspace of dimension 3, namely

$$x_1 + \operatorname{span}(x_1 + x_2, x_1 + x_3, x_1 + x_4).$$

Therefore, the equation is equivalent to

$$\delta_{x_1+x_2}\delta_{x_1+x_3}\delta_{x_1+x_4} f(x_1) = 0,$$

which is clearly true for any $f \in \mathcal{BF}_{t,2}$ and any x_1, x_2, x_3, x_4 since all third-order derivatives of a quadratic function are equal to zero. This observation gives new insights on how to generalize the linear layer in order to preserve *higher-degree invariants.*

Proposition 7.4. *Let $d \geq 2$ be an integer. Then there exists a matrix $L \in \mathbb{F}_2^{n\times m}$ with $n = d + 2, m = 2^{d+1} - d - 2$ and full rank n such that for any $t \geq 1$ and any $f \in \mathcal{BF}_{t,d}$, the following property holds:*

$$\forall X \in \mathbb{F}_2^{t\times n}\colon \bigoplus_{i=1}^{n} f\big((X^\top)_i\big) = \bigoplus_{j=1}^{m} f\big(((XL)^\top)_j\big). \tag{7.5}$$

An example of such L is given by a matrix with columns taken as all vectors from $\mathbb{F}_2^n$ with an odd Hamming weight greater or equal to 3.

Proof. For any $t \geq 1$ and any $x_0, \ldots, x_{d+1} \in \mathbb{F}_2^t$ consider the $(d+1)$-dimensional affine subspace

$$V = x_0 + \operatorname{span}(x_0 + x_1, x_0 + x_2, \ldots, x_0 + x_{d+1}) \ .$$

For any Boolean function f of degree d, any $(d+1)$-th derivative vanishes. Therefore, $\bigoplus_{v \in V} f(v) = 0$. This can be equivalently written as

$$f(x_0) + f(x_1) + \ldots + f(x_{d+1}) = \tag{7.6}$$

$$= \bigoplus_{\substack{I \subseteq \{1,\ldots,d+1\} \\ |I| \geq 2 \text{ even}}} f(x_0 + \bigoplus_{i \in I} x_i) + \bigoplus_{\substack{I \subseteq \{1,\ldots,d+1\} \\ |I| \geq 3 \text{ odd}}} f(\bigoplus_{i \in I} x_i) \tag{7.7}$$

$$= \bigoplus_{\substack{I \subseteq \{0,\ldots,d+1\} \\ |I| \geq 3 \text{ odd}}} f(\bigoplus_{i \in I} x_i). \tag{7.8}$$

The right-hand side contains $2^{d+1} - d - 2$ applications of f. Let Y be the set of the linear functions defining the arguments of f in the right-hand side of Equation 7.6, i.e.,

$$Y = \left\{ \bigoplus_{i \in I} x_i \;\middle|\; I \subseteq \{0, \ldots, d+1\},\ |I| \geq 3 \text{ odd} \right\},$$

and let L be the matrix of the linear function that maps $(x_0, x_1, \ldots, x_{d+1})$ to $(y_1, y_2, \ldots, y_{2^{d+1}-d-2})$, where $y_i \in Y$ and all y_i are pairwise different. Then, Equation 7.6 is equivalent to Equation 7.3 with the described L.

Since $m \geq n \geq 4$, any unit vector from $\mathbb{F}_2^n$ can be expressed a linear combination of 3 columns of L, e.g.,

$$(1, 0, 0, 0, \ldots, 0) = (1, 1, 1, 0, \ldots, 0) + (1, 0, 1, 1, \ldots, 0) + (1, 1, 0, 1, \ldots, 0).$$

We conclude that L has full rank n. □

Example 2. For $d = 2$ we obtain the orthogonal matrix given in Equation 7.4. For $d = 3$ we obtain an expanding linear mapping $\varphi : \mathbb{F}_2^5 \to \mathbb{F}_2^{11}$ defined by the following 5×11 matrix L:

$$L = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{bmatrix}.$$

7.3 Degree-d Zero-Sum Sets and Sum-Invariant Matrices

A natural question to ask is which other linear mappings have a similar property as given in Equation 7.5. To answer this question, we study *degree-d zero-sum* *sets* as a generalization of the above problem.

Definition 7.5 (Degree-d Zero-Sum Set)**.** *Let $S \subseteq \mathbb{F}_2^n$ and let $d \in \mathbb{N}$. We call S* degree-d zero-sum *if, for all $f \in \mathcal{BF}_{n,d}$,*

$$\bigoplus_{s \in S} f(s) = 0. \tag{7.9}$$

We define $\operatorname{rank}(S)$ *to be the maximum number of linearly independent elements in S and denote by $\mathrm{ZS}^d_{n\times m}$ the set of degree-d zero-sum sets with m elements and rank n.*

We first show the following equivalent characterizations of degree-d zero-sum sets.

Proposition 7.6. *Let $S = \{s_1, \dots, s_k\} \subseteq \mathbb{F}_2^n$ and let $d \in \mathbb{Z}_+$. Let $\mathbb{M}_S \in \mathbb{F}_2^{n\times k}$ be any matrix (up to a permutation of the columns) the columns of which correspond to the elements of S, i.e.,*

$$\mathbb{M}_S = \left[\ s_1^\top \mid \dots \mid s_k^\top\ \right].$$

Then the following statements are equivalent:

(i) *S is a degree-d zero-sum set.*

(ii) *k is even and, for any choice of d (not necessarily distinct) rows $r_1, \dots, r_d$ of $\mathbb{M}_S$, it is $\langle r_1, \dots, r_d\rangle = 0$.*

(iii) *in every $d \times k$ submatrix of $\mathbb{M}_S$, each column occurs an even number of times.*

(iv) $\deg(\mathbb{1}_S) \le n - d - 1$.

(v) *for all $t \geq 1$ and all $f \in \mathcal{BF}_{t,d}$, $\forall X \in \mathbb{F}_2^{t\times n}$: $\bigoplus_{s\in S} f(sX^\top) = 0$.*

In particular, the degree-d zero-sum sets in $\mathbb{F}_2^n$ are exactly the supports of the n-bit Boolean functions of degree at most $n-d-1$. Therefore, any non-empty degree-d zero-sum set must contain at least 2^{d+1} elements.

Proof. To prove $(i) \Rightarrow (ii)$, let

$$\mathbb{M}_S = \begin{bmatrix} r_1 \\ \hline \vdots \\ \hline r_n \end{bmatrix}$$

with $r_i \in \mathbb{F}_2^k$. Let $l_1, \dots, l_d$ be d (not necessarily distinct) row indices and consider the monomial function $f \in \mathcal{BF}_{n,d}$, $x \mapsto \prod_{i=1}^d x_{l_i}$, which has degree d. From Equation 7.9, it must be

$$0 = \bigoplus_{s\in S} f(s) = \bigoplus_{s \in S} \prod_{i=1}^{d} s_{l_i} = \langle r_{l_1}, \dots, r_{l_d}\rangle\,.$$

Clearly, k must be even because $\bigoplus_{s\in S} 1 = 0$.

$(ii) \Rightarrow (iii)$: We first see that any $1 \times k$ submatrix of $\mathbb{M}_S$ contains each element in $\mathbb{F}_2$ an even number of times. Indeed, let r be any row in $\mathbb{M}_S$. From (ii) we know that $\mathbf{wt}(r) \mod 2 = \langle r \rangle = 0$ and thus r contains an even number of 1's. Because k is even, it must also contain an even number of 0's. We now use induction on the number of rows. Let $d' < d$ such that $(ii) \Rightarrow (iii)$ holds for d'. Let us choose an arbitrary $(d'+1) \times k$ submatrix $H = [m_{i,j}]_{1\le i\le d'+1, 1\le j\le k}$ of $\mathbb{M}_S$. We define $H^{(0)} := [m^{(0)}_{i,j}]$ to be the submatrix of H that is obtained by selecting exactly the columns $m_{\star,j}$ of H for which $m_{d'+1,j} = 0$. Similarly, let $H^{(1)} := [m^{(1)}_{i,j}]$ be the submatrix of H that is obtained by selecting exactly the columns $m_{\star,j}$ of H for which $m_{d'+1,j} = 1$. We have already seen from the initial step that both $H^{(0)}$ and $H^{(1)}$ must contain an even number of columns (otherwise the row $m_{d'+1,\star}$ would have an odd weight). From (ii), we know that

$$\begin{aligned}0 = \langle m_{1,\star}, \dots, m_{d',\star}, m_{d'+1,\star}\rangle &= \left\langle m^{(0)}_{1,\star}, \dots, m^{(0)}_{d'+1,\star}\right\rangle + \left\langle m^{(1)}_{1,\star}, \dots, m^{(1)}_{d'+1,\star}\right\rangle \\ &= \left\langle m^{(1)}_{1,\star}, \dots, m^{(1)}_{d',\star}\right\rangle = \left\langle m^{(0)}_{1,\star}, \dots, m^{(0)}_{d',\star}\right\rangle .\end{aligned}$$

Because of the induction hypothesis, $H^{(0)}$ and $H^{(1)}$ contain each column an even number of times and therefore, every column of H occurs an even number of times.

$(iii) \Rightarrow (iv)$: Let $u \in \mathbb{F}_2^n$ with $\mathbf{wt}(u) \ge n - d$. Because of (iii),

$$|\{s \in S \mid s \preceq u\}|$$

is even (because d zeroes in positions i where $u_i = 0$ occur an even number of times among elements of S). It follows that

$$|\{s \in S \mid s \preceq u\}| \mod 2 = \bigoplus_{s\preceq u} \mathbb{1}_S(s) = 0$$

and thus, the monomial x^u doesn't occur in the ANF of $\mathbb{1}_S$. Since this holds for all u with $\mathbf{wt}(u) \ge n - d$, the algebraic degree of $\mathbb{1}_S$ is at most $n - d - 1$.

$(iv) \Rightarrow (v)$: Let $f \in \mathcal{BF}_{t,d}$ be an arbitrary function of degree at most d. Observe that

$$\forall X \in \mathbb{F}_2^{t\times n} \quad \bigoplus_{s\in\mathbb{F}_2^n} \mathbb{1}_S \cdot f(sX^\top) = 0, \tag{7.10}$$

because $\deg \mathbb{1}_S \cdot (f \circ X) \le \deg \mathbb{1}_S + \deg f \le n - 1$. Here, $f \circ X$ denotes the n-bit Boolean function $s \mapsto f(sX^\top)$. Equation 7.10 can equivalently be written as

$$\forall X \in \mathbb{F}_2^{t\times n} \quad \bigoplus_{s\in S} f(sX^\top) = 0,$$

which proves (v). The implication $(v) \Rightarrow (i)$ follows by letting $t = n$ and $X = I_{n\times n}$.

To see that any non-empty degree-d zero-sum set contains at least 2^{d+1}

elements, we use the fact that any non-zero Boolean function of degree at most $n-d-1$ has a weight at least $2^{n-(n-d-1)} = 2^{d+1}$. □

It is worth remarking that the property of being degree-d zero-sum is invariant under the application of an injective linear mapping. Indeed, if $\varphi\colon \operatorname{span}(S) \to \mathbb{F}_2^{n'}$ is an injective linear function on the subspace $\operatorname{span}(S)$ of dimension $\operatorname{rank}(S)$, then $|\varphi(S)| = |S|$ and if S is degree-d zero-sum, so is $\varphi(S)$. Further, $\operatorname{rank}(\varphi(S)) = \operatorname{rank}(S)$. Therefore, without loss of generality, we can represent a zero-sum set $S \in \mathrm{ZS}^d_{n\times m}$ as a subset of $\mathbb{F}_2^n$ and given by the columns of an $n \times m$ matrix $\mathbb{M}_S$ of the form

$$\mathbb{M}_S = \left[\ I_{n\times n} \mid L\ \right] \tag{7.11}$$

for an $L \in \mathbb{F}_2^{n\times(m-n)}$. We say that a zero-sum set (resp. a matrix $\mathbb{M}_S$) given in the representation of Equation 7.11 is in *systematic form*. We are in particular interested in the properties of such matrices L that define zero-sum sets in $\mathrm{ZS}^d_{n\times m}$ in the above way. For instance, such an L can only exist if m is even. We generalize this by introducing the notion of a *degree-d sum-invariant matrix* as follows.

Definition 7.7 (Degree-d Sum-Invariant Matrix)**.** *A matrix $L \in \mathbb{F}_2^{n\times m}$ is called* degree-d sum-invariant *if, for all $t \geq 1$ and all $f \in \mathcal{BF}_{t,d}$,*

$$\forall X \in \mathbb{F}_2^{t\times n}\colon \bigoplus_{i=1}^{n} f\big((X^\top)_i\big) = \bigoplus_{j=1}^{m} f\big(((XL)^\top)_j\big) + \varepsilon_{m+n} f(0), \tag{7.12}$$

where $\varepsilon_{m+n} = (m+n) \mod 2$.

Proposition 7.8. *Let $L \in \mathbb{F}_2^{n\times m}$ be a linear mapping and let $d \in \mathbb{N}$. Then the following statements are equivalent:*

(i) L is degree-d sum-invariant.

(ii) The columns of the matrix $\widehat{\mathbb{M}_L}$ occurring with odd multiplicity define a degree-d zero-sum set, where

$$\begin{cases} \widehat{\mathbb{M}_L} := \left[\ I_{n\times n} \mid L\ \right] \in \mathbb{F}_2^{n\times(m+n)}, & \text{if } m+n \text{ is even}\,; \\ \widehat{\mathbb{M}_L} := \left[\ I_{n\times n} \mid L \mid 0\ \right] \in \mathbb{F}_2^{n\times(m+n+1)}, & \text{if } m+n \text{ is odd}\,. \end{cases} \tag{7.13}$$

(iii) For all $x_1, \dots x_d \in \mathbb{F}_2^n$ it is $\langle x_1, \dots, x_d\rangle = \langle x_1L, \dots, x_dL\rangle$.

Moreover, if L fulfills (i) and if $d \geq 2$, then $n \leq m$, $LL^\top = I_n$ and L must have full rank n.

Proof. We first prove $(i) \Rightarrow (ii)$. If $m+n$ is even, then Equation 7.12 is equivalent to

$$\forall X \in \mathbb{F}_2^{t\times n}\colon \bigoplus_{i=1}^{n} f\big(e_i X^\top\big) + \bigoplus_{j=1}^{m} f\big((L^\top)_j X^\top\big) = 0, \tag{7.14}$$

where e_i denotes the i-th unit vector. If there is a j for which $(L^\top)_j$ is equal to a unit vector e_k, then $f((L^\top)_j X^\top) = f(e_k X^\top)$ and the two terms cancel in Equation 7.14. Similarly, if there exist two different j_1, j_2 such that $(L^\top)_{j_1} = (L^\top)_{j_2}$, then $f((L^\top)_{j_1} X^\top)$ and $f((L^\top)_{j_2} X^\top)$ cancel out. This is another way of saying that the columns of the matrix $\widehat{\mathbb{M}_L} = \left[\ I_{n\times n} \mid L\ \right]$ occurring with odd multiplicity define a degree-d zero-sum set.

If $m+n$ is odd, then $\varepsilon_{m+n} = 1$ and Equation 7.12 can be written as

$$\forall X \in \mathbb{F}_2^{t\times n}\colon \bigoplus_{i=1}^{n} f\big(e_i X^\top\big) + \bigoplus_{j=1}^{m} f\big((L^\top)_j X^\top\big) + f(0X^\top) = 0.$$

This is equivalent to say that the columns of the $n \times (m+n+1)$ matrix $\widehat{\mathbb{M}_L} = \left[\ I_{n\times n} \mid L \mid 0\ \right]$ occurring with odd multiplicity define a degree-d zero-sum set.

$(ii) \Rightarrow (iii)$. If the columns of $\widehat{\mathbb{M}_L}$ occurring with odd multiplicity define a degree-d zero sum set, then, because of Proposition 7.6, any d (not necessarily distinct) rows $\left[\ e_{l_1} \mid L_{l_1}\ \right], \dots, \left[\ e_{l_d} \mid L_{l_d}\ \right]$ of $\widehat{\mathbb{M}_L}$ fulfill

$$\big\langle \left[\ e_{l_1} \mid L_{l_1}\ \right], \dots, \left[\ e_{l_d} \mid L_{l_d}\ \right] \big\rangle = 0\ ,$$

which is equivalent to

$$\langle e_{l_1}, \dots, e_{l_d} \rangle = \langle e_{l_1} L, \dots, e_{l_d} L \rangle\ .$$

Because of the linearity of the inner product, i.e.,

$$\langle x_1 + x_1', x_2, \dots, x_d \rangle = \langle x_1, x_2, \dots, x_d \rangle + \langle x_1', x_2, \dots, x_d \rangle\ ,$$

the statement follows.

$(iii) \Rightarrow (i)$. If there are $f_1, f_2 \in \mathcal{BF}_{t,d}$ such that Equation 7.12 holds for both f_1 and f_2, then it clearly holds for $f_1 + 1$ and for $f_1 + f_2$ as well. Therefore, without loss of generality, let $f \in \mathcal{BF}_{t,d}$ be a monomial function, i.e., $f(z) = \prod_{k=1}^{d} z_{l_k}$ for $1 \le l_1 \le \dots \le l_d \le t$. Let $X \in \mathbb{F}_2^{t\times n}$. Then,

$$\bigoplus_{i=1}^{n} f((X^\top)_i) = \bigoplus_{i=1}^{n} \prod_{k=1}^{d} (X^\top)_{i,l_k} = \langle X_{l_1}, \dots, X_{l_d} \rangle$$

and

$$\bigoplus_{j=1}^{m} f(((XL)^\top)_j) + \varepsilon_{m+n} f(0) = \bigoplus_{j=1}^{m} \prod_{k=1}^{d} ((XL)^\top)_{j,l_k} = \langle X_{l_1} L, \dots, X_{l_d} L \rangle\ .$$

It follows that if L preserves all generalized inner products of d elements, then L is degree-d sum-invariant.

If L fulfills the equivalent statements (i) - (iii), then, for all $x, y \in \mathbb{F}_2^n$, it is

$$xy^\top = \langle x, y \rangle = \langle xL, yL \rangle = xL(yL)^\top = xLL^\top y\ .$$

It follows that $LL^\top$ must be the identity and thus, L must have full rank n. $\square$

This result shows a relation between degree-d sum-invariant matrices and semi-orthogonal matrices. A matrix $L \in \mathbb{F}_2^{n\times m}$ with $n \leq m$ is called *semi-orthogonal* if $LL^\top = I_{n\times n}$. Indeed, we have shown that a matrix is degree-2 sum-invariant if and only if it is semi-orthogonal.[2] Because of the above relation, the degree-$(d+1)$ sum-invariant matrices might also be called *d-th order semi-orthogonal*.

The invertible semi-orthogonal matrices are exactly the *orthogonal* matrices and the orthogonal matrices in dimension n form a multiplicative group, called the *orthogonal group*. With the above equivalences, we obtain an interesting characterization of the orthogonal groups over $\mathbb{F}_2$.

Corollary 7.9. *A matrix $L \in \mathbb{F}_2^{n\times n}$ is orthogonal if and only if in each $2 \times 2n$ submatrix of $\left[\, I_{n\times n} \mid L \,\right]$, each column occurs an even number of times.*

7.3.1 Relation to Orthogonal Arrays

Proposition 7.6 points out a relation between degree-d zero-sum sets and orthogonal arrays.

Definition 7.10 (Orthogonal Array [HSS99])**.** *An $m \times n$ matrix M with entries from a finite set of cardinality k is said to be an* orthogonal array with k levels, strength d and index λ, *denoted $OA(m, n, k, d)$, if every $m \times d$ submatrix of M contains each d-tuple exactly λ times as a row. Without loss of generality, we will assume that M is a matrix with elements in $\mathbb{Z}_k$.*

For our purposes we are only interested in the case of $k = 2$. We directly obtain the following.

Corollary 7.11. *Let $S \subseteq \mathbb{F}_2^n$. If $\mathrm{M}_S^\top$ is an $OA(|S|, n, 2, d)$ such that 2^{d+1} divides $|S|$ (i.e., if the index λ is even), then S is a degree-d zero-sum set.*

As an example, for $d = 3$, there is a well-known construction of orthogonal arrays from Hadamard matrices (see [HSS99, pp. 145–148]). A *Hadamard matrix* of order n is a matrix $H \in \mathbb{Z}^{n\times n}$ which can only take values in $\{-1, 1\}$ and which fulfills $H^\top H = nI_{n\times n}$. For a matrix M with elements in $\{-1, 1\}$, we denote by $\widetilde{M}$ the $\mathbb{F}_2$ matrix obtained from M by replacing -1 with 0, i.e., we define $\widetilde{M}$ to be the result of $\frac{1}{2}(M + 1)$, interpreted in $\mathbb{F}_2$.

If H is a Hadamard matrix of order $8k$ for $k \in \mathbb{Z}_+$, it is well known that

$$\widetilde{\begin{bmatrix} H \\ \hline -H \end{bmatrix}}$$

is an $OA(16k, 8k, 2, 3)$ of even index (see [HW78, Theorem 4.16]). Therefore, it defines a degree-3 zero-sum set $S \subseteq \mathbb{F}_2^{8k}$ with $16k$ elements. However, its rank can be at most $4k$ (see [PRV06, Proposition 2]) and we are interested in the zero-sum sets of full rank.

[2]We only consider matrices with $n \leq m$. If $L \in \mathbb{F}_2^{n\times m}$ with $n > m$, L would be defined to be semi-orthogonal if $L^\top L = I_m$. Then, L is semi-orthogonal if and only if $L^\top$ is degree-2 sum-invariant.

7.4 Minimal and Maximal Zero-Sum Sets

In this section we study zero-sum sets of particular rank n and prove results on their existence. We are particularly interested in the smallest of such sets, defined in the following sense.

Definition 7.12. *We denote by $F(n,d)$ the minimum number $m \in \mathbb{Z}_+$ for which there exists an $S \in \mathrm{ZS}^d_{n\times m}$. We call a zero-sum set* minimal *if it is contained in $\mathrm{ZS}^d_{n\times F(n,d)}$. Analogously, a zero-sum set $S \in \mathrm{ZS}^d_{n\times m}$ is called* maximal *if $\mathrm{ZS}^d_{n'\times m} = \varnothing$ for all $n' > n$.*

Note that $F(n,d)$ is only defined if $n > d$ as otherwise, the only degree-d zero-sum set in $\mathbb{F}_2^n$ is the empty set. We first characterize the zero-sum sets of particular rank n in terms of Boolean functions.

7.4.1 Relations between Zero-Sum Sets and Affine Annihilators of Boolean Functions

The first three existence results are presented in Proposition 7.13, Proposition 7.14 and Proposition 7.15 and outline the link between zero-sum sets and the dimensions of degree-1 annihilator spaces of Boolean functions.

Proposition 7.13. *There exists a degree-d zero-sum set $S \in \mathrm{ZS}^d_{n\times m}$ if and only if there exists a Boolean function $h \in \mathcal{BF}_{n,n-d-1}$ with $\mathbf{wt}(h) = m$ and $\dim \mathrm{AN}_1(h) \leq 1$.*

Proof. Let us assume that $S \in \mathrm{ZS}^d_{n\times m}$ is given in systematic form, i.e., it can be represented as in Equation 7.11. Then, $S = \mathrm{supp}(h)$ for a Boolean function $h \in \mathcal{BF}_{n,n-d-1}$ for which $\forall i \in \{1,\dots,n\} : h(e_i) = 1$. Such a function cannot have a linear annihilator and therefore, any $a \in \mathrm{AN}_1(h) \setminus \{0\}$ must be of the form $a = \ell + 1$ for a linear Boolean function ℓ. It follows that $\dim AN_1(h) \leq 1$.

Let now $h \in \mathcal{BF}_{n,n-d-1}$ with $\mathbf{wt}(h) = m$ and $\dim \mathrm{AN}_1(h) \leq 1$. Let $a \in \mathrm{AN}_1(h) \setminus \{0\}$. If $a = \ell + 1$ for a linear function ℓ, then h has no linear annihilator. If a is linear, we fix a constant $c \in \mathbb{F}_2^n$ for which $a(c) = 1$ and consider the function $h_c \colon x \mapsto h(x + c) \in \mathcal{BF}_{n,n-d-1}$ which is domain affine equivalent to h and thus has the same weight. It is easy to verify that $a + 1$ is an affine annihilator for h_c. Because the dimensions of the annihilator spaces are invariant under domain affine equivalence, h_c has no linear annihilators. Therefore, without loss of generality, we can assume that h has no linear annihilator. Let $S = \mathrm{supp}(h) \subseteq \mathbb{F}_2^n$ be the support of h and consider a matrix $\mathbb{M}_S$ the columns of which form exactly the set S. Since h has no linear annihilator, there is no linear combination of rows of $\mathbb{M}_S$ that is equal to zero. We conclude that $\mathbb{M}_S$ has full rank n and $S \in \mathrm{ZS}^d_{n\times m}$. □

Proposition 7.14. *Given a function $h \in \mathcal{BF}_{n,n-d-1}$ with $\mathbf{wt}(h) = m$ and $\mathrm{AN}_1(h) = \{0\}$, it is possible to construct a zero-sum set in $\mathrm{ZS}^d_{(n+1)\times m}$.*

Proof. Consider the function

$$h' : \mathbb{F}_2^{n+1} \to \mathbb{F}_2, (x_1,\dots,x_{n+1}) \mapsto x_{n+1} h(x_1,\dots,x_n)\ .$$

Note that h' has degree at most $n-d$. Further, h' has no linear annihilator. Otherwise, by setting $x_{n+1} = 1$, we would obtain that h has an annihilator of algebraic degree 1, contradicting $\mathrm{AN}_1(h) = \{0\}$. By Proposition 7.13, we can construct $S \in \mathrm{ZS}^d_{(n+1)\times m}$. □

The converse statement is true for maximal zero-sum sets.

Proposition 7.15. *Let $n \geq 2$ and let $S \in \mathrm{ZS}^d_{(n+1)\times m}$ be maximal. Then, $\mathbb{1}_S$ is domain linear equivalent to a function $h \in \mathcal{BF}_{n+1,n-d}$ of the form*

$$h(x_1, \dots, x_{n+1}) = x_{n+1} \cdot g(x_1, \dots, x_n), \tag{7.15}$$

where $g \in \mathcal{BF}_{n,n-d-1}$ with $\mathbf{wt}(g) = \mathbf{wt}(h) = m$ and $\mathrm{AN}_1(g) = \{0\}$. Further, if $m < 2^{n-1}$, then $\mathrm{AI}(g) \geq 2$.

Proof. Let $\mathbb{M}_S$ be a matrix which columns correspond to the elements of S. Because S is maximal, the vector subspace of $\mathbb{F}_2^m$ spanned by the rows of $\mathbb{M}_S$ must contain the all-1 vector $I_n := (1, 1, \dots, 1)$. Otherwise, one would obtain a zero-sum set in $\mathrm{ZS}^d_{(n+2)\times m}$ defined by the matrix

$$\left[\begin{array}{c} \mathbb{M}_S \\ \hline I_n \end{array}\right].$$

Therefore, we can apply a linear permutation A on the columns of $\mathbb{M}_S$ such that $\mathbb{1}_{A(S)} = h$ where $h \in \mathcal{BF}_{n+1,n-d}$ is of the form as given in Equation 7.15 with $g \in \mathcal{BF}_{n,n-d-1}$ and $\mathbf{wt}(g) = \mathbf{wt}(h)$. It is left to show that $\mathrm{AN}_1(g) = \{0\}$.

Clearly, g cannot have a linear annihilator. We assume now that g has an annihilator of degree 1 of the form $(x_1, \dots, x_n) \mapsto 1 + \bigoplus_{i=1}^n a_i x_i$. Then, $g(x) = 0$ for all x with $\bigoplus_{i=1}^n a_i x_i = 0$. Let j be such that $a_j = 1$. For the linear permutation $Q : \mathbb{F}_2^n \to \mathbb{F}_2^n$, $Q(x_1, \dots, x_n) = (x_1, \dots, x_{j-1}, \bigoplus_{i=1}^n a_i x_i, x_{j+1}, \dots, x_n)$, we have

$$g(Q(x_1, \dots, x_n)) = x_j \cdot g'(x_1, \dots, x_{j-1}, x_{j+1}, \dots, x_n)$$

for a function $g' \in \mathcal{BF}_{n-1,n-d-2}$. But this means that h is linear-equivalent to a function of the form $(x_1, \dots, x_{n+1}) \mapsto x_{n+1} \cdot x_n \cdot g'(x_1, \dots, x_{n-1})$, which has a linear annihilator $x_{n+1} + x_n$. We get a contradiction and conclude that $\mathrm{AN}_1(g) = \{0\}$.

If $m < 2^{n-1}$, it is easy to see that $g + 1$ cannot admit an annihilator of algebraic degree 1. Suppose that $a \in \mathrm{AN}_1(g+1) \setminus \{0\}$. Then, $\mathbf{wt}(a) = 2^{n-1}$ and $ag = a$, which is impossible. □

As Proposition 7.15 only holds for maximal zero-sum sets we cannot use it to establish an equivalence between minimal degree-d zero-sums of rank $n+1$ and n-bit Boolean functions of degree $n-d-1$ with algebraic immunity at least 2 and minimum weight. We therefore propose the following question:

Question 7.16. *Let $S \in \mathrm{ZS}^d_{n\times F(n,d)}$ be minimal. What are necessary and sufficient conditions for S to be maximal?*

7.4.2 Minimal Zero-Sum Sets: Bounds and Values for $F(n, d)$

In order to derive values for $F(n, d)$, we basically have to study the Boolean functions that admit at most one annihilator of algebraic degree 1 and find those of minimum weight. Indeed, from Proposition 7.13, we know that

$$F(n, d) = \min\{\mathbf{wt}(g) \mid g \in \mathcal{BF}_{n,n-d-1} \setminus \{0\} \text{ with } \dim \mathrm{AN}_1(g) \leq 1\}.$$

For $d = 1$ and $d = 2$ we can easily determine the cardinalities of minimal degree-d zero-sum sets, as stated in Proposition 7.17 and Proposition 7.18. The proofs also provide a construction for a minimal zero-sum set. While the proof for $d = 1$ is rather trivial, the proof for $d = 2$ relies on the relation between degree-2 zero-sum sets and semi-orthogonal matrices.

Proposition 7.17. *For $n \geq 2$, $F(n, 1) = n + 2 - (n \mod 2)$.*

Proof. Consider a zero-sum set $S \in \mathrm{ZS}^1_{n\times m}$ and its matrix in systematic form. Each row must have an even weight, therefore there must be at least one extra column besides the identity part, i.e. $m \geq n + 1$. By setting the extra column to the all-one vector I_n we make all rows to have even weight. Furthermore, m must be even and we may also need to add the all-zero column. The proposition follows. □

Proposition 7.18. *For $n = 4$ and for $n > 5$, it is $F(n, 2) = 2n$. Further, $F(3, 2) = 8$ and $F(5, 2) = 12$.*

Proof. Let $n \geq 3$ and m be minimal such that there exists an $S \in \mathrm{ZS}^2_{n\times m}$. Let further $L \in \mathbb{F}_2^{n\times(m-n)}$ such that S is in systematic form with $\mathbb{M}_S = \left[\ I_{n\times n} \mid L\ \right]$. As $\mathbb{M}_S$ cannot contain any repeated columns, it is $\mathbb{M}_S = \widehat{\mathbb{M}_L}$ and thus, L must be semi-orthogonal and $n \leq (m - n)$. It follows that $F(n, 2) = m \geq 2n$.

Let now $n = 4$ or $n \geq 6$. To prove the existence of an $S \in \mathrm{ZS}^2_{n\times 2n}$, we observe that if $L \in \mathbb{F}_2^{n\times n}$ is an orthogonal matrix for which each column has weight larger than 1, $\widehat{\mathbb{M}_L}$ defines a degree-2 zero-sum set of size $2n$ and rank n according to Proposition 7.8. It is left to show that, for any dimension $n = 4$ or $n \geq 6$, there exists an orthogonal matrix for which no column corresponds to a unit vector. We are going to distinguish four cases. Let us define the orthogonal matrices M_4 and M_6 as

$$M_4 = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix}, \quad M_6 = \begin{bmatrix} 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 \end{bmatrix}.$$

Case 1 ($n = 0 \mod 4$): The block-diagonal matrix $\mathrm{diag}(M_4, \ldots, M_4)$ which contains M_4 as its diagonal blocks is orthogonal and each column weight is equal to 3.

Case 2 ($n = 2 \mod 4$): Because $n > 5$, it is $n = 4k + 6$ for $k \geq 0$ and the matrix $\mathrm{diag}(M_6, M_4, M_4, \dots, M_4)$ is orthogonal and each column has weight at least 3.

Case 3 ($n = 3 \mod 4$): Because $n > 5$, it is $n = 4k + 3$ for $k \geq 1$ and the two matrices $D_1 = \mathrm{diag}(1, 1, 1, M_4, M_4, \dots, M_4)$ and $D_2 = \mathrm{diag}(M_4, 1, 1, \dots, 1)$ are orthogonal. Their product is orthogonal and of the form

$$D_1 D_2 = \left[\begin{array}{cccc|cccc} 0 & 1 & 1 & 1 & 0 & 0 & \dots & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & \dots & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & \dots & 0 \\ \hline \multicolumn{4}{c|}{\mathrm{A}} & \multicolumn{4}{c}{\mathrm{D}} \end{array}\right], \tag{7.16}$$

where D is the $4k \times (4k - 1)$ submatrix of $\mathrm{diag}(M_4, \dots, M_4)$ omitting the first column. It is obvious that each column has weight at least 3.

Case 4 ($n = 1 \mod 4$): Because $n > 5$, it is $n \geq 9$ and $n = 4k + 6 + 3$ for $k \geq 0$. The two matrices $D_1 = \mathrm{diag}(1, 1, 1, M_6, M_4, \dots, M_4)$ and $D_2 = \mathrm{diag}(M_4, 1, 1, \dots, 1)$ are orthogonal. Their product is orthogonal and of the form given in Equation 7.16 with D as the $4k + 6 \times (4k + 6 - 1)$ submatrix of $\mathrm{diag}(M_6, M_4, M_4, \dots, M_4)$ omitting the first column. It is obvious that each column has weight at least 3.

For $n = 3$ we use that any degree-d zero-sum set must contain at least 2^{d+1} elements. Thus, $F(n, 2) \geq 8$. We obtain $F(3, 2) = 8$ because $\mathbb{F}_2^3$ is a degree-2 zero-sum set.

For $n = 5$, assume that there exists an orthogonal matrix $L \in \mathbb{F}_2^{5 \times 5}$ which does not have a unit vector as its row (or column). From point (*iii*) of Proposition 7.6 it follows that any 2×5 submatrix of L must contain an odd number of columns equal to each of $(0, 1), (1, 0), (0, 0)$ and an even number of columns equal to $(1, 1)$ (same applies for rows of any 5×2 submatrix of L). It follows that, up to a permutation of rows, L has the following form:

$$L = \begin{bmatrix} 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & . & . & . \\ 1 & 1 & . & . & . \\ 1 & 1 & . & . & . \end{bmatrix}. \tag{7.17}$$

It is easy to see that it is not possible to complete this matrix such that all 2×5 and 5×2 submatrices satisfy the condition. Therefore, $F(5, 2) > 10$. Moreover, it is easy to verify that

$$\mathbb{M}_S = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix}$$

defines a zero-sum set in $\mathrm{ZS}^2_{5 \times 12}$, thus $F(5, 2) = 12$. $\square$

Proposition 7.19 below presents a simple way to construct a $d + 1$ zero-sum set of rank $n+1$ from a degree-d zero-sum set of rank n. This construction might be used to derive an upper bound on $F(n,d)$.

Proposition 7.19. *If there exists an $S \in \mathrm{ZS}^d_{n\times m}$, one can construct a zero-sum set $S' \in \mathrm{ZS}^{d+1}_{(n+1)\times 2m}$. In particular, for $n > d+1$, $F(n,d) \le 2F(n-1,d-1)$.*

Proof. If $S \in \mathrm{ZS}^d_{n\times m}$, then the columns of the matrix

$$\begin{bmatrix} 0 \dots 0 & | & 1 \dots 1 \\ \mathbb{M}_S & | & \mathbb{M}_S \end{bmatrix}$$

define a degree-$(d+1)$ zero-sum set S' with $2m$ elements of rank $n+1$. We remark that both sets S and S' have essentially the same indicator function, only the domain dimension is different. $\square$

Note that the upper bound on $F(n,d)$ given by this construction is not always tight. Let $S \subseteq \mathbb{F}_2^9$ be such that $\mathbb{1}_S(x) = x_1(x_2x_3x_4x_5 + x_6x_7x_8x_9)$. It easy to verify that $S \in \mathrm{ZS}^3_{9\times 30}$. It follows that $F(9,3) \le 30 \ne 2F(8,2) = 32$. The corresponding matrix $\mathbb{M}_S$ is given by:

$$\mathbb{M}_S = \begin{bmatrix}
0&0&0&0&0&0&0&0&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1 \\
0&0&0&0&1&1&1&1&0&0&0&0&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1 \\
0&0&1&1&0&0&1&1&0&0&1&1&0&0&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1 \\
0&1&0&1&0&1&0&1&0&1&0&1&0&1&0&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1 \\
1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&0&0&0&0&0&0&0&0&1&1&1&1&1&1&1 \\
1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&0&0&0&0&1&1&1&1&0&0&0&0&1&1&1 \\
1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&0&0&1&1&0&0&1&1&0&0&1&1&0&0&1 \\
1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&0&1&0&1&0&1&0&1&0&1&0&1&0&1&0 \\
1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1
\end{bmatrix}. \qquad (7.18)$$

Proposition 7.20. *For any $d \in \mathbb{Z}_+$ and $n_1, n_2 > d$, $F(n_1+n_2,d) \le F(n_1,d) + F(n_2,d)$.*

Proof. If $S_1 \in \mathrm{ZS}^d_{n_1\times m_1}$, $S_2 \in \mathrm{ZS}^d_{n_2\times m_2}$, then the columns of the matrix

$$\mathbb{M}_S = \left[\begin{array}{c|c} & 0 \dots 0 \\ \mathbb{M}_{S_1} & \vdots \\ & 0 \dots 0 \\ \hline 0 \dots 0 & \\ \vdots & \mathbb{M}_{S_2} \\ 0 \dots 0 & \end{array}\right]$$

repeating an odd number of times define a degree-d zero-sum set S with at most m_1+m_2 elements of rank n_1+n_2. More precisely, if both S_1 and S_2 contain the zero vector, then the resulting zero-sum set has size m_1+m_2-2 due to the zero-vector being cancelled by the repetition. Otherwise, S has size m_1+m_2. $\square$

Proposition 7.21. *Let $d \ge 2$. If there exist an $S \in \mathrm{ZS}^d_{n\times m}$, one can construct a zero-sum set in $\mathrm{ZS}^{d-1}_{(n+d)\times m}$. In particular, for $n > d$, $F(n,d) \ge F(n+d,d-1)$.*

Proof. Let $\mathbb{M}_S = \left[\begin{array}{c|c} I_{n\times n} & L \end{array}\right]$ be a matrix for S in systematic form. By re-ordering the rows of $\mathbb{M}_S$, one can bring it into the form

$$\left[\begin{array}{c|c|c|c} 1\ldots 1 & 1 & 0\ldots 0 & 0\ldots 0 \\ A & 0 & B & I_{(n-1)\times(n-1)} \end{array}\right], \tag{7.19}$$

where $A \in \mathbb{F}_2^{(n-1)\times m_1}$ and $B \in \mathbb{F}_2^{(n-1)\times m_2}$ for some m_1, m_2 with $m_1+m_2+n = m$. Moreover, m_1 cannot be zero because the first row must have an even weight. We see that $\left[\begin{array}{c|c} A & 0 \end{array}\right]$ must define a degree-$(d-1)$ zero-sum set in $\mathbb{F}_2^{n-1}$, i.e., $\left[\begin{array}{c|c} A & 0 \end{array}\right] = \mathbb{M}_T$ for a $T \in \mathrm{ZS}^{d-1}_{r\times(m_1+1)}$. This is simply because the Hadamard (component-wise) product of any $d-1$ rows of $\left[\begin{array}{c|c} A & 0 \end{array}\right]$ can be expressed as the Hadamard (component-wise) product of d rows of $\mathbb{M}_S$, i.e., the $d-1$ rows at the same positions as those of $\left[\begin{array}{c|c} A & 0 \end{array}\right]$ and the first row $[11\ldots100\ldots0]$. We conclude that $m_1 = |T| \geq 2^d$ and thus, $r \geq d$.

Let $v_1, \ldots, v_d$ be d linearly independent rows of A and consider the matrix

$$\left[\begin{array}{c|c|c|c} 1\ldots 1 & 1 & 0\ldots 0 & 0\ldots 0 \\ A & 0 & B & I_{(n-1)\times(n-1)} \\ v_1 & 0 & 0\ldots 0 & 0\ldots 0 \\ v_2 & 0 & 0\ldots 0 & 0\ldots 0 \\ \vdots & \vdots & \vdots & \vdots \\ v_d & 0 & 0\ldots 0 & 0\ldots 0 \end{array}\right],$$

which must define a zero-sum set in $\mathrm{ZS}^{d-1}_{(n+d)\times m}$ by the same argument as above, i.e., the Hadamard product of any $d-1$ rows can be expressed as the Hadamard product of d rows of $\mathbb{M}_S$. It is also easy to see that no linear combination of rows can be equal to zero, i.e. the constructed set has full rank $n+d$. □

Using the above result and Proposition 7.18, we can prove a lower bound on $F(n,3)$ as follows.

Corollary 7.22. *For $n \geq 4$ it is $F(n,3) \geq 2n+6$.*

So far, we were able to characterize the minimal degree-d zero-sum sets for $d=1$ and $d=2$ and proved some inequalities for the general case. Further, we can use the following classification theorem by Kasami, Tokura and Azumi in order to derive some more exact values of $F(n,d)$.

Theorem 7.23 ([KT70, KTA76]). *Let $r \geq 2$ and let $f \in \mathcal{BF}_{n,r}$ with $\mathbf{wt}(f) < 2^{n-r+1}$. Then f is domain affine equivalent to either (i) or (ii), where*

(i) $f = x_1 \ldots x_{r-2}(x_{r-1}x_r + x_{r+1}x_{r+2} + \ldots + x_{r+2\ell-3}x_{r+2\ell-2}), n \geq r+2\ell-2$

(ii) $f = x_1 \ldots x_{r-\ell}(x_{r-\ell+1} \ldots x_r + x_{r+1} \ldots x_{r+\ell}), r \geq \ell, n \geq r+\ell$.

A direct application leads to the following results.

Proposition 7.24 (Values of $F(n,d)$ for $n \leq 2d+4$).

(i) $F(d+1,d) = 2^{d+1}$.

(ii) $F(d+2,d) = 2^{d+1}$ *and the minimal zero-sum sets in* $\mathbb{F}_2^{d+2}$ *correspond to the Boolean functions of algebraic degree* 1.

(iii) $F(d+3,d) = 3 \cdot 2^d$ *and the minimal zero-sum sets in* $\mathbb{F}_2^n$ *correspond to the Boolean functions domain affine equivalent to* $x \mapsto x_1x_2 + x_3x_4$.

(iv) For $d+4 \leq n \leq 2d+3$, $F(n,d) = 2^{2d-n+4}(2^{n-d-2}-1) = \mathbf{wt}(h_{n,d})$, *where*

$$r = n-d-1, h_{n,d}\colon (x_1,\dots,x_n) \mapsto x_1(x_2x_3\dots x_r + x_{r+1}x_{r+2}\dots x_{2r-1}) \; .$$

(v) $F(2d+4,d) = 2^{d+2} = \mathbf{wt}(g_d)$, *where:*

$$g_d\colon (x_1,\dots,x_{2d+4}) \mapsto x_1(x_2x_3\dots x_{d+3} + (x_2+1)x_{d+4}x_{d+5}\dots x_{2d+4}) \; .$$

Proof. For $d \in \mathbb{Z}_+, d < n$, let us define the set

$$S_{n,d} \coloneqq \{g \in \mathcal{BF}_{n,d} \setminus \{0\} \text{ with } \dim \mathrm{AN}_1(g) \leq 1\} \; .$$

From Proposition 7.13 we know that $F(n,d) = \min\{\mathbf{wt}(g) \mid g \in S_{n,n-d-1}\}$. Therefore, we trivially obtain $F(d+1,d) = 2^{d+1}$. $S_{d+2,1}$ is the set of Boolean functions of algebraic degree 1 and thus $F(d+2,d) = 2^{d+1}$.

To obtain the minimum weight of functions in $S_{d+3,2}$, we first note that every Boolean function of algebraic degree 2 of the minimum weight 2^{d+1} must be domain affine equivalent to a monomial function, i.e., $x \mapsto x_1x_2$ (see Proposition 12 of [Car07]). As this monomial function admits the annihilators $x \mapsto x_1+1$ and $x \mapsto x_2+1$, the minimum weight in $S_{d+3,d}$ must be at least $2^{d+2} - 2^d$ (see, e.g., [Car07, p. 70] for the possible weights of quadratic Boolean functions). This weight is obtained by the function $x \mapsto x_1x_2 + x_3x_4$, which clearly is in $S_{d+3,2}$. To see that all other functions in $S_{d+3,2}$ of minimal weight are domain affine equivalent to it, it is enough to see that all of the functions

$$q_{n,\ell}\colon (x_1,\dots,x_n) \mapsto x_1x_2 + x_3x_4 + \dots + x_{2\ell-1}x_{2\ell}$$

with $\ell \geq 3$ have a strictly larger weight. Indeed, by induction on ℓ, it can be easily shown that $\mathbf{wt}(q_{n,\ell}) = 2^{n-1} - 2^{n-\ell-1}$.

Let now $d+4 \leq n \leq 2d+3$. It is easy to see that $h_{n,d} \in S_{n,n-d-1}$. Further, its weight can be computed as

$$\mathbf{wt}(h_{n,d}) = 2^{d+1} + 2^{d+1} - 2^{2d-n+4} = 2^{2d-n+4}(2^{n-d-2}-1) \; .$$

It is left to show that $h_{n,d}$ is an element of minimum weight in $S_{n,n-d-1}$. Let therefore be h' in $S_{n,n-d-1}$ with $\mathbf{wt}(h') \leq \mathbf{wt}(h_{n,d})$. Since $\mathbf{wt}(h_{n,d}) < 2^{n-(n-d-1)+1} = 2^{d+2}$, the assumptions of Theorem 7.23 are fulfilled and h' would be domain affine equivalent to one of the forms given in cases (i) and (ii) of Theorem 7.23. If $n \geq d+5$, Case (i) corresponds to a Boolean function of the form $x \mapsto x_1x_2g$ which admits $x \mapsto x_1+1$ and $x \mapsto x_2+1$ as degree-1 annihilators. For $n = d + 4$, Case (i) corresponds to a function of the form

$$x \mapsto x_1(x_2x_3 + x_4x_5 + \cdots + x_{2\ell}x_{2\ell+1}) = x_1 g$$

for $g \in S_{n,2}$ and, therefore, its weight must be at least $2^{n-2} - 2^{n-4} = 2^{2d-n+4}(2^{n-d-2} - 1)$.

Otherwise, h' must be domain affine equivalent to one of the functions given in Case (ii). Since it cannot admit two annihilators of algebraic degree 1, it must be domain affine equivalent to either

$$x \mapsto x_1(x_2x_3 \dots x_r + x_{r+1}x_{r+2} \dots x_{2r-1}) = h_{n,d},$$

or

$$g_{n,d} \colon x \mapsto x_1x_2 \dots x_r + x_{r+1}x_{r+2} \dots x_{2r},$$

where $r = n - d - 1$. As

$$\mathbf{wt}(g_{n,d}) = 2^{2d-n+3}(2^{n-d-1} - 1) > \mathbf{wt}(h_{n,d}) = 2^{2d-n+3}(2^{n-d-1} - 2),$$

the point (iv) follows.

It is easy to see that $\mathbf{wt}(g_d) = 2^{d+2}$, i.e. $F(2d + 4, d) \leq 2^{d+2}$. By Proposition 7.19 and (iv) of this Theorem, $F(2d+4, d) \geq F(2d+5, d+1)/2 = (2^{d+2} - 1)$. Since $F(2d + 4, d)$ has to be even, the Theorem follows. □

We are now going to show that, for any fixed d, the sequence $F(n, d)$ is increasing with n. For that, we need the following lemma.

Lemma 7.25. *For $n > 2d + 3$, we have $F(n, d) \leq \frac{2^n}{n+1}$.*

Proof. By repeatedly applying Proposition 7.19, we obtain

$$F(n, d) \leq 2^{d-1}(n - d + 2) = 2^n \frac{n - d + 2}{2^{n-d+1}} .$$

It is left to show that $\frac{n-d+2}{2^{n-d+1}} \leq \frac{1}{n+1}$. We know that

$$(n+1)(n-d+2) < (2n-2d-2)(n-d+2) = 2(n-d-1)(n-d+2) \leq 2^{n-d+1} ,$$

which is true for $n - d \geq 5$. The latter is guaranteed by $n \geq 2d + 4$ and $d \geq 1$. This proves the statement. □

Proposition 7.26. *For $n > d + 1$, it is $F(n, d) \geq F(n - 1, d)$.*

Proof. We prove this statement by induction on d. If $d = 1$ and $d = 2$, the statement is obviously true by Proposition 7.17 and Proposition 7.18. Let thereby $d \geq 3$ and assume that the statement is true for $d - 1$.

Let $S \in \mathrm{ZS}^d_{n\times m}$ be a minimal zero-sum set, i.e., $m = F(n, d)$, such that $\mathbb{M}_S$ can be given as in Equation 7.19 for $A \in \mathbb{F}_2^{(n-1)\times m_1}$ and $B \in \mathbb{F}_2^{(n-1)\times m_2}$ with m_1, m_2 such that $m_1+m_2+n = m$. Let $m' := m_2+n-1$. We see that $[B|I_{(n-1)\times(n-1)}]$ must define a degree-$(d-1)$-zero-sum set in $\mathbb{F}_2^{n-1}$, i.e., $[B|I_{(n-1)\times(n-1)}] = \mathbb{M}_T$ for a $T \in \mathrm{ZS}^{d-1}_{(n-1)\times m'}$. This is because every $(d - 1) \times (m')$ submatrix of $\mathbb{M}_T$ must occur an even number of times (from the property of S being a degree-d

zero-sum set) and, since $\mathbb{M}_T$ contains $I_{(n-1)\times(n-1)}$, it must have rank $n-1$. We now distinguish two cases.

Case 1 ($m' \leq \frac{m}{2}$): In that case we directly obtain

$$m = F(n,d) \geq 2F(n-1,d-1) \geq 2F(n-2,d-1) \geq F(n-1,d) ,$$

where the second estimation follows from the induction hypothesis and the last one follows from Proposition 7.19.

Case 2 ($m' > \frac{m}{2}$): We first remark that if $n \leq 2d+3$, the statement directly follows from Proposition 7.24. For example, for $n \geq d+5$,

$$F(n,d) = 2^{d+2} - 2^{2d-n+4} \geq 2^{d+2} - 2^{2d-n+5} = F(n-1,d) .$$

Let us therefore assume that $n > 2d+3$. Note that in the matrix $\mathbb{M}_S$, we can add the first row $[11\dots100\dots0]$ to any other row and would obtain an equivalent zero-sum set. This operation does not change the right part of $\mathbb{M}_S$ containing $I_{(n-1)\times(n-1)}$. Indeed, it allows us to obtain a zero-sum set $S_c \in \mathrm{ZS}^d_{n\times m}$ represented by

$$\mathbb{M}_{S_c} = \left[\begin{array}{c|c|c|c} 1\dots1 & 1 & 0\dots0 & 0\dots0 \\ A+c^\top & c^\top & B & I_{(n-1)\times(n-1)} \end{array}\right]$$

for any $c \in \mathbb{F}_2^{n-1}$. Let us denote by R the set of columns of A together with the $(n-1)$-bit zero vector. Our statement to prove follows if we can guarantee the existence of a vector $\tilde{c}$ such that, for all $v \in (R+\tilde{c}^\top)$, $\mathbf{wt}(v) \geq 2$. Then, we would obtain a zero-sum set in $\mathrm{ZS}^d_{(n-1)\times m''}$ defined by

$$\left[\begin{array}{c|c|c|c} A+\tilde{c}^\top & \tilde{c}^\top & B & I_{(n-1)\times(n-1)} \end{array}\right]$$

as there won't be any cancellation between $[A+\tilde{c}^\top \mid \tilde{c}^\top]$ and $I_{(n-1)\times(n-1)}$, thus keeping the rank maximum. Indeed, such a vector must always exist. Assume that, for all $c \in \mathbb{F}_2^{n-1}$, there exists a $v \in (R+\tilde{c}^\top)$ with weight at most 1. This is equivalent to say that the covering radius of the set $R \subseteq \mathbb{F}_2^{n-1}$ is equal to 1. By a simple counting argument it follows that $|R| \geq \frac{2^{n-1}}{n}$. On the other hand, it is

$$|R| = m - m' < F(n,d) - \frac{F(n,d)}{2} = \frac{1}{2}F(n,d) \leq \frac{2^{n-1}}{n+1} ,$$

where the last inequality follows from the previous lemma. We get a contradiction, therefore such vector $\tilde{c}$ always exists. □

7.5 Implications for Degree-d Sum-Invariant Matrices

In this section, I point out the implications of the above results on degree-d sum-invariant matrices. The most interesting implication is that any bijective degree-3 sum-invariant matrix must be trivial. As the linear layer of a block cipher based on an LS-design certainly has to be bijective, this shows that one cannot extend the observation of Todo *et al.*. to invariants of degree higher than two.

Corollary 7.27. *Let $L \in \mathbb{F}_2^{n\times n}$ be a degree-d sum-invariant matrix for $d \geq 3$. Then L must be a permutation matrix.*

Proof. Let us assume a degree-3 sum-invariant matrix $L \in \mathbb{F}_2^{n\times n}$ and let $\widehat{\mathbb{M}_L}$ be given by

$$\widehat{\mathbb{M}_L} = \left[\, I_{n\times n} \,\middle|\, L \,\right] \in \mathbb{F}_2^{n\times 2n} .$$

By Proposition 7.8 the columns of $\widehat{\mathbb{M}_L}$ occurring an odd number of times correspond to a degree-3 zero-sum set $S \subseteq \mathbb{F}_2^n$. Note that the unit columns of $I_{n\times n}$ do not repeat inside $I_{n\times n}$. Therefore, after removing the even occurrences of each column, the number of columns left in $I_{n\times n}$ will be not smaller than the number of columns left in L. It follows that $\operatorname{rank}(S) \geq |S|/2$. From Corollary7.22,

$$|S| \geq F(\operatorname{rank}(S), 3) \geq 2 \cdot \operatorname{rank}(S) + 6$$

Therefore, S must be empty and thus L is a permutation matrix. □

Consider a degree-d sum-invariant matrix L and consider the matrix $\widehat{\mathbb{M}_L}$ defined as in Proposition 7.8:

$$\begin{cases} \widehat{\mathbb{M}_L} := \left[\, I_{n\times n} \,\middle|\, L \,\right] \in \mathbb{F}_2^{n\times(m+n)}, & \text{if } m+n \text{ is even;} \\ \widehat{\mathbb{M}_L} := \left[\, I_{n\times n} \,\middle|\, L \,\middle|\, 0 \,\right] \in \mathbb{F}_2^{n\times(m+n+1)}, & \text{if } m+n \text{ is odd,} \end{cases} \tag{7.20}$$

where it is shown that the columns of $\widehat{\mathbb{M}_L}$ occurring and odd number of times define a degree-d zero-sum set. Because of the cancellations, the size and the rank of the zero-sum set may be lower. We deduce the following decomposition of sum-invariant matrices.

Proposition 7.28. *Let $L \in \mathbb{F}_2^{n\times m}$ be a degree-d sum-invariant matrix such that no column of L is equal to zero. Then, up to permutations of rows and columns, L can be expressed in the following form:*

$$L = \left[\, A \,\middle|\, \begin{array}{c} 0 \\ \hline I_k \end{array} \,\middle|\, M \,\middle|\, M \,\right], \tag{7.21}$$

where k, t are some integers, $M \in \mathbb{F}_2^{n\times t}$, $A \in \mathbb{F}_2^{n\times(m-2t-k)}$, and the columns of A do neither contain unit vectors nor repetitive columns. Such integers k, t are unique. Consider the matrix $\widehat{A}$:

$$\begin{cases} \widehat{A} := \left[\, \begin{array}{c} I_{n-k} \\ \hline 0 \end{array} \,\middle|\, A \,\right] \in \mathbb{F}_2^{n\times(m+n-2t-2k)}, & \text{if } m+n \text{ is even;} \\ \widehat{A} := \left[\, \begin{array}{c} I_{n-k} \\ \hline 0 \end{array} \,\middle|\, A \,\middle|\, 0 \,\right] \in \mathbb{F}_2^{n\times(m+n-2t-2k+1)}, & \text{if } m+n \text{ is odd.} \end{cases} \tag{7.22}$$

The columns of the matrix $\widehat{A}$ are pairwise distinct and form a degree-d zero-sum set.

Proof. The columns of $\widehat{\mathbb{M}_L}$ occurring an odd number of times form a degree-d zero-sum set. The columns of $I_{n\times n}$ may only cancel with columns from L. Let k be the number of unit vectors occurring an odd number of times in L. Let A be the matrix consisting of the columns of L that are repeated an odd number of times and which are not unit vectors. It follows that L can be expressed in the form given in Equation 7.21. Now consider the matrix $\widehat{\mathbb{M}_L}$. After removing even repetitions of columns, the matrix will be equal to $\widehat{A}$. It follows that the columns of $\widehat{A}$ define a degree-d zero-sum set.

To show uniqueness of k, t, first recall that A must not contain unit vectors. It follows that all columns of L occurring an even number of times must be in M, and all columns occurring an odd number of times must be either in A or in I_k depending only on the column weight.

□

7.5.1 Minimum Expansion Rate

We have shown that for $d \geq 3$, there exist no bijective degree-d sum-invariant matrices. However, there exist rectangular degree-d sum-invariant matrices resulting in expanding linear mappings. A natural problem would be to find a degree-d sum-invariant matrix with a minimum expansion rate.

Definition 7.29 (Expansion Rate)**.** *The* expansion rate *of a matrix $L \in \mathbb{F}_2^{n\times m}$ is the ratio $\frac{m}{n}$.*

Note that, given a degree-d sum-invariant matrix $L \in \mathbb{F}_2^{n\times m}$, we can always build a a degree-d sum-invariant matrix in $\mathbb{F}_2^{(n+1)\times(m+1)}$ of the form

$$\begin{bmatrix} L & 0 \\ 0 & 1 \end{bmatrix}.$$

Therefore, by repetitively extending any matrix L by unit vectors in the above way, we can construct a matrix with an expansion rate arbitrarily close to 1. Indeed, the permutation matrices have an expansion rate of exactly 1. Therefore, by the *minimum expansion rate* for a degree-d sum-invariant matrix of fixed d, we refer to the minimum expansion rate over all degree-d sum-invariant matrices that do not contain a unit vector as a column.

It is clear that for $d = 2$ the minimum expansion rate is 1 and is achieved by orthogonal matrices. For $d \geq 3$ the minimum expansion rate is an open problem. It corresponds to the minimum value of $\frac{F(n,d)}{n} - 1$. Among the established values of $F(n, d)$ the minimum expansion rate is achieved for $F(d + 2, d) = 2^{d+1}$, i.e. by the matrices from the construction given in Proposition 7.4. We conjecture that this is indeed the optimal expansion rate.

Conjecture 7.30. *Let $d \geq 3$. The minimum expansion rate of a degree-d sum-invariant matrix is equal to $\frac{2^{d+1}-d-2}{d+2}$.*

7.6 Conclusion and Open Problems

In the work I described in this chapter we have revealed the precise properties of the linear layer used in LS-designs that allow to preserve nonlinear invariants of a similar form than those observed by Todo *et al.*. As a negative result, we have shown that it is not possible to construct such an LS-design block cipher that generalizes the invariants to be preserved up to algebraic degree 3. Those results were obtained by studying the Boolean functions of minimum weight that admit no linear annihilator.

An interesting open question is stated in Question 7.16. That is, can we understand in which cases the minimal degree-d zero-sum sets are also maximal? A more general and indeed remarkable result would be to derive exact formulas for $F(n,d)$ in those cases where we were only able to provide upper and lower bounds. Indeed, solutions to those problems would have interesting implications such as understanding the minimum expansion rate of degree-d sum-invariant matrices and deriving equivalences between degree-d zero-sum sets and Boolean functions with algebraic immunity at least 2.

7.7 Values and Bounds for $F(n,d)$

In the following table we describe known exact values or known bounds of $F(n,d)$ for $n \in \{2,\dots,30\}$ and $d \in \{1,\dots,10\}$. The exact values come from Proposition 7.17, Proposition 7.18 and Proposition 7.24. The lower bounds come from Proposition 7.21 and Proposition 7.19. The upper bounds come from Proposition 7.24. We remark that for $F(2d+5,d)$ the upper bound is obtained by using a slightly different construction. We use the same diagonal construction but fill the free space with 1s. Consider the matrix $\widehat{\mathbb{M}_S}$ given by

$$\widehat{\mathbb{M}_S} = \left[\begin{array}{c|c} \widehat{\mathbb{M}_{S_1}} & \begin{matrix} 1 & \dots & 1 \\ & \vdots & \\ 1 & \dots & 1 \end{matrix} \\ \hline \begin{matrix} 1 & \dots & 1 \\ & \vdots & \\ 1 & \dots & 1 \end{matrix} & \widehat{\mathbb{M}_{S_2}} \end{array}\right],$$

where $S_1 \in \mathrm{ZS}^d_{(d+1)\times F(d+1,d)}, S_2 \in \mathrm{ZS}^d_{(d+4)\times F(d+4,d)}$ and both $\widehat{\mathbb{M}_{S_1}}, \widehat{\mathbb{M}_{S_2}}$ contains a column $(1,\dots,1)$ so that two columns repeat in $\widehat{\mathbb{M}_S}$. Note that the row span of S_1 does not contain a row $(1,\dots,1)$ and thus $\mathrm{rank}(\widehat{\mathbb{M}_S}) = \mathrm{rank}(\widehat{\mathbb{M}_{S_1}}) + \mathrm{rank}(\widehat{\mathbb{M}_{S_2}}) = 2d+5$. The columns of $\widehat{\mathbb{M}_S}$ form a zero-sum set from $\mathrm{ZS}^d_{(2d+5)\times(5\cdot 2^d-2)}$.

n, d	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
2	4									
3	4	8								
4	6	8	16							
5	6	12	16	32						
6	8	12	24	32	64					
7	8	14	24	48	64	128				
8	10	16	28	48	96	128	256			
9	10	18	30	56	96	192	256	512		
10	12	20	32	60	112	192	384	512	1024	
11	12	22	[32,38]	62	120	224	384	768	1024	2048
12	14	24	[32,40]	64	124	240	448	768	1536	2048
13	14	26	[32,44]	[64,78]	126	248	480	896	1536	3072
14	16	28	[34,46]	[64,80]	128	252	496	960	1792	3072
15	16	30	[36,48]	[64,88]	[128,158]	254	504	992	1920	3584
16	18	32	[38,54]	[64,92]	[128,160]	256	508	1008	1984	3840
17	18	34	[40,56]	[64,94]	[128,176]	[256,318]	510	1016	2016	3968
18	20	36	[42,60]	[64,96]	[128,184]	[256,320]	512	1020	2032	4032
19	20	38	[44,62]	[64,110]	[128,188]	[256,352]	[512,638]	1022	2040	4064
20	22	40	[46,64]	[64,112]	[128,190]	[256,368]	[512,640]	1024	2044	4080
21	22	42	[48,70]	[64,120]	[128,192]	[256,376]	[512,704]	[1024,1278]	2046	4088
22	24	44	[50,72]	[64,124]	[128,222]	[256,380]	[512,736]	[1024,1280]	2048	4092
23	24	46	[52,76]	[64,126]	[128,224]	[256,382]	[512,752]	[1024,1408]	[2048,2558]	4094
24	26	48	[54,78]	[64,128]	[128,240]	[256,384]	[512,760]	[1024,1472]	[2048,2560]	4096
25	26	50	[56,80]	[64,142]	[128,248]	[256,446]	[512,764]	[1024,1504]	[2048,2816]	[4096,5118]
26	28	52	[58,86]	[66,144]	[128,252]	[256,448]	[512,766]	[1024,1520]	[2048,2944]	[4096,5120]
27	28	54	[60,88]	[68,152]	[128,254]	[256,480]	[512,768]	[1024,1528]	[2048,3008]	[4096,5632]
28	30	56	[62,92]	[70,156]	[128,256]	[256,496]	[512,894]	[1024,1532]	[2048,3040]	[4096,5888]
29	30	58	[64,94]	[72,158]	[128,286]	[256,504]	[512,896]	[1024,1534]	[2048,3056]	[4096,6016]
30	32	60	[66,96]	[74,160]	[128,288]	[256,508]	[512,960]	[1024,1536]	[2048,3064]	[4096,6080]

TABLE 7.1: This table shows the values of $F(n,d)$ for $n \in \{2,\ldots,30\}$ and $d \in \{1,\ldots,10\}$. In cases where the exact value is not known, $[a,b]$ denotes that $a \le F(n,d) \le b$.

Part III

White-box Cryptography

White-box cryptography studies the security of cryptographic implementations in the white-box model. In this model, an adversary has full access to the implementation, in the form of a program or a circuit. She can, therefore, read or write memory at any time, perform precise fault attacks, analyze the program's control flow. Her goal depends on the security requirement. For white-box implementations of symmetric-key primitives, the most basic security requirement is the secrecy of the key. Such implementations are a long-standing open problem in cryptography. Starting from seminal works of Chow *et al.* [CEJvO02b, CEJvO02a] in 2002, several constructions were proposed in the literature. Unfortunately, all were broken by practical attacks. However, in industry, such implementations are of large interest. Companies use white-box implementations with private designs. This led to a recent direction of applying side-channel attacks to white-box implementations. Bos *et al.* [BHMT16] show that most implementations can be broken by a side-channel attack in a fully automated way.

In this part, I present the work I have done on white-box implementations of symmetric-key primitives. I explore further the space of automated attacks and provide provably secure protection against a new attack. This part is based on the joint work with Alex Biryukov [BU18a].

Chapter 8

Attacks on White-box Implementations

In this chapter, I describe several automated attacks on white-box implementations. It is assumed that the analyzed white-box implementation is protected by some masking scheme, in a rather general sense. The main goal is not to break existing implementations, but to discover properties that a secure obfuscation scheme has to satisfy. This chapter is based on the first part of [BU18a], a joint work with Alex Biryukov.

8.1 Introduction

In the traditional symmetric cryptography, an adversary has access only to the inputs and outputs of a cryptographic primitive. This model is called the *black-box* model. Relaxation of this model is called *grey-box* and in it attacker may also obtain side-channel or fault information from the cryptographic implementation. In the extreme *white-box* model the adversary is given full access to the implementation which contains secret keys. He can use both static and dynamic analysis as well as fault analysis in order to break the cryptosystem, e.g. to extract embedded secret keys. Implementations secure in such model have many applications in industry. However, creating such implementations turns out to be a very challenging if not an impossible task.

In 2002, Chow *et al.* [CEJvO02b, CEJvO02a] proposed first white-box implementations of the AES and DES block ciphers. The main idea is to represent small parts of a block cipher as look-up tables and compose them with randomized invertible mappings to hide the secret key information. Each such look-up table by itself does not give any information about the key. In order to attack such scheme, multiple tables must be considered. Another approach was proposed by Bringer *et al.* [BCD06]. Instead of look-up tables, the cipher is represented as a sequence of functions over $\mathbb{F}_{2^n}$ for some n, with some additional computations as noise. These functions are then composed with random linear mappings to hide the secret key, similarly to the Chow *et al.* approach.

Unfortunately, both approaches fell to practical attacks [BGEC05, DMWP10, LR13]. Consequent attempts to fix them were not successful [Kar10, XL09]. Moreover, Michiels *et al.* [MGH09] generalized the attack by Billet *et al.* [BGEC05] and showed that the approach of Chow *et al.* is not secure for any SPN cipher with MDS matrices. This follows from the efficient cryptanalysis of any SASAS structure [BS01]. Recently several white-box schemes based on the ASASA structure were proposed [BBK14a]. However the strong white-box scheme from that paper was broken [MDFK15, GPT15, BKP17] (which also broadens the white-box attacker's arsenal even further). Another recent approach consists in obfuscating a block cipher implementation using candidates for indistinguishability obfuscation (e.g. [GGH$^+$13]).

Besides academia, there are commercial white-box solutions that are used in real products. The design behind those implementations is kept secret, thus adding *security-by-obscurity* protection. Nevertheless, Bos *et al.* [BHMT16] proposed a framework for attacks on white-box implementations which can automatically break many white-box implementations. The idea is to apply techniques from grey-box analysis (i.e. side-channel attacks) but using more precise data traces obtained from the implementation. The attack is called *differential computation analysis* (DCA). Sasdrich *et al.* [SMG16] pointed out that the weakness against the DCA attack can be explained using the Walsh transform of the encoding functions. Banik *et al.* [BBIJ17] analyzed software countermeasures against the DCA attack and proposed another automated attack called Zero Difference Enumeration attack. More recently, Bock *et al.* [BBMT18] analyzed internal encodings in white-box implementations. Consequently, Rivain and Wang [RW19] provided in-depth analysis and showed that internal encodings can be easily broken in most cases, improved the attack complexities and proposed a new collision attack.

In light of such powerful automated attack the question arises: how to create a whitebox scheme secure against the DCA attack? The most common countermeasure against side-channel attacks is masking, which is a form of secret

Attack	Ref.	Time
Correlation	[BHMT16],Sec. 8.3.1	$\mathcal{O}(n^t k 2^{2t})$
Time-Memory Trade-off	Sec. 8.3.2	$\mathcal{O}(n^{\lceil s/2 \rceil} + n^{\lfloor s/2 \rfloor} k)$
Linear Algebra	[GPRW18],Sec. 8.4.1	$\mathcal{O}(n^\omega + n^2 k)$
Generalized Lin. Alg.	[GPRW18],Sec. 8.4.2	$\mathcal{O}(\binom{n}{\le d}^\omega + \binom{n}{\le d}^2 k)$
LPN-based Gen. Lin. Alg.	Sec. 8.4.4	$T_{LPN}(r, \binom{n}{\le d})$
1-Share Fault Injection	Sec. 8.5.2	$\mathcal{O}(n^2)$
2-Share Fault Injection	Sec. 8.5.1	$\mathcal{O}(n^3)$

Notations: n denotes size of the obfuscated circuit or its part selected for the attack; s is the number of shares in the masking scheme; k is the number of key candidates required to compute a particular intermediate value in the circuit; t denotes the correlation order ($t \le s$); ω is the matrix multiplication exponent (e.g. $\omega = 2.8074$ for Strassen algorithm); d is the algebraic degree of the masking decoder (see Section 8.4.2); $\binom{n}{\le d} = \sum_{i=0}^{d} \binom{n}{i}$ is the number of monomials of n bit variables of degree at most d; r is the noise ratio in the system of equations, $T_{LPN}(r, m), D_{LPN}(r, m)$ are time and data complexities of solving an LPN instance with noise ratio r and m variables.

Table 8.1: Attacks on masked white-box implementations.

sharing. It is therefore natural to apply masking to protect white-box implementations. We define masking to be any obfuscation method that encodes each original bit by a relatively small amount of bits. Such masking-based obfuscation may be more practical in contrast to cryptographic obfuscation built from current indistinguishability obfuscation candidates [GGH+13, CMR17].

8.1.1 Our Contribution

This chapter studies the possibility of using masking schemes in the white-box setting. We restrict the analysis to implementations in the form of Boolean circuits.

We develop a more generic DCA framework and describe multiple generic attacks against masked implementations. The attacks show that the classic Boolean masking (XOR-sharing) is inherently weak. Previous and new attacks are summarized in Table 8.1. We remark that conditions for different attacks vary significantly and the attacks should not be compared solely by time complexity. For example, the fault-based attacks are quite powerful, but it is relatively easy to protect an implementation from these attacks. From the attacks we conclude that more general nonlinear encodings are needed and we deduce constraints that a secure implementation must satisfy. We believe that these results provide new insights on the design of white-box implementations. Note that a basic variant of the (generalized) linear algebra attack was independently proposed by Goubin *et al.* [GPRW18].

A code implementing the described attacks and protections from Chapter 9 is publicly available at [BU18b]:

`https://github.com/cryptolu/whitebox`

8.1.2 Outline

The general attack setting and attacks are described in Section 8.2. Combinatorial and algebraic attacks in the DCA setting are described in Section 8.3 and Section 8.4 respectively. In Section 8.5 I suggest fault-based attacks. Finally, I conclude in Section 8.6.

8.2 Differential Computational Analysis

I describe the general setting for our attacks. We consider *a keyed symmetric primitive*, e.g. a block cipher. A *white-box designer* takes a naive implementation with a hardcoded secret key and obfuscates it producing *a white-box implementation*. An adversary receives the white-box implementation and her goal is to recover the secret key or a part of it. We restrict our analysis to implementations in the form of *Boolean circuits*.

Definition 8.1. *A* Boolean circuit *C is a directed acyclic graph where each node with the indegree $k > 0$ has an associated k-ary symmetric Boolean function g_v. Nodes with the indegree equal to zero are called* inputs *of C and nodes with the outdegree equal to zero are called* outputs *of C.*

Let $x = (x_1, \ldots, x_N)$ (resp. $y = (y_1, \ldots, y_M)$) be a vector of input (resp. output) nodes in a fixed order. For each node v in C we say that it computes a Boolean function $f_v : \mathbb{F}_2^N \to \mathbb{F}_2$ defined as follows:

- *for all $1 \le i \le N$ set $f_{x_i}(z) = z_i$,*
- *for all non-input nodes v in C set $f_v(z) = g_v(f_{c_1}(z), \ldots, f_{c_k}(z))$, where $c_1, \ldots, c_k$ are nodes having an outgoing edge to v.*

The set of f_v for all nodes v in C is denoted $\mathcal{F}(C)$ and the set of f_{x_i} for all input nodes x_i is denoted $\mathcal{X}(C)$. By an abuse of notation we also define the function $C : \mathbb{F}_2^N \to \mathbb{F}_2^M$ as $C = (f_{y_1}, \ldots, f_{y_m})$.

Masking Schemes

We assume that the white-box designer uses masking in some form, but we do not restrict him from using other obfuscation techniques. The only requirement is that there exists a relatively small set of nodes in the obfuscated circuit (called *shares*) such that during a legitimate computation the values computed in these nodes sum to *a predictable value*. We at least expect this to happen with overwhelming probability. In a more general case, we allow arbitrary functions to be used to compute the predictable value from the shares instead of plain XOR. We call these functions *decoders*. The classic Boolean masking technique is based on the XOR decoder. The number of shares is denoted by s.

I give a broad definition of a masking scheme that will be used also in Chapter 9.

Definition 8.2 (Masking Scheme)**.** *An T-bit masking scheme is defined by an encoding function $Encode : \mathbb{F}_2 \times \mathbb{F}_2^R \to \mathbb{F}_2^T$, a decoding function $Decode : \mathbb{F}_2^T \to \mathbb{F}_2$ and a set of triplets $\{(\diamond, Eval_\diamond, C_\diamond), \ldots\}$ where each triplet consists of:*

1. *a Boolean operator $\diamond : \mathbb{F}_2 \times \mathbb{F}_2 \to \mathbb{F}_2$,*
2. *a circuit $Eval_\diamond : \mathbb{F}_2^T \times \mathbb{F}_2^T \times \mathbb{F}_2^{R'} \to \mathbb{F}_2^T$.*

For any $r \in \mathbb{F}_2^R$ and any $x \in \mathbb{F}_2$ it must hold that $Decode(Encode(x, r)) = x$. Moreover, the following equation must be satisfied for all operators $\diamond$ and all values $r' \in \mathbb{F}_2^{R'}, x_1 \in \mathbb{F}_2^T, x_2 \in \mathbb{F}_2^T$:

$$Decode(Eval_\diamond(x_1, x_2, r')) = Decode(x_1) \diamond Decode(x_2).$$

The degree of the masking scheme is the algebraic degree of the Decode function. The masking scheme is called nonlinear if its degree is greater than 1.

Note that $Eval_\diamond$ takes three arguments in the definition. The first two are shares of the secret values and the third one is optional randomness that must not change the secret values.

Predictable Values

A predictable value typically is a value computed in the beginning or in the end of the reference algorithm such that it depends only on a few key bits and on the plaintexts/ciphertexts. In such case the adversary makes a guess for the key bits and computes the corresponding candidate for the predictable value. The total number of candidates is denoted by k.

The obfuscation method may require random bits e.g. for splitting the secret value into random shares. Even if the circuit may have input nodes for random bits in order to achieve non-deterministic encryption, the adversary can easily manipulate them. Therefore, the obfuscation method has to rely on pseudorandomness computed solely from the input. Locating and manipulating the pseudorandomness generation is a possible attack direction. However, as we aim to study the applicability of masking schemes, we assume that the adversary can not directly locate the pseudorandomness computations and remove the corresponding nodes. Moreover, the adversary can not predict the generated pseudorandom values with high probability, i.e. such values are not predictable values.

Window Coverage

In a typical case shares of a predictable value will be relatively close in the circuit (for example, at the same circuit level or at a short distance in the circuit graph). This fact can be exploited to improve efficiency of the attacks. The adversary covers the circuit by sets of closely located nodes. Any such set is called *a window* (as in power analysis attack terminology e.g. from [BB17]).

The described attacks can be applied to each window instead of the full circuit. By varying the window size the attacks may become more efficient. Here we do not investigate methods of choosing windows to cover a given circuit. One possible approach is to assign each level or a sequence of adjacent levels in the circuit to a window. Choosing the full circuit as a single window is also allowed. In our attacks we assume that a coverage is already chosen. For simplicity, we describe how each attack is applied to a single window. In case when multiple windows are chosen, the attack has to be repeated for each window. The window size is denoted by n. It is equal to the circuit size in the case of the single window coverage.

General DCA Attack

I would like to note that the term “differential computation analysis” (DCA) is very general. In [BHMT16] the authors introduced it mainly for the correlation-based attack. In fact our new attacks fit the term well and provide new tools for the “analysis” stage of the attack. The first stage remains the same except that we adapt the terminology for the case of Boolean circuits instead of recording the memory access traces. Our view of the procedure of the DCA attack on a white-box implementation C is given in Algorithm 8.1

Algorithm 8.1 General procedure of DCA attacks on a Boolean circuit $C\colon \mathbb{F}_2^N \to \mathbb{F}_2^M$

```
 1: generate a random tuple of plaintexts $P = (p_1, p_2, \ldots), p_i \in \mathbb{F}_2^N$
 2: for all $p_i \in P$ do
 3:     compute the circuit $C$ on input $p_i$: $c_i \leftarrow C(p_i) \in \mathbb{F}_2^M$
 4:     for all $j \in [1 \ldots |C|]$ do
 5:         $v_{j,i} \leftarrow$ computed value in the node indexed $j$
 6:     for all $j \in [1 \ldots k]$ do
 7:         $\tilde{v}_{j,i} \leftarrow$ predictable value indexed $j$
                      computed from plaintext $p_i$ and/or ciphertext $c_i$
 8: generate the list of all computed vectors:
    $V \leftarrow (v_1, \ldots, v_{|C|})$, where $v_j = (v_{j,1}, \ldots, v_{j,|P|}) \in \mathbb{F}_2^{|P|}$
 9: generate the list of all predictable vectors:
    $\tilde{V} \leftarrow (\tilde{v}_1, \ldots, \tilde{v}_k)$, where $\tilde{v}_j = (\tilde{v}_{j,1}, \ldots, \tilde{v}_{j,|P|}) \in \mathbb{F}_2^{|P|}$
10: choose a coverage $\mathcal{P}$ of $V$ by windows of size $n$
11: for all $W \in \mathcal{P}$ do
12:     perform analysis on the window $W \subseteq V$
        using the set of predictable vectors $\tilde{V}$
```

We remark that the correlation-based DCA attack from [BHMT16] can be implemented on-the-fly, without computing the full vectors v_j. In contrast, most of our attacks require full vectors. Though, various optimizations are possible.

In the following two sections I describe two classes of DCA attacks: combinatorial and algebraic. They both follow the procedure described above and differ only in the analysis part (Step 12). Afterwards, I describe two fault-injection attacks which allow to find locations of shares efficiently.

8.3 Combinatorial DCA attacks

The most straightforward way to attack a masked implementation is to guess location of shares inside the current window. For each guess we need to check if the shares match the predictable value. In the basic case of classic Boolean masking where the decoder function is simply XOR of the shares the check is trivial. If an unknown general decoder function has to be considered, the attack becomes more difficult. One particularly interesting case is a basic XOR decoder with added noise (i.e. low-weight pseudorandom functions of the input). The main attack method in such cases is correlation.

8.3.1 Correlation attack

The correlation DCA attack from [BHMT16] is based on correlation between single bits. However, in the case of classic Boolean masking with strong pseudorandom masks all s shares are required to perform a successful correlation attack. In the case of a nonlinear decoder less shares may be enough: even a single share correlation can break many schemes as demonstrated in [BHMT16,RW19]. Existing higher-order power analysis attacks are directly applicable to memory or value traces of white-box implementations. However, the values leaked in the white-box setting are exact in contrast to side-channel setting and the attack may be described in a simpler way. I reformulate the higher-order correlation attack in our DCA framework. Different correlation metrics of binary vectors can be used, see e.g. [W+08]. In this chapter I defined the correlation as the sample Pearson correlation coefficient.

Definition 8.3. *The* correlation *of two n-bit vectors v_1 and v_2 is defined as*

$$\mathbf{cor}(v_1, v_2) = \frac{n_{11}n_{00} - n_{01}n_{10}}{\sqrt{(n_{00}+n_{01})(n_{00}+n_{10})(n_{11}+n_{01})(n_{11}+n_{10})}},$$

where n_{ij} denotes the number of positions where v_1 equals to i and v_2 equals to j. If the denominator is zero then the correlation is set to zero. **cor** *is the sample Pearson correlation coefficient of two binary variables, also known as the Phi coefficient.*

Assume that locations of t shares are guessed and t vectors $v_j \in \mathbb{F}_2^{|P|}$ are selected. For simplicity, I denote them by $(v_1, \ldots, v_t) \subseteq V$. For each vector $m \in \mathbb{F}_2^t$ we compute $u_m \in \mathbb{F}_2^{|P|}$ where

$$u_{m,i} = (v_{1,i} = m_1) \wedge \ldots \wedge (v_{t,i} = m_t).$$

In other words, $u_{m,i}$ is equal to 1 if and only if during encryption of the i-th plaintext the shares took the value described by m . For each predictable vector $\tilde{v}$ we compute the correlation $\mathbf{cor}(u_m, \tilde{v})$. If its absolute value is above a predefined threshold, we conclude that the attack succeeded and possibly recover part of the key from the predictable value $\tilde{v}$. Furthermore, the entire vector of correlations $(\mathbf{cor}(u_{(0,\ldots,0)}, \tilde{v}), \mathbf{cor}(u_{(0,\ldots,1)}, \tilde{v}), \ldots)$ may be used in analysis, e.g. the average or the maximum value of its absolute entries.

We assume that the predictable value is not highly unbalanced. Then for the attack to succeed we need the correlated shares to hit at least one combination m a constant number of times (that is obtain $wt(u_m) \geq const$). Therefore the data complexity is $|P| = \mathcal{O}(2^t)$. However, with larger number of shares the noise increases and more data may be required. We estimate the time complexity of the attack as $\mathcal{O}(n^t k 2^t |P|) = \mathcal{O}(n^t k 2^{2t})$. Here n^t corresponds to guessing location of shares inside each window (we assume $t \ll n$); k corresponds to iterating over all predictable values; 2^{2t} corresponds to iterating over all t-bit vectors m and computing the correlations.

The main advantage of this attack is its generality. It works against general decoder functions even with additional observable noise. In fact, the attack may work even if we correlate less shares than the actual encoding requires. Indeed, the attack from [BHMT16] relied on single-bit correlations and still was successfully applied to break multiple whitebox designs. The generality of the attack makes it inefficient for some special cases, in particular for the classic Boolean masking. We investigate this special case and describe more efficient attacks.

8.3.2 Time-Memory Trade-off

Consider now the case of XOR decoder and absence of observable noise. That is, the decoder function must map the shares to the correct predictable value for all recorded plaintexts. In such case we can use extra memory to improve the attack. Consider two simple cases by the number of shares:

1. Assume that the decoder uses a single share (i.e. unprotected implementation). We precompute all the predictable vectors and put them in a table. Then we simply sweep through the circuit nodes and for each vector v_i check if it is in the table. For the right predictable vector $\tilde{v}$ we will have a match.

2. Assume that the decoder uses two shares (i.e. first-order protected implementation). We are looking for indices i, j such that $v_i \oplus v_j = \tilde{v}$ for some predictable vector $\tilde{v}$. Equivalently, $v_i = \tilde{v} \oplus v_j$. We sweep through the window's nodes and put all the node vectors in a table. Then we sweep again and for each vector v_j in the window and for each predictable vector $\tilde{v}$ we check if $v_j \oplus \tilde{v}$ is in the table. For the right $\tilde{v}$ we will have a match and it will reveal both shares.

This method easily generalizes for arbitrary number of shares. We put the larger half of shares on the left side of the equation and put the corresponding tuples of vectors in the table. Then we compute the tuples of vectors for the smaller half of shares and look-up them in the table. We remark that this attack's complexity still has combinatorial explosion. However the time-memory trade-off essentially allows to half the exponent in the complexity.

The attack effectively checks $n^s k$ sums of vectors to be equal to zero. To avoid false positives, the data complexity should be set to $O(s \log_2 n + \log_2 k)$. We consider this data complexity negligible, especially because for large number of shares the attack quickly becomes infeasible. For simplicity, we assume the data complexity is $O(1)$ and then the time complexity of the attack is $\mathcal{O}(n^{\lceil s/2 \rceil} + n^{\lfloor s/2 \rfloor}k)$.

The described attack is very efficient for unprotected or first-order masked implementations. For small windows it can also be practical for higher-order protections. In the following section I describe a more powerful attack whose complexity is independent of the number of shares.

8.4 Algebraic DCA attacks

8.4.1 Linear Algebra Attack

For the classic Boolean masking the problem of finding shares consists in finding a subset of the window's vectors which sums to one of predictable vectors. Clearly, this is a basic linear algebra problem. Let A be the matrix that has as columns vectors from the current window. For each predictable vector $\tilde{v}$ we solve the equation $A \times x = \tilde{v}$. A solution vector x reveals shares locations. To avoid false-positive solutions the number $|P|$ of encryptions should be increased proportionally to the window size. For the same matrix A we need to check all predictable vectors. Instead of solving the entire system each time, we precompute the LU decomposition of the matrix and then use it for checking each predictable vector much faster. We estimate the data complexity $|P| = \mathcal{O}(n)$ and the time complexity $\mathcal{O}(n^{\omega} + n^2k)$, where ω is the matrix multiplication exponent. This attack was independently proposed by the CryptoExperts team in [GPRW18] and among other techniques was successfully applied [GPRW17] during the WhibOx 2017 competition [ECR17] in order to break the winning challenge "Adoring Poitras".

We conclude that classic Boolean masking is insecure regardless of the number of shares. The attack complexity is polynomial in the circuit size. Even though it may not be highly practical to apply the attack to entire circuits containing millions of nodes, good window coverage makes the attack much more efficient. The attack becomes especially dangerous if a window containing all shares may be located by analyzing the circuit. Indeed, this is how team CryptoExperts attacked the main circuit of the winning challenge of the WhibOx competition. They obtained a minimized circuit containing around 300000 nodes; they draw the data dependency graph (DDG) of the top 5% nodes and visually located several groups of 50 nodes and successfully mounted the described linear attack on each of the groups.

8.4.2 Generalization through Linearization

The described linear attack suggests that a nonlinear masking scheme has to be used. We show that the attack can be generalized to nonlinear masking schemes as well. Of course, the complexity grows faster. Still, the attack can be used to estimate the security of such implementations.

The generalization is based on the linearization technique. The idea is to compute products of vectors (with bitwise AND) and include them as possible shares of the predictable vector. Each such product corresponds to a possible monomial in the algebraic normal form of the decoder function. The correct linear combination of monomials equals to the decoder function. The corresponding linear combination of products of vectors equals to the correct predictable vector.

The set of products may be filtered. If a bound on the degree of the decoder function is known, products with higher degrees are not included. For example, for a quadratic decoder function only the vectors v_i and all pairwise products $v_i v_j$ should be included.

The data complexity is dependent on the number of possible monomials in the decoder function. For simplicity, we consider an upper bound d on the algebraic degree. Then the number of possible monomials is equal to

$$\binom{n}{\leq d} := \sum_{i=0}^{d} \binom{n}{i}.$$

This generalized attack has the data complexity $\mathcal{O}(\binom{n}{\leq d})$ and the time complexity $\mathcal{O}(\binom{n}{\leq d}^{\omega} + \binom{n}{\leq d}^{2} k)$.

The following definition is useful in formalizing the attack. It will be particularly useful in Chapter 9, where countermeasures against this attack are analyzed.

Definition 8.4 (*d*-th order closure)**.** *Let $V \subseteq \mathbb{F}_2^n, V = \{v_1, v_2, \ldots\}$. Define the* d-th order closure *of V (denoted $V^{(d)}$) to be the vector space spanned by all component-wise products of at most d vectors from V.*

$$V^{(d)} = \operatorname{span}\{\mathbf{1}\} \cup \{(v_{i_1} \wedge v_{i_2} \wedge \ldots \wedge v_{i_d} \mid 1 \leq i_1 \leq i_2 \leq \ldots \leq i_d \leq |V|)\}.$$

Let $\mathcal{V}$ be a set of Boolean functions with the same domain $\mathbb{F}_2^N$. The d-th order closure of $\mathcal{V}$ (denoted $\mathcal{V}^{(d)}$) is defined completely analogously to $V^{(d)}$.

Example 3.
$\mathcal{V}^{(1)}$ is spanned by $\{\mathbf{1}\} \cup \{g_i \mid g_i \in \mathcal{V}\}$,
$\mathcal{V}^{(2)}$ is spanned by $\{\mathbf{1}\} \cup \{g_i g_j \mid g_i, g_j \in \mathcal{V}\}$ (includes $\mathcal{V}^{(1)}$ as $i = j$ is allowed).

The (first-order) linear algebra attack can then be described as searching for a predictable vector $\tilde{v}$ in the vector space $V^{(1)}$. The generalized linear algebra attack of order d then searches in the vector space $V^{(d)}$.

It is worth remarking that it is enough to consider only nonlinear (e.g. AND, OR) and input nodes inside the current window. All other nodes are affine combinations of these and are redundant. This fact is formalized in the following proposition.

Proposition 8.5. *Let C be a Boolean circuit. Let $\mathcal{N}(C)$ be the set of all functions computed in the circuit's nonlinear nodes (i.e. any node except XOR, NOT, NXOR) together with functions returning input bits. Then for any integer $d \geq 1$ the sets $\mathcal{F}^{(d)}(C)$ and $\mathcal{N}^{(d)}(C)$ are the equal.*

Proof. Note that for any set $\mathcal{V}$ we have $\mathcal{V}^{(d)} = (\mathcal{V}^{(1)})^{(d)}$. Therefore, we only need to prove that $\mathcal{F}^{(1)}(C) = \mathcal{N}^{(1)}(C)$. It is sufficient to show that any function from $\mathcal{F}$ belongs to $\mathcal{N}^{(1)}(C)$. This can be easily proved by induction on circuit levels. □

Remark 16. Note that linear relations may still hold between functions computed in the nonlinear gates. For example, the XOR gate may be implemented by several NAND gates. All such relations can be exploited to reduce the search space by simply reducing the set V to a basis of the space that it spans. It is easy to show that the d-th order closure of such basis is equal to the d-th order closure of V itself.

I describe an interesting scenario where this generalized attack is highly relevant. Assume that a white-box designer first applies classic Boolean masking to the reference circuit. Afterwards, each intermediate bit is encoded by e.g. 8 bits using a random nonlinear encoding. The masked circuit then is transformed into a network of lookup tables which perform operations on the encoded bits without explicitly decoding them. The motivation for such scheme is that there will be no correlation between a single 8-bit encoding and any predictable vector because of the linear masking applied under the hood. For the generalized linear attack the degree bound is equal to 8 and normally, the time complexity would be impractical. However, in this case the lookup tables reveal the locations of encodings, i.e. the 8-bit groups. Therefore, we include only 2^8 products from each group and no products across the groups. The attack works because the predictable value is a linear combination of XOR-shares which in turn are linear combinations of products (monomials) from each group. I remark that the system has a simpler expression in the point basis, i.e. when we consider functions of the form $x \mapsto (x = c)$ for all $c \in \mathbb{F}_2^8$ instead of monomial maps.

8.4.3 Value-restriction Analysis

The described algebraic attack can be modified to cover a broader range of masking schemes. Consider a low-degree combination of vectors from the current window and assume that the function it computes can be expressed as $s \wedge r$, where s is the correct predictable value and r is some uniform pseudorandom (unrelated) value. The basic algebraic attack will not succeed because $s \wedge r$ is not always equal to the predictable value s. However, it is possible to extend the attack to exploit the leakage of $s \wedge r$. The adversary chooses a set of inputs for which the predictable value s is equal to 0 and adds a single random input for which the predictable value is equal to 1 (the adversary may need to guess a part of the key to compute the predictable value). Then with probability $1/2$ he is expected to find a vector with all bits equal to 0 except the last bit equal to 1. In case the predictable value is wrong, the chance of finding such vector is exponentially small in the size of the plaintext set. The same approach works for more complex leaked functions. In particular, the leaked function may depend on multiple predictable values, e.g. on all output bits of an S-Box. The only requirement is that the leaked function must be constant for at least one assignment of the predictable values (except of course the case when the leaked function is constant on all inputs). However, the adversary must be able to find the correct assignment of predictable values. As a conclusion, this attack variant reveals a stronger constraint that a masking scheme must satisfy in order to be secure.

8.4.4 Algebraic Attack in the Presence of Noise

In spirit of the value-restriction analysis, we continue to explore classes of exploitable leaking functions. Assume that a low-degree combination of vectors from the current window corresponds to a function $s \oplus e$, where s is the correct predictable vector and e is a function with a low Hamming weight. The function e may be unpredictable and we consider it as noise. The problem of solving a noisy system of linear equations is well known as Learning Parity with Noise (LPN). It is equivalent to the problem of decoding random linear codes. The best known algorithms have exponential running time. We refer to a recent result by Both and May [BM18] where the authors propose an algorithm with approximated complexity $2^{1.3nr}$, where n is the number of unknown variables and r is the noise ratio. Several algorithms with low memory consumption were recently proposed by Esser *et al.* [EKM17]. The best algorithm for the problem depends on the exact instance parameters. The number of variables in our case corresponds to the number of monomials considered, i.e. the window size n in the linear attack and $\binom{n}{\le d}$ in the generalized attack. For example, if a linear combination of vectors from a 100-node window leaks s with noise ratio $1/4$ then the LPN-based attack will take time $2^{32.5}$ using the algorithm from [BM18].

8.5 Fault Attacks

Previous attacks assumed that the adversary knows the obfuscated circuit and can analyze it in an arbitrary way. Still, the attacks described in previous sections were passive: they relied on analysis of computed intermediate values during encryptions of random plaintexts. In this section I describe active attacks - fault injections - that can also be used to attack masked white-box implementations. We assume that the classic Boolean masking is used. We also allow any form of *integrity protection* which protects the values but does not protect the shares. That is, the protection may detect a fault that influences ciphertext, but does not detect a fault that modifies masks in a way that does not alter the masked value.

8.5.1 Two-Share Fault Injection

The main goal of a fault attack against masking is to locate shares of the masked values. Observe that flipping two XOR-shares of a value does not change the value. This property can be used to locate positions of possible shares. The attack procedure is given in Algorithm 8.2.

Remark 17. As shares of the same value should be placed closely in the circuit, a window coverage can be used to improve efficiency of this attack too. The idea is to choose two shares only inside each window and not across the windows.

Algorithm 8.2 Two-share fault attack on a circuit $C\colon \mathbb{F}_2^N \to \mathbb{F}_2^M$

```
1: $p \xleftarrow{\$} \mathbb{F}_2^N$
2: $c \leftarrow C(p) \in \mathbb{F}_2^M$
3: for all $i, j \in [1 \dots |C|], i < j$ do
4:     $c_{i,j} \leftarrow C(p) \in \mathbb{F}_2^M$, with the values in the nodes indexed $i$ and $j$
       flipped during encryption
5:     if $c = c_{i,j}$ then
6:         repeat the check several times (for random plaintexts)
7:         return possible shares $i, j$
```

Remark 18. There may be a lot of false positives. For example, if values in the nodes indexed i and j are XORed and not used anymore, the attack will always return these two nodes. In general, for any two nodes returned by the algorithm, the two values can be compressed into one. Indeed, since flipping both values does not change the result, then only the XOR of the two can be relevant. Effectively this means that these two nodes can be excluded from analysis, and their XOR included instead. After finishing the process, all shares will be compressed into one and can be attacked with simple DCA attacks.

The described attack allows to locate all shares of each value, independently of the sharing degree. The attack performs $\mathcal{O}(n^2)$ encryptions and has time complexity $\mathcal{O}(|C|n^2)$.

8.5.2 One-Share Fault Injection

Recall that we allow an integrity protection on the values but not on the shares. One possible way an integrity protection may be implemented is to perform the computations twice and spread the difference between the two results across the output in some deterministic way. In such way small errors are amplified into random ciphertext differences. In case of such protection or absence of any protection the efficiency of the fault attack can be improved.

The main idea for improvements comes from the following observation: if we flip a single share of some value, the masked value will be flipped as well. This results in a fault injected in the unmasked circuit. The assumption is that the circuit output does not depend on which share was faulted. This observation allows to split the two-share fault attack and perform fault injection only for each node instead of each pair of nodes, at the cost of additional storage. The procedure is given in Algorithm 8.3

The attack performs $\mathcal{O}(n)$ encryptions, which requires $\mathcal{O}(|C|n)$ time. It provides substantial improvement over previous attack, though it requires stronger assumption about the implementation. The most relevant counter-example is when the integrity protection does not amplify the error but simply returns a fixed output for any detected error. In a sense, such protection does not reveal in the output any information about the fault. On the other hand, it may be easier to locate the error checking part in the circuit and remove the protection.

Algorithm 8.3 One-share fault attack on a circuit $C\colon \mathbb{F}_2^N \to \mathbb{F}_2^M$

```
1: $p \xleftarrow{\$} \mathbb{F}_2^N$
2: $c \leftarrow C(p) \in \mathbb{F}_2^M$
3: initialize a hash map $T\colon \mathbb{F}_2^M \to \{1, \ldots, |C|\}^*$
4: for all $i \in [1 \ldots |C|]$ do
5:     $c_i \leftarrow C(p) \in \mathbb{F}_2^M$, with the value in the node indexed $i$
       flipped during encryption
6:     append $i$ to $T(c_i)$
7:     if $T(c_i)$ contains more than one value then
8:         $(i_1, i_2, \ldots) \leftarrow T(c_i)$
9:         repeat the check several times (for random plaintexts)
10:        return possible shares $(i_1, i_2, \ldots)$
```

The attacks can be adapted for nonlinear masking as well. In such case the injected fault may leave the masked value unflipped. When a zero difference is observed in the output, the fault injection should be repeated for other plaintexts. As plaintext is the only source of pseudorandomness, changing the plaintext should result in different values of shares. Flipping a share would result in flipping the masked value with nonzero probability. The exact probability depends on the decoder function.

Similarly to the two-share fault attack, there may be many false-positives. That is, the algorithm may return nodes that do not correspond to shares of the same value. Still, it is likely that there is a strong relation between the nodes. The algorithm thus provides some information about the implementation, which can be further used for detailed analysis.

Remark 19. The two described attacks perform faults on *nodes* of the circuit. In some cases, a node value may be used as a share of multiple different values, for example, if the same pseudorandom value is used to mask several values. A more general variant of attacks would inject faults on *wires*. However, multiple wires may need to be faulted in order to succeed and the attack may become complicated and inefficient.

8.6 Conclusions

In this chapter we studied the possibility of using masking techniques for white-box implementations. We presented several attacks applicable in different scenarios. As a result, we obtained several requirements for a masking scheme useful for white-box implementations. In Chapter 9, I will describe an analysis the requirements and a partial solution against DCA-style attacks - a nonlinear masking scheme with provable properties that guarantee security against the linear algebra attack.

We applied the attacks to several challenges from the WhibOx 2017 competition [ECR17]. However, we did not perform an extensive study of the applicability of the attacks to public white-box implementations. One problem is that most implementations can not be converted to a circuit in a simple way. This is an interesting direction for future work.

Another interesting open problem is to develop countermeasures for fault attacks in the white-box setting. Indeed, these attacks are quite powerful and known gray-box protection may be not strong enough. From the attacks we can see that the *shares* must be protected as well, meaning that an integrity protection should be applied on top of a masking scheme.

Chapter 9

Provably Secure Countermeasures

In this chapter, I describe analysis of the attacks from Chapter 8 and a general method for protecting white-box implementations. The protection splits into two independent components: *value hiding* and *structure hiding*. Value hiding must provide protection against passive DCA-style attacks that rely on analysis of computation traces. Structure hiding must provide protection against circuit analysis attacks. We focus on the development of the value hiding component. As a result, I show a nonlinear masking scheme provably secure against the linear algebra attack, described in Chapter 8. This chapter is based on the second part of [BU18a], a joint work with Alex Biryukov.

9.1 Introduction

9.1.1 Our Contribution

Components of Protection. We propose in Section 9.2 a general method for designing a secure white-box implementation. The idea is to split the protection into two independent components: *value hiding* and *structure hiding*. The value hiding component must provide protection against passive DCA-style attacks - attacks that rely solely on analysis of computed values. In particular, it must provide security against the correlation attack and the algebraic attack. We suggest that security against these two attacks can be achieved by applying a classic linear masking scheme on top of a nonlinear masking scheme protecting against the algebraic attack. The structure hiding component must secure the implementation against circuit analysis attacks. The component must protect against circuit minimization, pattern recognition, pseudorandomness removal, fault injections, etc. Possibly this component may be splitted into more sub-components (e.g. an integrity protection). Development of a structure hiding protection is left as a future work.

Provably Secure Construction. Classic t-th order masking schemes protect against adversaries that are allowed to probe t intermediate values computed by the implementation. The complexity of the attack grows fast when t increases. In the new algebraic attack the adversary is allowed to probe all intermediate values but she can combine them only with a function of low algebraic degree d. Similarly, the attack complexity grows fast when d increases and also when the circuit size increases. We develop a framework for securing an implementation against the algebraic attack. It includes a formal security model and a proof of the composability of first-order secure circuits. Finally, I describe our first-order secure masking scheme implementing XOR and AND operations. As a result, our framework provides provable security against the first-order algebraic attack. I show concrete security bounds for our construction. Finally, we implement the AES-128 block cipher protected using our new masking scheme.

A code implementing the attacks from Chapter 8, verification of the algebraic masking schemes and the masked AES-128 implementation is publicly available at [BU18b]:

https://github.com/cryptolu/whitebox

9.1.2 Outline

I describe our general method for securing a white-box design in Section 9.2. In Section 9.3 a framework is developed for countermeasures against the algebraic attack. In Section 9.4 I describe a simple quadratic masking scheme following the proposed framework. Finally, I conclude and suggest future work in Section 9.5.

9.2 Protection Components

The attacks described in Chapter 8 significantly narrow down the space of masking schemes useful for white-box obfuscation. We deduce the following main constraints:

1. The number of shares should be high enough to avoid combinatorial attacks. Moreover, the minimum number of shares that correlate with the reference circuit values should be high as well.

2. There should be no low-degree decoders in order to prevent the algebraic attack.

3. The circuit must not admit analysis that allows to locate shares of the same values.

4. The integrity of pseudorandom shares must be protected.

The aim of this chapter is to analyze the possibility of using masking schemes with *relatively small* number of shares for white-box cryptography. The complexity of combinatorial attacks splits into two parts: locating the shares and correlating them. If the number of shares is very high then the correlation part becomes infeasible. Possibly, in such case it is not even necessary to hide the location of shares. The downside is that designing such masking schemes is quite challenging and this direction leads into rather theoretical constructions like indistinguishability obfuscation [GGH+13] from fully homomorphic encryption and other cryptographic primitives. We aim to find more practical obfuscation techniques. Therefore, we have to study obfuscation methods relying on hardness of locating shares inside the obfuscated circuit. Such obfuscation is a challenging problem. In the light of described attacks, we suggest a modular approach to solve this problem. We split the problem into two components:

1. *(Value Hiding)* Protection against generic passive attacks that do not rely on the analysis of the circuit.

2. *(Structure Hiding)* Protection against circuit analysis and fault injections.

9.2.1 Value Hiding

The first component basically requires designing a proper masking scheme. As we have shown, the requirements are much stronger than for the usual masking in the side-channel setting (e.g. the provably secure masking by Ishai *et al.* [ISW03]). To the best of our knowledge, this direction was not studied in the literature. However, there is a related notion: fully homomorphic encryption (FHE). Indeed, it can be seen as an extreme class of masking schemes. FHE encryption is a process of creating shares of a secret value and the FHE's evaluation functions allow to perform arbitrary computations on the ciphertexts (shares) without leaking the secret value. In fact, any secure FHE scheme would solve the "Value Hiding" problem (even though the adversary may learn the key from the decryption phase, the locations of intermediate shares should remain unknown due to structure-hiding protection and the scheme may remain secure). However, this direction leads to very inefficient schemes: typical FHE schemes have very large ciphertexts and complex circuits. This contradicts our goal to investigate schemes with reasonable number of shares.

We suggest to further split the first component into two parts. The first part is protection against algebraic attacks. It is a nonlinear masking scheme without low-degree decoders. However, we allow the scheme to be imperfect: the computed values may correlate with the secret values. Though one has to be careful and avoid very strong correlation, otherwise the LPN-based variant of the algebraic attack may be applicable. The second part is protection against correlation attacks. It can be implemented using a provably secure linear masking scheme on top of the nonlinear masking from the first part. The two parts may be composed in the following way: the algebraically secure nonlinear masking scheme is applied to the reference circuit and afterwards the linear masking scheme is applied to the transformed circuit. We investigate possibilities for the algebraically secure nonlinear masking in the next section.

9.2.2 Structure Hiding

The second component resembles what is usually understood by *software obfuscation*. Indeed, the usual software obfuscation aims to obfuscate the control flow graph and hide important operations. Often such obfuscation includes integrity protections to prevent patching. The computed values are not hidden but merely blended among redundant values computed by dummy instructions. For circuits the problem is less obscure and ad hoc. In particular, an integrity protection scheme for circuits was proposed by Ishai *et al.* in [IPSW06]. Though, formalizing the "protection against analysis" is not easy. Applying structure hiding protection on top of value hiding protection should secure the implementation from attacks described in Chapter 8. We do not investigate structure hiding further in this work.

We note that it is not possible to formally separate value hiding from structure hiding. If we give the adversary computed vectors of values even in shuffled order, she can reconstruct the circuit in reasonable time and then analyze it. One possible direction is to mix the value vectors linearly by a random linear mapping before giving to the adversary. It may be a difficult problem for the adversary to recover the circuit or its parts from such input. However, such model makes the correlation DCA attack almost inapplicable, since a lot of values are unnaturally mixed up and the correlations are not predictable, even though it is perfectly possible that the original unmixed values have strong correlations with secret variables.

9.3 Framework for Algebraically Security

The algebraic attack is very powerful and the classic XOR-sharing masking schemes can not withstand it. Therefore, it is important to develop new masking schemes which are secure against the algebraic attack. In this section I describe a formalization of security against the algebraic attack and a provably first-order secure construction.

I start by discussing the attack model in Section 9.3.1. A formal game-based security definition is given in Section 9.3.2. Ways of proving security in the new model are developed in Section 9.3.3. Next, the composability is studied in

Section 9.3.4. An algorithm for checking security of gadgets is proposed in Section 9.3.5.

9.3.1 Security Model

Consider a subproblem from the whitebox design problem. Recall that during the algebraic attack, the adversary tries to find a function f of low degree d such that when applied to values computed in the nodes of the obfuscated circuit it would produce some predictable value. Typically, predictable value is a value computed using the *reference circuit* and it depends on a small fraction of the key. Our aim is to "hide" predictable values among unpredictable values. The unpredictability of computed functions may only come from the secret key/randomness used during the obfuscation process. In order to develop a formal attack model we allow the obfuscated circuit to use *random* bits. We underline that randomness here is merely an abstraction required for provable security arguments.

In the real whitebox implementation the random bits may be implemented as *pseudorandom* values computed from the input. Of course the pseudorandom generation part has to be protected as well. However, the white-box designer is free to choose arbitrary pseudorandom generator and its protection is an easier task then obfuscating a general circuit. For example, the designer can choose a random circuit satisfying some basic properties like computing a balanced function. The resulting circuit protected against the algebraic attack using pseudorandomly generated bits must further be obfuscated and protected from *removal* of the pseudorandomness. This is type of protection that we called *structure hiding* in Section 9.2 and it is out of the scope of this work. It is indeed a challenging problem.

There is a strong similarity between the algebraic attack and the side channel probing attack. In the t-th order probing attack the adversary may observe t intermediate values computed in the circuit. In the d-th order algebraic attack the adversary has access to all intermediate values but she can combine them only with a function of degree at most d.

Semantic Security. The main idea of masking schemes is to *hide the values* computed in the reference circuit using (pseudo)random masks. We assume that the adversary knows the reference circuit. Given the inputs (e.g. a plaintext and a key) she can compute all intermediate values. The final goal of the adversary is to recover the key of an obfuscated implementation or, at least, learn some partial information about it. To formalize this, we adapt classic semantic security and indistinguishability ideas. The adversary may ask to encrypt two different vectors of inputs. The challenger chooses randomly one of the vectors and provides an oracle modelling the algebraic attack to the adversary. The goal of the adversary is to decide which of the vectors was encrypted. If she can not do this, then she can not learn any information about the hidden inputs (e.g. the plaintext and the key). Note that the adversary is allowed to choose many different keys which is not possible in the white-box scenario. However, it leads to simpler definitions since we do not have to distinguish plaintext and key and we just treat them as one input. It is possible to add a constraint allowing to choose only a single key per input vector, but this would not lead to any improvement.

Algebraic Attack Oracle. The oracle modelling the algebraic attack should not reveal too much information about computed values. Otherwise, it may be possible for the adversary to reconstruct the obfuscated circuit and then we would arrive in the general white-box scenario. We model the attack as follows: the adversary chooses the target function among linear (or higher-order) combinations of the intermediate functions in the circuit and she tries to guess its values during encryptions of the inputs from one of the two vectors. Note that some functions may have strong correlation with some function of the input. For a small vector of inputs the adversary may simply guess the value, ask the oracle a few times until the guess is correct and then compute the correlations. However, in the real algebraic attack this is not possible due to presence of "noise" in the circuit. For a small number of plaintexts there will be a lot of false matches for any "predicted" value, because there are many different functions computed in the circuit and it is highly probable that there is a linear combination of them matching an arbitrary value. We take this into account and require that only the function chosen by the adversary has to match the predicted value. As a result, the adversary can not accurately predict values of any *single* function in the d-th order closure of the circuit functions in order to run the linear algebra attack.

Encoding and Decoding. The circuit in the model can not take the input as it is, because these values allow for a simple distinguisher. Since we are developing a masking scheme, we assume that he inputs are already masked using random shares. The adversary targets masked operations, which we call *critical computations*. This goes in parallel with the classic Boolean masking scenarios. We would like to stress that this is necessary in order to formally analyze the security of masked computations. Therefore, we do not consider the initial encoding and the final decoding processes. Indeed, these procedures are not relevant for the algebraic attack since they are not related to the reference circuit. Therefore, their protection is a part of the structure hiding component.

The security model is illustrated in Figure 9.1.

9.3.2 Prediction Security

Taking into account the above discussions, we propose the following game-based security definition:

Definition 9.1 (Prediction Security (d-PS)). *Let $C : \mathbb{F}_2^{N'} \times \mathbb{F}_2^{R_C} \to \mathbb{F}_2^M$ be a Boolean circuit, $E : \mathbb{F}_2^N \times \mathbb{F}_2^{R_E} \to \mathbb{F}_2^{N'}$ an arbitrary function, $d \geq 1$ an integer and $\mathcal{A}$ an adversary. Consider the following security game:*

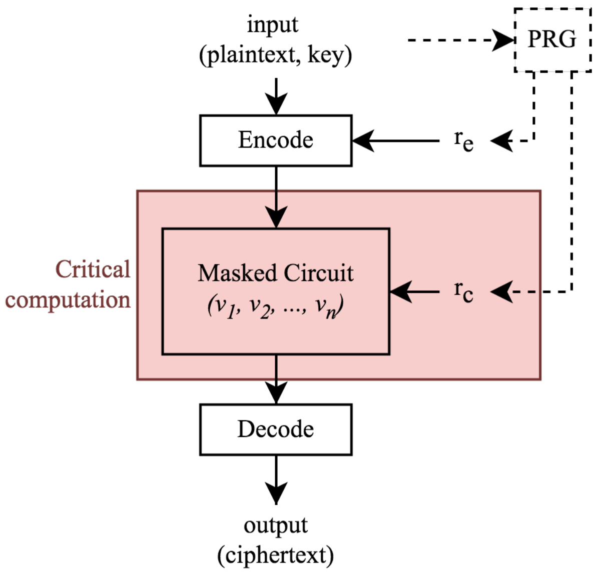


FIGURE 9.1: (Pseudo)randomness in a masked circuit.

```
**Experiment** $\mathsf{PS}^{C,E,d}(\mathcal{A}, b)$:
    $(\tilde{f}, x^{[0]}, x^{[1]}, \tilde{y}) \leftarrow \mathcal{A}(C, E, d)$, where
        $\tilde{f} \in \mathcal{F}^{(d)}(C) \setminus \{\mathbf{0}, \mathbf{1}\}, x^{[l]} = (x_1^{[l]}, \ldots, x_Q^{[l]}), x_i^{[l]} \in \mathbb{F}_2^N, \tilde{y} \in \mathbb{F}_2^Q$
    $(r_1, \ldots, r_Q) \overset{\$}{\leftarrow} (\mathbb{F}_2^{R_E})^Q$
    $(\tilde{r}_1, \ldots, \tilde{r}_Q) \overset{\$}{\leftarrow} (\mathbb{F}_2^{R_C})^Q$
    For any $f \in \mathcal{F}^{(d)}(C)$ define
        $y(f) = \left(f\left(E(x_1^{[b]}, r_1), \tilde{r}_1\right), \ldots, f\left(E(x_Q^{[b]}, r_Q), \tilde{r}_Q\right)\right)$
    $F \leftarrow \{f \in \mathcal{F}^{(d)}(C) \mid y(f) = \tilde{y}\}$
    **return** $F = \{\tilde{f}\}$
```

In the above experiment, $\overset{\$}{\leftarrow}$ *means sampling uniformly at random. Define the advantage of an adversary* $\mathcal{A}$ *as*

$$\mathsf{Adv}^{\mathsf{PS}}_{C,E,d}[\mathcal{A}] = \left| \mathsf{P}[\mathsf{PS}^{C,E,d}(\mathcal{A}, 0) = 1] - \mathsf{P}[\mathsf{PS}^{C,E,d}(\mathcal{A}, 1) = 1] \right|.$$

The pair (C, E) *is said to be d-th order prediction-secure (d-PS) if for any adversary* $\mathcal{A}$ *the advantage is negligible.*

Example 4. Consider a white-box AES implementation with a first-order Boolean masking protection. Assume that there are two nodes in the circuit computing two masks of an output bit of an S-Box in the first round. Denote the functions computed by masks as f_1, f_2. The adversary finds these nodes and chooses $\tilde{f} = f_1 \oplus f_2 \in \mathcal{F}^{(1)}(C)$. She also chooses sufficiently large Q and random vectors $x^{[0]}$ and $x^{[1]}$ of Q (plaintext, key) pairs. For example, the same key may be used for all pairs in $x^{[0]}$ and another key for all pairs in $x^{[1]}$. The adversary computes $\tilde{y} = (s(x_1^{[0]}), ..., s(x_Q^{[0]}))$ (where function s computes an output bit of the attacked S-Box in the first round from the plaintext and the key). In this case the adversary succeeds in the game with advantage close to 1 and the implementation is not prediction-secure (indeed, the adversary easily distinguishes the two keys). Note that we required the adversary to find the nodes in order to choose the right function $\tilde{f}$. Since the adversary is unbounded, this is just a technical requirement. In the real attack the adversary does not need to guess the exact function, only to generate a predicted vector of its values.

The function E in the definition should be referred to as *the encoding function*. Though the definition allows the encoding function to be arbitrary, we are mainly interested in the encodings with useful semantics, i.e. masking. Moreover, we expect the encoding to be lightweight and universal: main computations should be performed in the circuit C.

The circuit C can be completely unobfuscated but still prediction-secure, because the adversary is forced to consider the whole vector space $\mathcal{F}^{(d)}(C)$. In a real white-box implementation this restriction is expected to be enforced by the structure-hiding protection.

We now discuss possible attacks that are not covered by this definition. The definition ensures that any single function from $\mathcal{F}^{(d)}(C)$ is unpredictable. However, it may be possible that multiple functions jointly exhibit a behaviour that leads to an attack. For example, the dimension of $\mathcal{F}^{(d)}(C)$ may differ depending on the input being encoded. Though, such attack is related to the value-restriction method from Section 8.4.3. The definition also does not cover a general LPN-based attack.

Remark 20. The definition actually covers security against a simple LPN algorithm, which simply tries to guess the error vector and solve the error-less linear system. In general, security against any LPN algorithm can be achieved by increasing the number of unknowns and increasing the error probability. As will be shown further, the latter is harder and is the main difficulty. In fact, achieving a *constant error probability* for circuits of *arbitrary size* should be enough to guarantee security, given that the number of unknowns (i.e., the window size) can be increased arbitrarily by the structure hiding component.

9.3.3 Security Analysis

In the experiment both the encoding function E and the circuit C use randomness. However, the d-th order closure is computed only using functions from $\mathcal{F}(C)$. Still, the inputs of C include the outputs of E and that is how the randomness used in E affects the computations in C. In other words, E generates some distribution in the inputs of C. Therefore, in order to study functions from $\mathcal{F}^{(d)}(C)$ we need to compose them with E.

It is crucial to study how functions from $\mathcal{F}^{(d)}(C)$ composed with E behave on a *fixed input* x. Consider a function $f \in \mathcal{F}^{(d)}(C)$. If the function $f(E(x,\cdot),\cdot)$ is constant for some x and the function $f(E(x',\cdot),\cdot)$ is non-constant for some $x' \neq x$ (or is constant but $f(E(x,\cdot),\cdot) \neq f(E(x',\cdot),\cdot)$, then these inputs are distinguishable and the pair (C, E) is insecure[1]. More generally, if for some

[1]Unless $f(E(x',\cdot),\cdot)$ has extremely high bias and is indistinguishable from the constant function on practice.

$f \in \mathcal{F}^{(d)}(C) \setminus \{\mathbf{0}, \mathbf{1}\}$ and for some $x \in \mathbb{F}_2^N$ the function $f(E(x, \cdot), \cdot)$ is non-constant but has a high bias (i.e. it has very low or very high weight), then the adversary still may have high chances to predict its output. We conclude that for all functions $f \in \mathcal{F}^{(d)}(C) \setminus \{\mathbf{0}, \mathbf{1}\}$ and for all $x \in \mathbb{F}_2^N$ the function $f(E(x, \cdot), \cdot)$ should have a low bias.

We now show that this requirement is enough to achieve d-th order prediction security if there are enough random bits used in the main circuit. The following proposition gives an upper bound on d-PS advantage from the maximum bias and the number of random bits.

Definition 9.2. *Let C, E, d be defined as above. For any function $f \in \mathcal{F}^{(d)}(C) \setminus \{\mathbf{0}, \mathbf{1}\}$ and for any $x \in \mathbb{F}_2^N$ define $f_x : \mathbb{F}_2^{R_E} \times \mathbb{F}_2^{R_C} \to \mathbb{F}_2$ given by*

$$f_x(r_e, r_c) \coloneqq f(E(x, r_e), r_c)$$

and denote the set of all such functions $\mathcal{R}$:

$$\mathcal{R} \coloneqq \left\{ f(E(x, \cdot), \cdot) \;\middle|\; f \in \mathcal{F}^{(d)}(C) \setminus \{\mathbf{0}, \mathbf{1}\}, x \in \mathbb{F}_2^N \right\}.$$

Furthermore, let ε be the maximum absolute correlation among all functions from $\mathcal{R}$:

$$\varepsilon \coloneqq \max_{f_x \in \mathcal{R}} |f_x|.$$

The pair (C, E) is then said to be a d-th order algebraically ε-secure (ε-d-AS) scheme.

Proposition 9.3. *Let (C, E) be a d-th order algebraically ε-secure scheme. Let $e \coloneqq \log_2((1 + \varepsilon)/2)$. Then, for any adversary $\mathcal{A}$ choosing vectors of size Q*

$$\mathsf{Adv}^{\mathsf{PS}}_{C,E,d}[\mathcal{A}] \leq min(2^{Q - R_C}, 2^{eQ}). \tag{9.1}$$

Proof. First, we prove that $\mathsf{Adv}^{\mathsf{PS}}_{C,E,d}[\mathcal{A}] \leq 2^{Q - R_C}$. If $\tilde{f}$ chosen by the adversary is an affine function of random bits r (independent of x), then it is clear that the advantage in this case is zero. Otherwise, we compute the probability of the event when the predicted value $\tilde{y}$ matches some linear function of random bits r. There are R_C independent uniformly distributed random vectors $r_1, \ldots, r_{R_C}$ from $\mathbb{F}_2^Q$. Let p be the probability of the event that they span the whole space $\mathbb{F}_2^Q$. In this case the experiment returns 0, because any $\tilde{y}$ matches a function different from the one chosen by the adversary. The following holds (see e.g. [FJVP13]):

$$\begin{aligned} p &\coloneqq \Pr_{r_1, \ldots, r_{R_C} \xleftarrow{\$} \mathbb{F}_2^Q} [\operatorname{span}(r_1, \ldots, r_{R_C}) = \mathbb{F}_2^Q] = \prod_{i=0}^{Q-1} \left(1 - 2^{i - R_C}\right), \\ &\log_2(1 - p) \leq Q - R_C. \end{aligned}$$

We conclude that $p \geq 1 - 2^{Q - R_C}$ and the advantage is upper bounded by $2^{Q - R_C}$.

Now we prove that $\mathsf{Adv}^{\mathsf{PS}}_{C,E,d}[\mathcal{A}] \leq 2^{eQ}$. We simply bound the probability that the adversary submits $\tilde{f}, \tilde{y}$ such that $y(\tilde{f}) = \tilde{y}$ in the experiment. Since elements of $y(\tilde{f})$ are independent, the probability to have $y(\tilde{f}) = \tilde{y}$ is maximized when each bit of $\tilde{y}$ equals to the most probable value of the respective bit of $y(\tilde{f})$ (the adversary would also need to use the least probable value at least once to avoid matching with the constant functions). For each bit the probability is bounded by $(1+\varepsilon)/2 = 2^e$, therefore for Q bits the upper bound is 2^{eQ}. $\square$

Note that the bounds are quite loose. The randomness-based term takes into account only single random bits from r_c. The randomness in the inputs of C (generated from r_e in the encoding process) as well as all intermediate values computed in the circuit add much more noise (note that we can not directly include r_e since it is used in the encoding process and not in the main circuit). The correlation-based term bounds only the probability of predicting the output for a single vector of inputs. It does not include the cost of distinguishing the two vectors. We stick to these loose bounds as our current goal is to provide a simple and sound provably secure protection. Assume that we know the maximum absolute correlation ε in $\mathcal{R}$ and we want to achieve a better security bound. We can always add "dummy" random bits to the circuit. Note that this leads to stronger requirements for the structure-hiding protection. It follows that given the maximum bias, we can compute how many "dummy" random bits are needed to achieve any required security level:

Corollary 9.4. *Let k be a positive integer. Then for any adversary $\mathcal{A}$*

$$\mathsf{Adv}^{\mathsf{PS}}_{C,E,d}[\mathcal{A}] \le 2^{-k} \textit{ if}$$

$$\varepsilon < 1 \textit{ and } R_C \ge k \cdot (1 - \frac{1}{e}).$$

Proof. Consider each term of the bound from Proposition 9.3:

$$Q - R_C \le -k \text{ or } eQ \le -k.$$

The result follows from the second term if $Q \ge -\frac{k}{e}$ (note that e is negative when $\varepsilon < 1$). To cover all other Q we need

$$R_C \ge Q + k \ge k \cdot (1 - \frac{1}{e}).$$

$\square$

Remark 21. The advantage bound is information-theoretic as we do not constraint the adversary's powers. This is an effect of the attack formalization given in Definition 9.1: the attack requires that the adversary predicts the chosen function precisely. An unbounded adversary could simply iterate over all functions $f \in \mathcal{F}^{(d)}(C)$ and e.g. compute the bias. We argue that this kind of attack is not the linear algebra attack that we consider. Furthermore, the attack model restricts the adversary to use the full circuit C. Without this restriction it would be possible to choose a part of the circuit (a *window*) to reduce the noise. In our model we expect that a structure-hiding protection is used to prevent this.

9.3.4 First-order Secure Construction

Given the notion of prediction security we are now interested in developing secure constructions. A common strategy is to develop small secure circuits (called *gadgets*) and compose them in a provably secure way. The definition of prediction security does not immediately lead to composability, because it includes the encoding step which is not expected to be present in the intermediate gadgets. In order to proceed, we split up the prediction security into circuit security and encoding security. The new notions are stronger in order to get proofs of secure composability. They are limited to the first-order security ($d = 1$) and it is not obvious how to extend them to higher orders.

Definition 9.5 (Circuit Algebraic Security (ε-1-AS)).
Let $C(x, r) : \mathbb{F}_2^{N'} \times \mathbb{F}_2^{R_C} \to \mathbb{F}_2^M$ be a Boolean circuit. Then C is called first-order algebraically ε-secure (ε-1-AS) if for any $f \in \mathcal{F}^{(1)}(C) \setminus \{\mathbf{0}, \mathbf{1}\}$ one of the following conditions holds:

1. *f is an affine function of x,*
2. *for any $x \in \mathbb{F}_2^{N'}$, $|\mathbf{cor}(f(x, \cdot))| \leq \varepsilon$, where $f(x, \cdot) : \mathbb{F}_2^{R_C} \to \mathbb{F}_2$.*

Definition 9.6 (Encoding Algebraic Security (ε-1-AS)).
Let $E(x, r) : \mathbb{F}_2^N \times \mathbb{F}_2^{R_E} \to \mathbb{F}_2^{N'}$ be an arbitrary encoding function. Let $\mathcal{Y}$ be the set of the coordinate functions of E (i.e. functions given by the outputs bits of E). The function E is called a first-order algebraically ε-secure encoding (ε-1-AS) if for any $f \in \mathcal{Y}^{(1)} \setminus \{\mathbf{0}, \mathbf{1}\}$ and for any $x \in \mathbb{F}_2^N$,

$$|\mathbf{cor}(f(x, \cdot))| \leq \varepsilon,$$

where $f(x, \cdot) : \mathbb{F}_2^{R_E} \to \mathbb{F}_2$.

The following proposition shows that if both an encoding and a circuit are algebraically secure, then their combination is prediction-secure:

Proposition 9.7. *Let $C(x', r) : \mathbb{F}_2^{N'} \times \mathbb{F}_2^{R_C} \to \mathbb{F}_2^M$ be a Boolean circuit and let $E(x, r) : \mathbb{F}_2^N \times \mathbb{F}_2^{R_E} \to \mathbb{F}_2^{N'}$ be an arbitrary encoding function.*

If C is ε_C-1-AS circuit and E is ε_E-1-AS encoding, then the pair (C, E) is a $\max(\varepsilon_C, \varepsilon_E)$-1-AS scheme.

Proof. If the function $\tilde{f}$ chosen by the adversary is an affine combination of the input x' of C, then the encoding security of E applies leading to the bound with $\varepsilon = \varepsilon_E$. Otherwise, ε_C-1-AS security of C provides the bound with $\varepsilon = \varepsilon_C$ (the ε_C bound applies for any fixed input x' of C, therefore it applies for any distribution of x' generated by E as well). □

Finally, we show that ε-1-AS circuits are composable, i.e. are secure gadgets. We can compose gadgets in arbitrary ways and then join the final circuit with a secure encoding function to obtain a prediction-secure construction.

Proposition 9.8 (ε-1-AS Composability). *Consider ε-1-AS circuits $C_1(x_1, r_1)$ and $C_2(x_2, r_2)$. Let C be the circuit obtained by connecting the output of C_1 to the input x_2 of C_2 and letting the input r_2 of C_2 be the extra input of C:*

$$C(x_1, (r_1, r_2)) \coloneqq C_2(C_1(x_1, r_1), r_2).$$

Then $C(x_1, (r_1, r_2))$ is also a ε-1-AS circuit.

Proof. Consider an arbitrary function $\tilde{f}(x_1, r_1, r_2) \in \mathcal{F}^{(1)}(C)$. By linearity, it can be written as $u \oplus v$, where $u \in \mathcal{F}^{(1)}(C_1)$ and v is a function from $\mathcal{F}^{(1)}(C_2)$ composed with C_1 (by connecting the output of C_1 to the input x_2 of C_2). Since C_2 is ε-1-AS, v is either an affine function of x_2 (which belongs to $\mathcal{F}^{(1)}(C_1)$) or $|\mathbf{cor}(v)|$ is not greater than ε when x_2 is fixed (i.e. when x_1, r_1 are fixed). In the first case, we get that $\tilde{f}$ belongs to $\mathcal{F}^{(1)}(C_1)$ and security follows from ε-1-AS security of C_1. In the second case, observe that the absolute correlation of v can not exceed ε for any fixed x_2 and, therefore, it can not exceed ε for any distribution of x_2. Moreover, u is independent from r_2. Therefore, for $\tilde{f} = u \oplus v$ it follows that $|\mathbf{cor}(\tilde{f})| \le |\mathbf{cor}(v)| \le \varepsilon$ since C_2 is an ε-1-AS circuit. □

This result shows that due to frequent use of fresh randomness it is guaranteed that the maximum bias does not grow when we build large algebraically secure circuits from smaller ones. It means that ε-1-AS circuits offer a solid protection against the LPN-based variant of the algebraic attack as well. The complexity of LPN algorithms grows exponentially with the number of unknowns. Therefore, increasing the number of random nodes as suggested by the Corollary 9.4 allows to reach any required level of security against LPN attacks at the same time. Exact required number of random nodes depends on the value of ε and chosen LPN algorithm.

9.3.5 Verifying Algebraic Security

Proposition 9.8 shows that we can compose algebraically secure circuits. Large circuits can be constructed from a set of *gadgets* - small algebraically secure circuits with some useful semantics. In order to design new gadgets we need to be able to check their algebraic security. The simplest way to get a bound on the absolute correlation is based on the algebraic degree of computed functions: the minimum weight of a *nonzero* function of n bits of degree d is equal to 2^{n-d} (see e.g. [Car10a]). Therefore, we can think about the following algorithm for checking a circuit $C(x, r_C)$: for any fixed input x compute the ANFs of the functions computed in $C(x, \cdot)$ (functions of r_C) and return the maximum observed degree. The degree does not grow when functions are combined linearly. Therefore, the absolute correlation bound can not grow as well, except when the resulting function is constant in which case the absolute correlation is maximal and the gadget may be insecure. As a result, our method for verifying algebraic security splits into two parts:

1. verify that there is no absolute correlation equal to 1 among restrictions of functions from $\mathcal{F}^{(1)}(C)$ except the constant functions and affine functions of x;

2. compute the maximum degree among all restrictions of the intermediate functions and compute the corresponding correlation bound.

The second step is straight-forward. We describe an algorithm that solves the first step.

Consider a circuit $C(x, r) : \mathbb{F}_2^N \times \mathbb{F}_2^R \to \mathbb{F}_2^M$. For all $c \in \mathbb{F}_2^N$ let L_c be the linear map that returns the restriction $x = c$ of a function f from $\mathcal{F}^{(1)}(C)$ (e.g. if functions are represented as truth table vectors then L_c returns the truth table entries corresponding to the case $x = c$). Note that the domain of L_c is defined to be the subspace $\mathcal{F}^{(1)}(C)$.

We now give an equivalent condition for the first part of the verification. It serves as a basis for the verification algorithm given in Algorithm 9.1.

Proposition 9.9. *The circuit C is ε-1-AS for some $\varepsilon < 1$ if and only if for all c the following holds:*

$$\dim \ker L_c = N. \tag{9.2}$$

Proof. For any $c \in \mathbb{F}_2^N$ let F_c be the subspace of $\mathcal{F}^{(1)}(C)$ containing functions that are constant when x is fixed to c. Also let $F = \bigcup_c F_c$. $\varepsilon < 1$ requires that any $f \in \mathcal{F}^{(1)}(C)$ either belongs to $\mathcal{X}^{(1)}(C)$ or is non-constant for any fixed x. It is equivalent to require that F is equal to $\mathcal{X}^{(1)}(C)$. Note that each F_c includes $\mathcal{X}^{(1)}(C)$ as a subset. Therefore, $F = \bigcup_c F_c$ is equal to $\mathcal{X}^{(1)}(C)$ if and only if for all c $F_c = \mathcal{X}^{(1)}(C)$. Since these are linear subspaces then we can compare their dimensions.

$\mathcal{X}^{(1)}(C)$ is spanned by all x_i and the constant-1 function:

$$\dim \mathcal{X}^{(1)}(C) = N + 1; \tag{9.3}$$

The constant-1 function always belongs to $\mathcal{F}^{(1)}(C)$ and to any of the F_c. The subspace of functions that are constant on the restriction can be obtained by adding the constant-1 function to the subspace of functions that are equal to zero on the restriction:

$$F_c = \ker L_c \oplus \{\mathbf{0}, \mathbf{1}\}, \tag{9.4}$$

$$\dim F_c = \dim \ker L_c + 1. \tag{9.5}$$

By comparing the dimensions obtained in Equation 9.3,Equation 9.5 we prove the proposition. □

The algorithm operates on functions using their truth tables. The truth tables are obtained by evaluating the circuit on all possible inputs and recording the values computed in each node. The set of computed truth tables corresponds to $\mathcal{F}(C)$. By removing redundant vectors we can compute a basis $\mathcal{B}$ of $\mathcal{F}^{(1)}(C)$ (and also ensure presence of the constant-1 vector). Then, for each c we take the part of each basis vector that corresponds to the fixed $x = c$ (and r taking all possible values). These parts form the subspace $\operatorname{Im} L_c$. We compute a basis $\mathcal{B}_c$ of these parts. Finally, we verify that

$$\dim \ker L_c = \dim \mathcal{F}^{(1)}(C) - \dim \operatorname{Im} L_c = |\mathcal{B}| - |\mathcal{B}_c| = N. \tag{9.6}$$

The algorithm is implemented in SageMath [SD19] and is publicly available in [BU18b].

Algorithm 9.1 Verification of Algebraic Security

```
Input: a Boolean circuit $C(x, r) : \mathbb{F}_2^N \times \mathbb{F}_2^R \to \mathbb{F}_2^M$;
Output: Secure if the circuit $C$ is $\varepsilon$-1-AS for some (unknown) $\varepsilon < 1$,
        Insecure otherwise.
 1: evaluate $C$ on all possible inputs;
 2: associate the vector of computed values to each node of $C$;
 3: let $\mathcal{V}$ be the set of all associated vectors;
 4: let $\mathcal{B}$ be a basis of $\mathcal{V}^{(1)}$;
 5: for all $c \in \mathbb{F}_2^N$ do
 6:     let $\mathcal{V}_c$ be the set of all vectors from $\mathcal{B}$ restricted to the case of $x = c$;
 7:     let $\mathcal{B}_c$ be a basis of $\mathcal{V}_c^{(1)}$;
 8:     if $|\mathcal{B}| - |\mathcal{B}_c| \neq N$ then
 9:         return Insecure;
10: return Secure.
```

Complexity analysis. The truth tables have size 2^{N+R} bits. Computing the basis of $\mathcal{F}^{(1)}(C)$ takes time $\mathcal{O}(min(2^{N+R}, |C|)^{\omega})$. The same holds for $\operatorname{Im} L_c$ except that the vectors have size 2^R and for small R this can be done more efficiently. The total complexity is $\mathcal{O}(min(2^{N+R}, |C|)^{\omega} + 2^N min(2^R, |C|)^{\omega})$. Recall that by Proposition 8.5 we should consider only the nonlinear nodes of the circuit.

9.4 Minimalist Quadratic Masking Scheme

In this section I show a first-order algebraically secure quadratic masking scheme. Then I describe concrete circuits which can be verified to be first-order algebraically secure gadgets using Algorithm 9.1.

Minimalist Quadratic Masking.

Since the decoding function has to be at least quadratic, we need at least two bits to encode a single bit. For two bits all nonlinear decoding functions are linear equivalent to a quadratic monomial being simply the product of the two input bits. Unfortunately, this decoding function is vulnerable to the linear algebra attack. Any quadratic function with 2-bit input is unbalanced. Therefore, one of the reference bit values can be encoded by 3 different values and the other value has only 1 possible encoding. For example, if the value is equal to 1 and the decoding function is simply AND, the input has to be equal to (1, 1). In this case there is no randomness involved and the hidden value is leaked. The conclusion is that any value of the original bit should include randomness in its encoding. In particular, the decoding function can not be a point function.

We move on to 3-bit encodings. The simplest quadratic function using all 3 input bits a, b, c is $ab \oplus c$. Note the similarity with the broken 2-bit scheme: the quadratic monomial ab is simply linearly masked by c. However, this linear mask is enough to prevent the attack: in this case $Decode(a, b, c) = 1$ does not imply $a = 1$ or $b = 1$. In fact, such $Decode$ is balanced: both

$$\begin{aligned}
Encode(x, r_a, r_b) &= (r_a, r_b, r_a r_b \oplus x), && (9.7)\\
Decode(a, b, c) &= ab \oplus c, && (9.8)\\
Eval_{XOR}((a, b, c), (d, e, f)) &= (a \oplus d,\ b \oplus e,\ ae \oplus bd \oplus c \oplus f), && (9.9)\\
Eval_{AND}((a, b, c), (d, e, f)) &= (ae,\ bd,\ (cd)e \oplus a(bf) \oplus cf), && (9.10)\\
Refresh((a, b, c), (r_a, r_b)) &= (a \oplus r_a,\ b \oplus r_b,\ c \oplus r_a b \oplus r_b a \oplus r_a r_b). && (9.11)
\end{aligned}$$

FIGURE 9.2: An Insecure Quadratic Masking Scheme.

0 and 1 have exactly 4 preimages. We first describe an insecure yet simple masking scheme based on this decoding function in Figure 9.2. It is easy to verify that $Eval_{XOR}$ and $Eval_{AND}$ satisfy the requirements from Definition 8.2. In addition, $Refresh(a, r)$ returns fresh random encoding of a, meaning that $Decode(a) = Decode(Refresh(a, r))$ for any r and new encoding reveals no information about the old encoding.

We now observe that $Refresh$ is not ε-1-AS for any $\varepsilon < 1$: the computed term $r_a b$ is constant when b is fixed to 0 and equals to r_a otherwise (leading to $\varepsilon = 1$). This can be fixed by using an extra random bit r_c to mask a, b through the computations:

$$\begin{aligned}
&Refresh((a, b, c), (r_a, r_b, r_c)) = \\
&\big(a \oplus r_a,\ b \oplus r_b,\ c \oplus r_a(b \oplus r_c) \oplus r_b(a \oplus r_c) \oplus (r_a \oplus r_c)(r_b \oplus r_c) \oplus r_c\big). \quad (9.12)
\end{aligned}$$

The new $Refresh$ function can be verified to be secure using the algorithm from Section 9.3.5. Moreover, the circuit computing $Eval_{XOR}$ applied to refreshed inputs is secure as well. However, $Eval_{AND}$ is not secure even if composed with the fixed $Refresh$ gadget. Consider the linear combination of computed terms $a(bf) \oplus cf = (ab \oplus c)f$. Here the variables are refreshed masks and can not be fixed by the adversary. However, the refreshing function does not change the hidden value. Therefore, $ab \oplus c$ would be equal to the value hidden by initial non-refreshed shares which can be fixed. Fixing the hidden value to 0 makes the combination $f(ab \oplus c)$ equal to 0 and be equal to the random share f when the hidden value is fixed to 1. We observe that it is possible to use a trick similar to the one used to fix the $Refresh$ function. In fact, the extra random shares added to fix the $Refresh$ function may be reused to fix the $Eval_{AND}$ function. As a result, we obtain a fully secure masking scheme. The complete description is given in Algorithm 9.2.

Security. First, we verify $Eval_{XOR}$ and $Eval_{AND}$ gadgets using Algorithm 9.1. We obtain that they are ε-1-AS circuits for some $\varepsilon < 1$. Then we construct the ANFs of intermediate functions. The maximum degree is equal to 4. It is achieved for example in the term cf in the gadget $Eval_{AND}$: its ANF contains the term $r_a r_b r_d r_e$. Therefore, $Eval_{AND}$ is ε-1-AS with $\varepsilon \leq 7/8$. The gadget $Eval_{XOR}$ has degree 2 and is 1/2-1-AS. Unfortunately, we do not have a pen-and-paper proof for security of the gadgets and rely solely on the verification algorithm (which is able to spot the described weaknesses in the insecure versions of the gadgets).

Verifying security of the encoding function *Encode* can be done in the same way. Clearly, no linear combination of $r_a, r_b, r_a r_b \oplus x$ is constant for any fixed x. The coordinate $r_a r_b \oplus x$ has degree 2 and its absolute correlation is equal to 1/2. Therefore, *Encode* is an ε-1-AS encoding with $\varepsilon = 1/2$.

By applying Proposition 9.7, we obtain that for any adversary $\mathcal{A}$, for any circuit C build from the gadgets $Eval_{XOR}, Eval_{AND}$ and for the described *Encode* encoding we have:

$$\mathsf{Adv}^{\mathsf{PS}}_{C,E,d}[\mathcal{A}] \le min(2^{Q-R_C}, 2^{eQ}), \tag{9.13}$$

where $e = \log_2 (1 + 7/8)/2 \approx -0.093$. According to Corollary 9.4, in order to achieve provable 80-bit security we need to have $R_C \ge 80(1 - 1/e) \approx 940$ random bits in the circuit. Note that it does not depend on the actual size of the circuit, i.e. 940 random bits are enough for an arbitrary-sized circuit. However, the adversary should not be able to shrink the window so that it contains less than 940 random bits. This is expected to be guaranteed by a structure hiding protection. Finally, we remark that the bounds are rather loose and more fine-grained analysis should improve the bound significantly.

9.4.1 Implementation

We applied our masking scheme to an AES-128 implementation to estimate the overhead. Our reference AES circuit contains 31,783 gates. It is based on Canright's S-Box implementation [Can05] and naive implementation of MixColumns. After applying our nonlinear masking scheme and a first-order linear masking scheme on top the circuit expands to 2,588,743 gates of which 409,664 gates are special gates modeling external random bits. The circuit can be encoded in 16.5 MB. Extra RAM needed for computations is less than 1KB. On a common laptop it takes 0.05 seconds to encrypt 1 block. Since the implementation is bitwise, 64 blocks can be done in parallel at the same time on 64-bit platforms. There is still a large room for optimizations. We used the Daredevil CPA tool [HBE+16] to test our implementation. Due to the first-order linear masking on top we did not detect any leakage. Pure nonlinear masking scheme does leak the key so the combination of both is needed as we suggested in Section 9.2. The implementation code is publicly available [BU18b]. We remark that it is a proof-of-concept and not a secure white-box implementation; it can be broken in various ways.

9.5 Conclusions

In this chapter we investigated the possibility of using masking techniques for white-box implementations. We presented several attacks applicable in different scenarios. As a result we obtained requirements for a masking scheme to be useful. We divided the requirements into value hiding and structure hiding protections. Furthermore, we suggested that value hiding may be achieved using an algebraically secure nonlinear masking scheme and a classic linear masking scheme. We developed a framework for provable security against the algebraic attack and proposed a concrete provably secure first-order masking scheme. Therefore, a value hiding protection can be implemented.

We believe that our work opens new promising directions in obfuscation and white-box design. We focused on value hiding protection and developed a first-order protection against the algebraic attack. The natural open question is developing higher-order countermeasures for the algebraic attack. Another direction is to study structure hiding countermeasures. Finally, it seems that pseudorandom generators play an important role in white-box obfuscation and are useful at all layers of protection. Randomness helps to develop formal security models and pseudorandom generators bridge the gap between theoretical constructions and real world implementations. Therefore, designing an easy-to-obfuscate pseudorandom generators is another important open problem.

Algorithm 9.2 Minimalist Quadratic Masking Scheme.

```
1: function Encode($x, r_a, r_b$)
2:     return $(r_a, r_b, r_a r_b \oplus x)$

3: function Decode($a, b, c$)
4:     return $ab \oplus c$

5: function EvalXOR($(a, b, c), (d, e, f), (r_a, r_b, r_c), (r_d, r_e, r_f)$)
6:     $(a, b, c) \leftarrow$ Refresh($(a, b, c), (r_a, r_b, r_c)$)
7:     $(d, e, f) \leftarrow$ Refresh($(d, e, f), (r_d, r_e, r_f)$)
8:     $x \leftarrow a \oplus d$
9:     $y \leftarrow b \oplus e$
10:    $z \leftarrow c \oplus f \oplus ae \oplus bd$
11:    return $(x, y, z)$

12: function EvalAND($(a, b, c), (d, e, f), (r_a, r_b, r_c), (r_d, r_e, r_f)$)
13:    $(a, b, c) \leftarrow$ Refresh($(a, b, c), (r_a, r_b, r_c)$)
14:    $(d, e, f) \leftarrow$ Refresh($(d, e, f), (r_d, r_e, r_f)$)
15:    $m_a \leftarrow bf \oplus r_c e$
16:    $m_d \leftarrow ce \oplus r_f b$
17:    $x \leftarrow ae \oplus r_f$
18:    $y \leftarrow bd \oplus r_c$
19:    $z \leftarrow a m_a \oplus d m_d \oplus r_c r_f \oplus cf$
20:    return $(x, y, z)$

21: function Refresh($(a, b, c), (r_a, r_b, r_c)$)
22:    $m_a \leftarrow r_a \cdot (b \oplus r_c)$
23:    $m_b \leftarrow r_b \cdot (a \oplus r_c)$
24:    $r_c \leftarrow m_a \oplus m_b \oplus (r_a \oplus r_c)(r_b \oplus r_c) \oplus r_c$
25:    $a \leftarrow a \oplus r_a$
26:    $b \leftarrow b \oplus r_b$
27:    $c \leftarrow c \oplus r_c$
28:    return $(a, b, c)$
```

Part IV

Design of Symmetric-key Algorithms

In this part, I present the work I have done on the design of symmetric-key primitives. The current trend in the design of cryptographic primitives is *lightweight cryptography*. Lightweight cryptography targets small devices (e.g. microcontrollers, smart cards, RFID tags). These devices are very constrained in resources, and it is necessary to minimize memory usage, code size, energy consumption, time of computation. Lightweight cryptography lowers the security margin in order to obtain more efficient cryptosystems. Another reason supporting the lightweight trend is that many existing designs survived many years of cryptanalysis, and there were no breakthrough techniques in cryptanalysis for a long time. Therefore, designing a secure primitive is a problem with many existing solutions, and these solutions have to compete by other properties, e.g. *lightweightness*.

I participated in the design of the SPARX family of block ciphers [DPU+16] and the SPARKLE cryptographic permutation [BBdS+19b]. I and my colleagues further used SPARKLE and the sponge construction to design the hash function family Esch and authenticated encryption family Schwaemm. My main contributions were in the security evaluations of the designs.

It has become a standard requirement for a symmetric-key design to include a proof against linear and differential cryptanalysis. The designers of AES, the current block cipher standard, used the so-called *wide trail argument* for the proof. It is a quite effective argument for block ciphers with strong, small S-Boxes and strong, heavy linear layers. However, it fails for ARX-based designs, i.e. designs build only from addition, rotation, and XOR operations. Such designs have certain advantages, such as better resistance against side-channel attacks and better performance in software. My colleagues came up with a novel way to prove security against linear and differential attacks, called a *long trail argument*. It is effective for designs using light linear layers and light but large S-Boxes. I designed an algorithm for applying the long-trail argument to a particular subset of SPN structures. We used this algorithm to evaluate a large class of linear layer candidates for the block cipher SPARX, together with the division property [Tod15] for security evaluation against integral cryptanalysis. I also used the algorithm to evaluate the security of SPARKLE, a cryptographic permutation based on SPARX, that I and my colleagues designed for the NIST call for lightweight cryptography.

Chapter 10

The SPARX Family of Block Ciphers

In this chapter, I describe the SPARX family of block ciphers. It is a joint work with my coauthors Daniel Dinu, Léo Perrin, Vesselin Velichkov, Johann Großschädl, and Alex Biryukov [DPU+16]. SPARX is the first ARX-based block cipher with provable security against linear and differential cryptanalysis. The design is a motivated by the novel *long-trail strategy*. My contributions are designing an algorithm for long-trail evaluation, evaluation of potential linear layers and integral cryptanalysis based on division property.

10.1 Introduction

Lightweight cryptography is a modern direction in the design of symmetric-key primitives. It aims to provide cryptographic security with constrained resources. Lightweight ciphers usually have a low security margin against unknown attacks and rely on the cryptanalysis done in the design phase.

My colleagues developed a framework for benchmarking lightweight ciphers, called FELICS [DCK+16, DBG+15, DCK+15]. A large amount of implementations for 3 target platforms - AVR, MSP, ARM - was collected and benchmarked. The leading block ciphers were Chaskey [MMH+14], Simon and Speck [BSS+13], RECTANGLE [ZBL+14], LEA [HLK+13], HIGHT [HSH+06], AES [DR98]. Chaskey is an Even-Mansour block cipher and has a data-security trade-off; it does not have a security proof against linear/differential attacks. Simon and Speck were designed by the NSA and do not have a proof too.

As the top designs are ARX-based, i.e. they are composed from Addition, Rotation and XOR operations, we decided to design an ARX-based block cipher. However, the current *wide-trail strategy* for proving security against linear/differential cryptanalysis does not apply well to ARX-based block ciphers. For this reason, we developed a novel *long-trail strategy*. As a result, the block cipher SPARX is the first ARX-based block cipher with provable security against single-trail linear and differential cryptanalysis.

In this chapter, I describe briefly the long-trail strategy. Afterward, I describe my contributions to the design. I developed an algorithm for efficient long-trail evaluation of a large class of SPN structures. We used this algorithm and the division property [Tod15] to evaluate a large class of potential linear layers. Interestingly, a Feistel-like linear layer turned out to provide an optimal balance between the linear/differential and integral attacks resistance, lightweightness of the primitive and simplicity. A few alternative linear layers seem to be a good choice as well.

10.1.1 Outline

I describe briefly the long-trail strategy and my algorithms in Section 10.2. In Section 10.3, I describe the procedure that we used to choose an optimal linear layer for the cipher. I omit the specification of SPARX, because it is not required for the contents of this chapter; it can be found in [DPU+16].

10.2 The Long-Trail Strategy

Linear and differential cryptanalysis are powerful methods of attacking block ciphers. It became a standard for new designs to be accompanied with arguments for security against linear and differential cryptanalysis.

The goal of linear and differential cryptanalysis is to find a *distinguisher* of a cryptographic function.

Definition 10.1 (Linear and Differential Distinguishers)**.** *Let* $E\colon \mathbb{F}_2^n \to \mathbb{F}_2^n$.

A pair $\alpha_{in}, \alpha_{out} \in \mathbb{F}_2^n$ *is called a* linear distinguisher *of* E *with linear correlation*

$$\begin{gathered}\mathsf{LC}_E(\alpha_{in}, \alpha_{out}) := 2^{-n}\mathsf{LAT}_E(\alpha_{in}, \alpha_{out}),\\ \textit{if } |\mathsf{LC}_E((\alpha_{in}, \alpha_{out})| \gg 2^{-n/2}.\end{gathered}$$

A pair $\alpha_{in}, \alpha_{out} \in \mathbb{F}_2^n$ *is called a* differential distinguisher *of* E *with differential probability*

$$\begin{gathered}\mathsf{DP}_E(\alpha_{in}, \alpha_{out}) := \mathsf{DDT}_E(\alpha_{in}, \alpha_{out}),\\ \textit{if } \mathsf{DP}_E(\alpha_{in}, \alpha_{out}) \gg 2^{-n}.\end{gathered}$$

For a keyed permutation $E_k(x)\colon \mathbb{F}_2^n \times \mathbb{F}_2^\kappa \to \mathbb{F}_2^n$, *a pair* $\alpha_{in}, \alpha_{out} \in \mathbb{F}_2^n$ *is a linear/differential distinguisher if for a large enough fraction of keys* $k \in \mathbb{F}_2^\kappa$, $(\alpha_{in}, \alpha_{out}$ *is a linear/differential distinguisher of* E_k.

Cryptographic functions are in most cases built in an iterated way. Intermediate values are analyzed and included in the distinguisher. The iterations of the round function are *assumed to be independent* and linear/differential distinguishers of each round are linked in a chain, called *a trail.*

Definition 10.2 (Linear and Differential trail).
Let $\mathbf{f} = (f_1, \ldots, f_r)$*,* $f_i \colon \mathbb{F}_2^n \to \mathbb{F}_2^n$*. A* trail *over* $\mathbf{f}$ *is a sequence* $\boldsymbol{\alpha}$ *of* $r+1$ *vectors:*

$$\boldsymbol{\alpha} = (\alpha_0, \ldots, \alpha_r), \alpha_i \in \mathbb{F}_2^n.$$

The expected differential probability *of the trail* $\boldsymbol{\alpha}$ *is defined as*

$$\mathsf{EDTP}_{\mathbf{f}}(\boldsymbol{\alpha}) \coloneqq \prod_{i=1}^{r} \mathsf{DP}_{f_i}(\alpha_{i-1}, \alpha_i).$$

The expected linear correlation *of the trail* $\boldsymbol{\alpha}$ *is defined as*

$$\mathsf{ELTC}_{\mathbf{f}}(\boldsymbol{\alpha}) \coloneqq \prod_{i=1}^{r} \mathsf{LC}_{f_i}(\alpha_{i-1}, \alpha_i).$$

In order to ensure that a cryptographic primitive is secure against trail-based differential/linear cryptanalysis, it is necessary to prove an upper-bound of the maximum EDTP and ELTC among all trails.

Definition 10.3. *Let* $\mathbf{f} = (f_1, \ldots, f_r)$*,* $f_i \colon \mathbb{F}_2^n \to \mathbb{F}_2^n$*.*
The maximum expected differential trail probability *of* $\mathbf{f}$ *is denoted* $\mathsf{MEDTP}(\mathbf{f})$ *and is equal to:*

$$\mathsf{MEDTP}(\mathbf{f}) \coloneqq \max_{\boldsymbol{\alpha} \in (\mathbb{F}_2^n)^{r+1}, \boldsymbol{\alpha} \neq 0} \mathsf{EDTP}_{\mathbf{f}}(\boldsymbol{\alpha}).$$

The maximum expected linear trail correlation *of* $\mathbf{f}$ *is denoted* $\mathsf{MELTC}(\mathbf{f})$ *and is equal to:*

$$\mathsf{MELTC}(\mathbf{f}) \coloneqq \max_{\boldsymbol{\alpha} \in (\mathbb{F}_2^n)^{r+1}, \boldsymbol{\alpha} \neq 0} \mathsf{ELTC}_{\mathbf{f}}(\boldsymbol{\alpha}).$$

10.2.1 The Wide-Trail Argument

The wide-trail strategy is the main method of proving an upper bound on the MEDTP and MELTC of a cryptographic primitive. It was introduced by Daemen and Rijmen [DR02] and was used to argue about the security of AES against linear and differential attacks.

I describe the argument for the differential trail cryptanalysis, the linear case is completely analogous.

Consider an SPN structure and a trail $\boldsymbol{\alpha}$ with a nonzero MEDTP. Any difference propagates through the linear layer of the structure with probability 1. Furthermore, a zero difference propagates through an S-Box to a zero difference with probability 1. It follows that the MEDTP of the trail depends only on the differential probabilities of S-Boxes with nonzero input/output differences in the trail. Such S-Boxes are called *active* S-Boxes.

The idea of the wide-trail strategy is to prove a lower bound on the number of active S-Boxes in a trail. Then, the differential uniformity of the S-Box is used to obtain an upper bound on the expected differential probability of a trail, i.e. the MEDTP. This is done by simply raising the minimum differential probability of the S-Box to the power of the minimum number of active S-Boxes. The first step is usually done by proving strong diffusion properties of the linear layer. For example, the MixColumns operation in the AES has branch number 5 and this already proves that every 2 rounds of AES have at least 5 active S-Boxes. The second step suggests that an S-Box with a low differential uniformity (and low linearity) should be used.

Assume that we want to design an ARX-based block cipher with provable security against linear and differential trail-based cryptanalysis. We can use an existing ARX-based block cipher with a small block as a (keyed) S-Box. We then have to use the MEDTP of the small block cipher instead of the differential uniformity. Indeed, for small block sizes, the MEDTP can be obtained for example using the Matsui search algorithm [Mat94]. This evaluation was performed by Biryukov *et al.* in [BVLC16] for the block ciphers SPECK-32 up to SPECK-64. See Table 10.1 for the results on 32-bit block size.

Remark 22. In order to justify the assumption of independent rounds in trails, the authors of [BVLC16] consider SPECKEY, a slightly modified variant of SPECK-32. The only difference is that, in SPECKEY, the round keys are added to the whole state. In this way, the independence assumption is lifted from the block cipher structure to the key schedule. In SPARX, we used SPECKEY in order to have better justified provable security.

r	1	2	3	4	5	6	7	8	9	10
MEDTP	−0	−1	−3	−5	−9	−13	−18	−24	−30	−34
MELTC	−0	−0	−1	−3	−5	−7	−9	−12	−14	−17

TABLE 10.1: MEDTP and MELTC of SPECK-32 / SPECKEY ($\log_2$ scale); r is the number of rounds.

Consider using 1 round of SPECK-32 as the keyed S-Box. Note that it has a differential with probability $1 = 2^{-0}$. Therefore, the bound on MEDTP obtained from the wide-trail argument will be trivial, i.e. MEDTP ≤ 1.

Now consider using 3 rounds of SPECK-32 as the keyed S-Box A. Assume that we design a block cipher E with 128-bit block, i.e. with 4 parallel SPECK-32-based S-Boxes. Assume that the linear layer is a 4×4 MDS matrix over $\mathbb{F}_{2^{32}}$, i.e. it has branching number 5. Then at least 5 S-Boxes are active every two rounds and each S-Box has $\mathsf{MEDTP}(A) = 2^{-3}$. It follows that for the r round block cipher E_r, the wide-trail argument provides bound $\mathsf{MEDTP}(E_r) \leq (2^{-3})^{5r/2}$. In order to get $\mathsf{MEDTP}(E_r) \leq 2^{-128}$, we need $r \geq 128/7.5 \approx 17.07$. Therefore, at least 18 rounds of SPN are needed, i.e. 54 rounds of SPECK-32 repeated four times in parallel. Such a block cipher would be very inefficient.

Using the novel *long-trail* strategy, we show that it is possible to build much more efficient block ciphers with ARX-based S-Boxes and provable security against linear and differential trail-based cryptanalysis.

10.2.2 The Long-Trail Argument

Observe that in the ARX-based block ciphers the MEDTP grows slower at the first few rounds and grows faster afterwards. For example, the MEDTP of the 10 round SPECK-32 is 2^{-34}, which is much less than the MEDTP of the 5 round SPECK-32 squared: $(2^{-9})^2 = 2^{-18}$. The wide-trail strategy does not exploit this fact and uses the worse bound. Indeed, in general, the better bound can not be used, because the 10 rounds of SPECK-32 are not always isolated inside the trail structure. Therefore, each concrete trail structure must be analyzed separately. We call *a long trail* such an isolated chain of (keyed) S-Boxes.

Definition 10.4 (Long Trail). *Consider an SPN-based block cipher and a fixed trail $\boldsymbol{\alpha}$. A* long trail (LT) *is a chain of active S-Boxes in the trail interleaved with key additions, such that no difference comes into the chain from outside (i.e., the linear layers do not mix in differences into the chain).*

Consider a partition of active S-Boxes in the trail into long trails. The multiset of lengths of long trails in any such partition is called a long trail decomposition *of the trail T, denoted* $\mathsf{LT}(\boldsymbol{\alpha})$.

Proposition 10.5 (Long-Trail Bound). *Let $\mathbf{f}$ be round function of an SPN-based block cipher with an S-Box S and let $\boldsymbol{\alpha}$ be a trail over $\mathbf{f}$. Then*

$$\mathsf{EDTP}_{\mathbf{f}}(\boldsymbol{\alpha}) \le \prod_{r^{(m)} \in \mathsf{LT}(\boldsymbol{\alpha})} \left(\mathsf{MEDTP}(S^r)\right)^m,$$

$$\mathsf{ELTC}_{\mathbf{f}}(\boldsymbol{\alpha}) \le \prod_{r^{(m)} \in \mathsf{LT}(\boldsymbol{\alpha})} \left(\mathsf{MELTC}(S^r)\right)^m,$$

where $r^{(m)}$ means that element r repeats m times in the multiset $\mathsf{LT}(\boldsymbol{\alpha})$, and $\mathsf{MEDTP}(S^r)$ (resp. $\mathsf{MELTC}(S^r)$) denote the MEDTP of r rounds of S (resp. MELTC).

Proof. Recall that in the definition of EDTP and ELTC all rounds are considered independent. Therefore, all S-Boxes are independent as well. Hence, $\mathsf{EDTP}_{\mathbf{f}}(\boldsymbol{\alpha})$ is a product of some DDT entry of each S-Box (depending on the trail $\boldsymbol{\alpha}$). The proposition simply replaces a subset of these factors by the upper bound on their product, which does not depend on the exact trail $\boldsymbol{\alpha}$, only on the fact that it is a non-zero trail. The same reasoning applies to the case of linear trails. □

This proposition gives an idea of improving a bound on MEDTP and MELTC of a block cipher. Instead of enumerating all valid *exact* trails, we only need to enumerate all valid *truncated* trails telling whether each S-Box is active or not. For each such trail, we need to obtain a preferably optimal long-trail decomposition, which leads to an upper bound on EDTP or ELTC of all exact trails fitting the current truncated trail. By taking the maximum bound among all truncated trails, we obtain an upper bound on MEDTP and MELTC of the block cipher.

For the sake of completeness, I express the wide-trail bound in the same way to highlight that it is a special case of the long-trail bound. Indeed, the long-trail partition of any trail into chains of length 1 is equivalent to counting the number of active S-Boxes. This, in turn, requires less information about each trail and allows to obtain a simple mathematical argument. On the contrary, the long-trail bound requires algorithmic evaluation.

Proposition 10.6 (Wide-Trail Bound)**.** *Let* $\mathbf{f}$ *be round function of an SPN-based block cipher with an S-Box* S *and let* $\boldsymbol{\alpha}$ *be a trail over* $\mathbf{f}$*. Then*

$$\mathsf{EDTP}_{\mathbf{f}}(\boldsymbol{\alpha}) \le \prod_{r^{(m)} \in \mathsf{LT}(\boldsymbol{\alpha})} (\mathsf{MEDTP}(S))^{rm},$$
$$\mathsf{ELTC}_{\mathbf{f}}(\boldsymbol{\alpha}) \le \prod_{r^{(m)} \in \mathsf{LT}(\boldsymbol{\alpha})} (\mathsf{MELTC}(S))^{rm}.$$

10.2.3 An Algorithm for Long-Trail Decomposition

The most straightforward way to apply the long-trail argument to bound the MEDTP and MELTC of a cipher is as follows:

1. enumerate all possible truncated trails composed of active/inactive S-boxes;
2. find an optimal decomposition of each trail into long trails (LT);
3. bound the probability of each trail using the product of the MEDTP (resp. MELTC) of all active long trails i.e. by applying the Long Trail Argument (see Proposition 10.5);
4. the maximum bound over all trails is the final upper bound.

Note that this approach is feasible only for a small number of rounds, because the number of truncated trails grows exponentially.

In this section, I sketch an algorithm for the only non-trivial step, step (2), i.e. an algorithm for finding an optimal decomposition of a given truncated trail into long trails.

First, note that the trail can be represented as a graph, where nodes are active S-Boxes and an edge corresponds to a possible connection of two S-Boxes in a long trail. Moreover, this graph is a forest. Indeed, an S-Box can't receive two edges from the previous round, because it contradicts a definition of long trail - there must be a single difference coming in. For each tree in the forest, we choose the root to be the S-Box from the earliest round, which is determined uniquely by the same reason. Then, for any node its children may only be in the next round.

The goal then is to cover all nodes with disjoint "vertical" paths, such that the product of the paths' probabilities is minimal. By the path probability we understand the respective long trail's probability. The simplest (and the worst) solution is to choose paths consisting of single nodes. Note that this solution already gives some upper bound and by finding a better decomposition we improve this bound.

I propose an algorithm based on recursive dynamic programming approach. For each node, we recursively solve the sub-problem for the subtree rooted at that node. However, we need to compute some additional information apart from the best decomposition of the subtree. Consider the optimal decomposition of the whole forest into such paths and consider the long trail which goes through the current subtree's root. Clearly, if we fix this long trail, the rest of the subtree becomes completely independent and has to be decomposed optimally. Therefore, from the subtree we need to know only the probability of this decomposition and the length of the long trail's part in the subtree. We don't know the optimal length beforehand, therefore we store the best probabilities for all possible lengths. Another view on this is that we group all possible subtree decompositions by length of the long trail which goes through the subtree root and for each such length we greedily choose the minimum probability. Then, when we obtain such tables for all children of some node, we can easily compute the table for the node itself - we check all possible ways to choose a child of the node and the length of the long trail which goes through the child and we try to join the current node to that long trail. Then the corresponding probability is the product of the best probabilities of the other children with the probability corresponding to the children's long trail and the probability stored in the children's table respectively for that length.

Complexity. The complexity is dominated by computing the table for each node. One of the w children has to be selected for the continuation of the trail, and the size of its child's table is limited by the number of rounds r. Therefore, each node's contribution to the complexity is at most $\mathcal{O}(wr)$. The total complexity of the algorithm then is $O(w^2r^2)$, where w is the number of S-Boxes in parallel, and r is the number of rounds. Note that wr corresponds to the total number of S-Boxes in the cipher.

Despite the reasonable efficiency of the algorithm, the amount of all truncated trails for which the algorithm has to be run adds a large factor to the complexity of the evaluation of a block cipher. In the next section, I will describe an algorithm which completes the whole evaluation in a much more efficient way, under a special condition on the linear layer.

10.2.4 Efficient Algorithm for Special Linear Layers

The most complicated step in the above procedure is finding an optimal decomposition of a given truncated trail into long trails. The difficulty arises from the so-called *branching*: situation in which a long trail may be extended in more than one way. The definition of long trail relies on the fact that there is no linear transformation on a path between two S-Boxes in a long trail. Therefore, branching happens only when some output word of the linear layer receives two or more active input words without modifications.

In order to cut off the branching effect (and thus to make finding the optimal decomposition of a long trail trivial), we can put some additional linear functions that will modify the contribution of some of the input words. Equivalently, when choosing a linear layer we simply do not consider layers which cause branching of long trails. As we will show later, this restriction has many advantages.

To simplify our study of the linear layer, we introduce a matrix representation for it. In an SPN-based block cipher operating on w words, the linear layer may be expressed as a $w \times w$ block matrix. We will denote the zero and the identity sub-matrices by 0 and 1 respectively and an unspecified (arbitrary) sub-matrix by L. This information is sufficient for analyzing the high-level structure of a cipher. Using this notation, the linear layers to which we restrict our analysis have matrices in which each column has at most one element 1.

For the special subset of linear layers outlined above, I present an algorithm for obtaining MEDTP and MELTC bounds, based on a dynamic programming approach. Since there is no branching, any truncated trail consists of disjoint sequences of active S-Boxes. We can treat each such sequence as a long trail to obtain an optimal decomposition. More importantly, because of this simplification, we can avoid enumerating all trails by grouping them in a particular way.

We proceed round by round and maintain a set of best truncated trails up to an equivalence relation, which is defined as follows. For all S-Boxes at the current last round s, we assign a number, which is equal to the length of the long trail that covers this S-Box, or zero if the S-Box is not active. We say that two truncated trails for s steps are equivalent if the tuples consisting of those numbers (lengths of long trails) are the same for both truncated trails. This equivalence captures the possibility to replace some prefix of a trail by an equivalent one without breaking the validity of the trail or its LT decomposition. The total probability, however, can change. The key observation is that from two equivalent trails we can keep only the one with the highest current probability. Indeed, if the optimal truncated trail for all r rounds is an extension of the trail for s rounds with lower probability, we can take the first s rounds from the trail with higher probability without breaking validity and obtain a better trail, which contradicts the assumed optimality.

The pseudo-code for the algorithm is given in Algorithm 10.1. Note that in the case of the MELTC bound, the matrix of the linear layer has to be inverted and transposed. However, instead of inversion, we can build up the trails in the reverse direction: from the ciphertext side to the plaintext side. In this way, it is sufficient to only transpose the linear layer.

Complexity. The complexity of the algorithm can be upper-bounded as follows. The size of the set S_i is upper-bounded by the number of all w-tuples of integers in $[0 \dots i]$, i.e. $(i+1)^w$. Generating extensions of an element $s \in S_i$ requires $w2^w$ operations. Repeating this for r rounds results in complexity $\mathcal{O}(r \cdot w2^w \cdot (r+1)^w)$. In practice, only a small subset of all possible w-tuples is possible. Note that this algorithm implicitly already performs the enumeration of all truncated trails and therefore, this is the complexity of the full evaluation of the MEDTP and MELTC of the block cipher.

10.3 The Linear Layer of SPARX

The linear layer of SPARX had to satisfy the following criteria:

Algorithm 10.1 Finding the best bound on the MEDTP of an SPN cipher (for MELTC the matrix should be transposed).

Input: number of rounds r; $w \times w$ matrix M over $\{0, 1, L\}$, with at most one 1 at each column; non-decreasing bounds on EDTP (or ELTC) of the iterated S-Box $(P[1], \ldots, P[r])$

Output: upper bound on the MEDTP (or MELTC)

```
 1: $S_0 \leftarrow \{0,1\}^w \setminus \{0^w\}$                 ▷ 0 - inactive, 1 - LT of length 1
 2: $pr_0[s] = 1.0$ for all $s \in S_0$, $pr_0[s] = 0.0$ otherwise
 3: for all $i \in [0 \ldots r-1]$ do
 4:     $S_{i+1} \leftarrow \{\}$
 5:     for all $s \in S_i$ do
 6:         for all $(s', p') \in Extensions(s, pr_i[s])$ do
 7:             add $s'$ to set $S_{i+1}$
 8:             $pr_{i+1}[s'] \leftarrow max(pr_{i+1}[s'], p')$
 9: return $max(pr_r[s]$ for $s \in S_r)$

10: function EXTENSIONS(s, p)
11:     $out\_states \leftarrow []$
12:     for all $cancel \in \{false, true\}^w$ do
13:         $s' \leftarrow 0^w, p' \leftarrow p$
14:         for all $o \in [0 \ldots w-1]$ do
15:             $mask \leftarrow$ (if $s_i > 0$ then $M_{o,i}$ else 0 for $i \in [0 \ldots w-1]$)
16:             if $mask$ contains single 1 then
17:                 $i \leftarrow$ index of 1 in $mask$
18:                 $s'[o] \leftarrow s[i] + 1$
19:                 $p' \leftarrow p' + P[s[i]+1] - P[s[i]]$         ▷ Extending an LT
20:             else if $mask$ contains single $L$ then
21:                 $s'[o] \leftarrow 1$                ▷ An LT is broken by the linear layer
22:                 $p' \leftarrow p' + P[1]$
23:             else if $mask$ contains at least two nonzero elements then
24:                 if cancel[o] then
25:                     $s'[o] \leftarrow 0$                        ▷ Differences cancelled
26:                 else
27:                     $s'[o] \leftarrow 1$
28:                     $p' \leftarrow p' + P[1]$                 ▷ Differences not cancelled
29:         if $s' \neq 0^w$ then
30:             append $(s', p')$ to $out\_states$
31:     return $out\_states$
```

1. the diffusion should be slow enough to foster long trails;
2. the diffusion should be fast enough to avoid integral attacks;
3. it should be simple and lightweight.

The first two criteria are in a trade-off with each other. Stronger diffusion means achieving security against structural attacks in less rounds, but fewer long trails and therefore, achieving provable security against linear and differential attacks in more rounds.

For SPARX with 64-bit block, there are only two branches, and the linear Feistel round was the best choice. For 128-bit SPARX instances, the choice was not so clear. We decided to exhaustively check a large class of possible linear layers with reasonable implementation properties and for which we could prove MEDCP and MELCC bounds. Note that, for the specified criteria, we are only interested in the high-level structure of the linear layer, i.e. its $w \times w$ matrix over $\{0, 1, L\}$, as in Section 10.2.4.

The algorithm from Section 10.2.4 requires the matrix to have at most one 1 in each column and in each row (because of the linear case). We strengthened the requirement for the matrix to have *exactly* one 1 in each row and column. This should lead to better implementation properties and foster long trails at the same time.

The matrices we look at correspond to permutations of 4 words with some zeroes possibly replaced by special elements which we denote by L. Though there may be several elements L in the matrix, it is not necessary that all the corresponding small linear functions are equal. The total number of such matrices is $4! \cdot 2^{12} = 98304$.

For any matrix M and for any word permutation matrix P, the matrices M and $P \times M \times P^{-1}$ are equivalent up to reordering the S-Boxes in the whole cipher. Only one representative from each such class is kept. Next, the matrices which do not provide full diffusion are also dropped. In order to check the diffusion, we replaced each L with a random small matrix (e.g. 5×5) and applied the matrix multiple times to inputs with one active word. We assumed full diffusion if the full diffusion was reached before 20 matrix applications. After this filtering step we had only 3282 matrices left.

For all reasonable numbers of steps and rounds in a step, we ran Algorithm 10.1 to obtain bounds on MEDTP and MELTC. We also searched for integral characteristics using the division property in order to both ensure good diffusion and to estimate resilience against this type of attack.

Recall that the S-Boxes in our cipher are actually 32-bit block ciphers, based on Speck. Let r_a denote the number of rounds used. The integral characteristic search does not depend on the number of rounds per step because we analyze only the high-level structure. However, the differential and linear bounds do depend on this value, so we had to make the choice. 2 rounds per step completely contradict our analysis simplification about randomness of the ARX box. Whereas for 5 or more rounds per step we have to take fewer steps and the cipher may become susceptible to structural attacks. Therefore, we considered only 3- or 4- round SPECKEY.

Matrices with many "L" are hard to analyze and to implement. We considered different cases based on the number of words which are copied from the input to the output without change. More copies results in easier and more efficient implementation, easier identification of long trails, but weaker diffusion.

Finally, we selected the best matrices according to one of the following two criteria.

1. Minimizing the differential/linear trail probability. We compute the number of steps when the trail probability bound derived by the algorithm is less than 2^{-128} for differential trails and less than 2^{-64} for linear.

2. Minimizing the number of steps of the integral characteristic found with division property.

The results are given in Table 10.2 and Table 10.3, where $+S$ denotes an additional S-Box layer.

#words copied	optim. for	best int. char.	min. rounds (diff./linear)	matrix
0	diff./linear	4	7/7	[`10L0,010L,L001,0L10`]
	diffusion	2+S	8/8	[`10L0,01L0,LLL1,001L`]
1	diff./linear	4+S	7/7	[`001L,0001,10L0,L10L`]
	diffusion	3+S	7/8	[`001L,0001,100L,L1LL`]
2	diff./linear	7+S	6/6	[`00L1,1000,L100,0010`] (A)
	diffusion	3+S	8/11	[`0010,0001,1LLL,L1LL`]
	tradeoff	4+S	7/7	[`0001,1L0L,0100,0L1L`] (B)
3	diff./linear	9+S	7/7	[`LL01,1000,0100,0010`]
	diffusion	7+S	8/9	[`LLL1,1000,0100,0010`]

TABLE 10.2: The best linear layers for $r_a = 3$.

The results show that heavier matrices (without words copied) lead to better diffusion, as expected, whereas for linear/differential security the matrices with 2 words copied give best results for both $r_a = 3$ and $r_a = 4$. Though heavy matrices can give a good compromise between these two criteria, they are hard to implement, to study and to implement their inverses. Thus, we decided to stick to light matrices.

The most interesting matrices are marked by (A),(B),(C),(D) and the structures of the corresponding layers are depicted in Figure 10.1. For $r_a = 3$ the matrix with the best differential/linear security, (A), yields an integral characteristic covering almost 8 steps. Another interesting matrix, (B), requires 7 steps which corresponds to 21 rounds. For $r_a = 4$, we can achieve differential/linear security in 5 steps (20 rounds) using matrix (C). Notably, this matrix is a Feistel round. Matrix (D) is similar but it adds additional mixing

#words copied	optim. for	best int. char.	min. rounds (diff./linear)	matrix
0	diff./linear	3	5/5	[L010,00L1,1L0L,01L0]
	diffusion	2+S	6/5	[10L0,01L0,LLL1,001L]
1	both	3+S	5/5	[10LL,01L0,LLL1,0010]
2	diff./linear	4+S	5/5	[0010,0001,10LL,01LL] (C)
	diffusion	3+S	5/6	[0010,0001,1LLL,L1LL] (D)
3	both	7+S	5/6	[LLL1,1000,0100,0010]

TABLE 10.3: The best linear layers for $r_a = 4$.

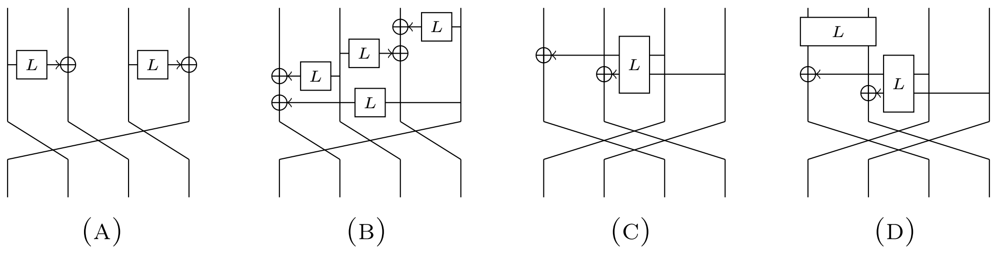


FIGURE 10.1: Possible linear layers.

between the two left branches, which improves diffusion but slightly weakens differential/linear provable security.

A cipher built with $r_a = 4$ and matrix (C) provides a good compromise between differential/linear security, diffusion, strength of the ARX-box, simplicity and easiness/efficiency of implementation. It also generalizes elegantly the linear layer of the 64-bit version of SPARX. We thus settled for this Feistel-like function. For convenience, we decided to use its mirrored version.

Chapter 11

The SPARKLE, ESCH and SCHWAEMM Algorithms

In this chapter, I describe a suite of lightweight cryptographic algorithms. It includes the Sparkle cryptographic permutation family, the Esch hash function family, and the Schwaemm authenticated encryption family. Together

Type	Name	Internal state size (bytes)	Data block size (bytes)	Security level (bits)
Hash function	Esch256	48	16	128
	Esch384	64	16	192
Authenticated encryption	Schwaemm128-128	32	16	120
	Schwaemm256-128	48	32	120
	Schwaemm192-192	48	24	184
	Schwaemm256-256	64	32	248

Table 11.1: Algorithms in the lightweight cryptographic suite.

with my colleagues Christof Beierle, Alex Biryukov, Luan Cardoso dos Santos, Johann Großschädl, Léo Perrin, Vesselin Velichkov and Qingju Wang, we designed and analyzed this suite. It is submitted to the recent NIST call for lightweight algorithms [NIS19]. I describe briefly the specification of the algorithms, and the analysis of the suite that I performed. It includes an evaluation of resistance against several cryptanalysis methods, and also attacks on round-reduced versions of the Schwaemm authenticated encryption family.

11.1 Introduction

With the advent of the Internet of Things (IoT), a myriad of devices are being connected one to another in order to exchange information. This information has to be secured. Symmetric cryptography can ensure that the data those devices share remains confidential, that it is properly authenticated and that it has not been tampered with.

As such objects have little computing power—and even less so that is dedicated to information security—the cost of the algorithms ensuring these properties has to be as low as possible. To answer this need, the NIST has called for the design of authenticated ciphers and hash functions providing a sufficient security level at as small an implementation cost as possible.

We present a suite of algorithms that answer this call. All our algorithms are built using the same core, namely the Sparkle family of permutations. The authenticated ciphers, Schwaemm, provide confidentiality of the plaintext as well as both integrity and authentication for the plaintext and for additional public associated data. The hash functions, Esch, are (second) preimage and collision resistant. Our aim for our algorithms is to use as little CPU cycles as possible to perform their task while retaining strong security guarantees and a small implementation size. This speed will allow devices to use much fewer CPU cycles than what is currently needed to ensure the protection of their data. To give one of many very concrete applications of this gain, the energy demanded by cryptography for a battery-powered micro-controller will be decreased.

The parameters of instances of Esch and Schwaemm are summarized in Table 11.1.

Permutation Sparkle

Sparkle is a family of cryptographic permutations built on the ARX paradigm. Its name comes from the block cipher SPARX [DPU+16], which Sparkle is closely related to, hence its name:

SPARx, but **K**ey **LE**ss.

We provide three versions corresponding to three block sizes, namely Sparkle256, Sparkle384, and Sparkle512. The number of steps used varies with the use case.

Hash Function Esch

A *hash function* takes a message of arbitrary length and outputs a digest with a fixed length. It should provide the cryptographic security notions of *preimage resistance*, *second preimage resistance* and *collision resistance*. The main instance of Esch is Esch256 which produces a 256-bit digest, offering a security level of 128 bits with regard to the above mentioned security goals. It is based on the permutation family Sparkle384. We also provide the member Esch384 based on the permutation family Sparkle512, which produces a 384-bit digest and offers a security level of 192 bits.

The name Esch stands for

Efficient, **S**ponge-based, and **C**heap **H**ashing.

Authenticated Cipher Schwaemm

A scheme for *authenticated encryption with associated data (AEAD)* takes a key and a nonce of fixed length, as well as a message and associated data of arbitrary size. The encryption procedure outputs a ciphertext of the message as well as a fixed-size authentication tag. The decryption procedure takes the key, nonce, associated data and the ciphertext and tag as input and outputs the decrypted message if the tag is valid, otherwise a symbolic error $\perp$. An AEAD scheme should fulfill the security notions of *confidentiality* and *integrity*. Users are expected to *not* reuse nonces for processing messages in a fixed-key instance.

The main instance of Schwaemm is Schwaemm256-128 which takes a 256-bit nonce, a 128-bit key and outputs a 128-bit authentication tag. It achieves a security level of 120 bits with regard to confidentiality and integrity. We further provide three other instances, i.e., Schwaemm128-128, Schwaemm192-192, and Schwaemm256-256 which differ in the length of key, nonce and tag and in the achieved security level.

The name Schwaemm stands for

Sponge-based **C**ipher for
Hardened but **W**eightless **A**uthenticated **E**ncryption
on **M**any **M**icrocontrollers

Outline

This chapter is structured as follows. First, in Section 11.2, I briefly describe the specification of SPARKLE,ESCH and SCHWAEMM families. In the following sections, I describe the security analysis that I performed on this suite.

In Section 11.3, I describe attempts to linearize the S-boxes used in SPARKLE, which we call ARX-boxes, by finding all inputs that inflict no carries during the ARX computations. The problem requires a solution of a system of quadratic equations. I describe a simple heuristics for a guess-and-determine algorithm that allows to solve the problems in a reasonable time. The results suggest that ARX-boxes are resistant against such linearization.

In Section 11.4, I describe a truncated differential analysis of SPARKLE. I propose a generic method for truncated trail analysis based on the binary matrix representation of the linear layer. The results show that SPARKLE has a strong resistance against structural truncated differential trails.

In Section 11.5, I use the division property technique to find integral characteristics of SPARKLE. First, the best characteristics of maximum dimension are found using MILP-aided bit-based division property. Then, I optimize them and prove by a pen-and-paper argument and the classical division property.

Finally, in Section 11.6, I describe several attacks on reduced-round variants of SCHWAEMM. They are based on a technique that I call birthday-differential attacks. It is a variant of known-plaintext attack where particular differences can be found from a relatively small pool of data by the birthday paradox.

11.2 Specification of SPARKLE, ESCH and SCHWAEMM

In this section, I describe in brief the specification of the cryptographic primitives that we designed. For the up-to-date specification and information about the suite I refer to the website [BBdS+19a].

The empty bitstring is denoted ϵ. The algorithms assume the byte order to be little-endian. "+" denotes the addition modulo 2^{32}.

11.2.1 The SPARKLE Permutations

Our schemes for authenticated encryption and hashing employ the permutation family SPARKLE which we specify in the following. In particular, the SPARKLE family consists of the permutations SPARKLE256$_{n_s}$, SPARKLE384$_{n_s}$ and SPARKLE512$_{n_s}$ with block sizes of 256, 384, and 512 bit, respectively. The parameter n_s refers to the number of *steps* and a permutation can be defined for any $n_s \in \mathbb{Z}_+$. The permutations are built using the following main components:

- An *ARX-box A*, a 64-bit block cipher with a 32-bit key

  $$A\colon (\mathbb{F}_2^{32} \times \mathbb{F}_2^{32}) \times \mathbb{F}_2^{32} \to (\mathbb{F}_2^{32} \times \mathbb{F}_2^{32}), ((x, y), c) \mapsto (u, v).$$

  We define A_c to be the permutation $(x, y) \mapsto A(x, y, c)$ from $\mathbb{F}_2^{32} \times \mathbb{F}_2^{32}$ to $\mathbb{F}_2^{32} \times \mathbb{F}_2^{32}$.

- A linear *diffusion layer* $\mathcal{L}_{n_b}\colon (\mathbb{F}_2^{64})^{n_b} \rightarrow (\mathbb{F}_2^{64})^{n_b}$, where n_b denotes the number of 64-bit branches, i.e., the block size divided by 64. It is necessary that n_b is even.

The high-level structure of the permutation is given in Algorithm 11.1. It is a classical Substitution-Permutation Network (SPN) construction except that functions playing the role of the S-boxes are different in each branch. More specifically, each member of the permutation family iterates a parallel application of the ARX-box A under different, branch-dependent, constants $c_i \in \mathbb{F}_2^{32}$. This small 64-bit block cipher is specified in Section 11.2.1. It is followed by an application of $\mathcal{L}_{n_b}$, a linear permutation operating on all branches; it is specified in Section 11.2.1. We call such a parallel application of the ARX-boxes followed by the linear layer a *step*. The high-level structure of a step is represented in Figure 11.1. Before each step, a sparse step-dependent constant is XORed to the cipher's state (more precisely, to y_0 and y_1).

In what follows, we rely on the following definition given below to simplify our descriptions.

Definition 11.1 (Left/Right branches)**.** *We call* left branches *those that correspond to the state inputs* $(x_0, y_0), (x_1, y_1), \dots, (x_{n_b/2-1}, y_{n_b/2-1})$*, and we call* right branches *those corresponding to* $(x_{n_b/2}, y_{n_b/2}), \dots, (x_{n_b-2}, y_{n_b-2}), (x_{n_b-1}, y_{n_b-1})$.

Algorithm 11.1 The Permutation SPARKLE$_{128n_b}$
In/Out: $\big((x_0, y_0), \dots, (x_{n_b-1}, y_{n_b-1})\big), x_i \in \mathbb{F}_2^{32}, y_i \in \mathbb{F}_2^{32}$

```
$(c_0, c_1) \leftarrow$ (0xB7E15162,0xBF715880)
$(c_2, c_3) \leftarrow$ (0x38B4DA56,0x324E7738)
$(c_4, c_5) \leftarrow$ (0xBB1185EB,0x4F7C7B57)
$(c_6, c_7) \leftarrow$ (0xCFBFA1C8,0xC2B3293D)
for all $s \in [0 \dots n_s - 1]$ do
    $y_0 \leftarrow y_0 \oplus c_{(s \bmod 8)}$
    $y_1 \leftarrow y_1 \oplus (s \bmod 2^{32})$
    for all $i \in [0 \dots n_b - 1]$ do
        $(x_i, y_i) \leftarrow A_{c_i}(x_i, y_i)$
    $\big((x_0, y_0), \dots, (x_{n_b-1}, y_{n_b-1})\big) \leftarrow \mathcal{L}_{n_b}\big((x_0, y_0), \dots, (x_{n_b-1}, y_{n_b-1})\big)$
return $\big((x_0, y_0), \dots, (x_{n_b-1}, y_{n_b-1})\big)$
```

Specific Instances. The SPARKLE permutations are defined for 4,6 and 8 branches and for any number of steps. In our suite we use two versions of the permutations which *differ only by the number of steps* used. More precisely, we use a *slim* and a *big* instance of SPARKLE. The slim and big versions of all SPARKLE instances are given in Table 11.2.

The ARX-Box

The ARX-box A is a 64-bit block cipher. It is specified in Algorithm 11.2 and depicted in Figure 11.2. It can be understood as a four-round iterated block

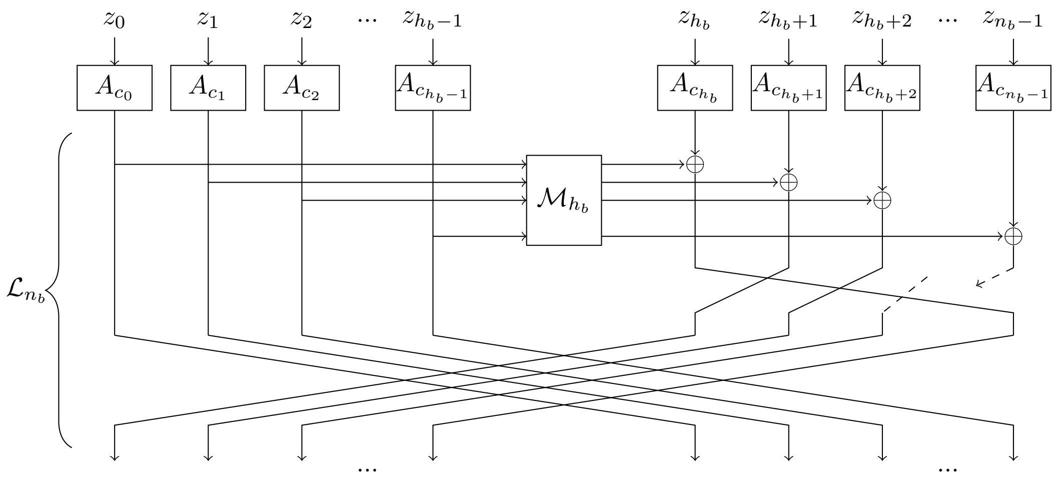


FIGURE 11.1: The overall structure of a step of SPARKLE. z_i denotes the 64-bit input (x_i, y_i) to the corresponding ARX-box.

Name	n	# steps slim	# steps big
SPARKLE256	256	7	10
SPARKLE384	384	7	11
SPARKLE512	512	8	12

TABLE 11.2: The different versions of each SPARKLE instance.

cipher for which the rounds differ in the rotation amounts. After each round, the 32-bit constant ("the key") is XORed to the left word. As the ARX-box has a simple Feistel structure, the computation of the inverse is straightforward.

Its purpose is to provide non-linearity to the whole permutation and to ensure a quick diffusion within each branch — the diffusion between the branches being ensured by the linear layer (Section 11.2.1). Its round constants ensure that the computations in each branch are independent from one another to break the symmetry of the permutation structure we chose. As the rounds themselves are different, we do not rely on the round constant to provide independence between them.

The Diffusion Layer

The diffusion layer has a structure which draws heavily from the one used in SPARX-128 [DPU+16]. We denote it $\mathcal{L}_{n_b}$. It is a Feistel round with a linear Feistel function $\mathcal{M}_{h_b}$ which permutes $\left(\mathbb{F}_2^{64}\right)^{h_b}$, where $h_b = \frac{n_b}{2}$. Formally, $\mathcal{M}_{h_b}$ is defined as follows.

Definition 11.2. *Let $w \in \mathbb{Z}_+$. We denote $\mathcal{M}_w$ the permutation of $(\mathbb{F}_2^{32})^w$ such that*

$$\mathcal{M}_w\big((x_0, y_0), \dots, (x_{w-1}, y_{w-1})\big) = \big((u_0, v_0), \dots, (u_{w-1}, v_{w-1})\big)$$

Algorithm 11.2 The ARX-box A_c

Input/Output: $(x, y) \in \mathbb{F}_2^{32} \times \mathbb{F}_2^{32}$

```
  $x \leftarrow x + (y \ggg 31)$
  $y \leftarrow y \oplus (x \ggg 24)$
  $x \leftarrow x \oplus c$
  $x \leftarrow x + (y \ggg 17)$
  $y \leftarrow y \oplus (x \ggg 17)$
  $x \leftarrow x \oplus c$
  $x \leftarrow x + (y \ggg 0)$
  $y \leftarrow y \oplus (x \ggg 31)$
  $x \leftarrow x \oplus c$
  $x \leftarrow x + (y \ggg 24)$
  $y \leftarrow y \oplus (x \ggg 16)$
  $x \leftarrow x \oplus c$
  **return** $(x, y)$
```

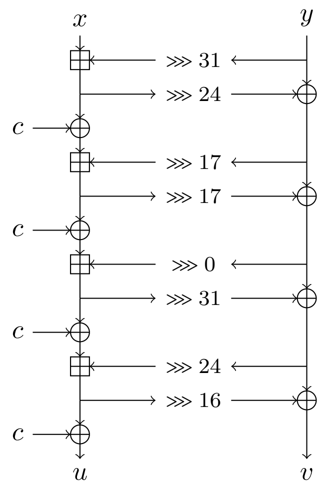


FIGURE 11.2: The ARX-box structure A_c for a 32-bit constant c.

where the branches (u_i, v_i) are obtained via the following equations

$$\begin{aligned} t_y &\leftarrow \bigoplus_{i=0}^{w-1} y_i \ , \ t_x \leftarrow \bigoplus_{i=0}^{w-1} x_i \ , \\ u_i &\leftarrow x_i \oplus \ell(t_y), \ \forall i \in \{0, ..., w-1\} \ , \\ v_i &\leftarrow y_i \oplus \ell(t_x), \ \forall i \in \{0, ..., w-1\} \ , \end{aligned} \tag{11.1}$$

where the indices are understood modulo w, and where $\ell : \mathbb{F}_2^{32} \to \mathbb{F}_2^{32}$ is a permutation defined by

$$\ell(x) = (x \lll 16) \oplus (x \wedge \texttt{0000FFFF}).$$

Note in particular that, if y and z are in $\mathbb{F}_2^{16}$ so that $(y, z) \in \mathbb{F}_2^{32}$, then

$$\ell(y, z) = (z, y \oplus z).$$

The diffusion layer $\mathcal{L}_{n_b}$ then applies the corresponding Feistel function $\mathcal{M}_{h_b}$ and swaps the left branches with the right branches. However, before the branches are swapped, we rotate the branches on the right side by 1 branch to the left. This process is pictured in Figure 11.1. As an example, an algorithm describing the linear diffusion layer of SPARKLE384 is given in Algorithm 11.3

Algorithm 11.3 The Linear Layer $\mathcal{L}_6$
Input/Output: $\big((x_0, y_0), \ldots, (x_5, y_5)\big) \in (\mathbb{F}_2^{32} \times \mathbb{F}_2^{32})^6$

```
                                                        ▷ Feistel round
  $(t_x, t_y) \leftarrow \big(x_0 \oplus x_1 \oplus x_2,\ y_0 \oplus y_1 \oplus y_2\big)$
  $(t_x, t_y) \leftarrow \big((t_x \oplus (t_x \ll 16)) \lll 16,\ (t_y \oplus (t_y \ll 16)) \lll 16\big)$
  $(y_3, y_4, y_5) \leftarrow (y_3 \oplus y_0 \oplus t_x,\ y_4 \oplus y_1 \oplus t_x,\ y_5 \oplus y_2 \oplus t_x)$
  $(x_3, x_4, x_5) \leftarrow (x_3 \oplus x_0 \oplus t_y,\ x_4 \oplus x_1 \oplus t_y,\ x_5 \oplus x_2 \oplus t_y)$
                                                        ▷ Branch permutation
  $(x_0, x_1, x_2, x_3, x_4, x_5) \leftarrow (x_4, x_5, x_3, x_0, x_1, x_2)$
  $(y_0, y_1, y_2, y_3, y_4, y_5) \leftarrow (y_4, y_5, y_3, y_0, y_1, y_2)$
  return $\big((x_0, y_0), \ldots, (x_5, y_5)\big)$
```

11.2.2 The ESCH Hash Function

We propose two instances for hashing, i.e., ESCH256 and ESCH384, which allow to process messages $M \in \mathbb{F}_2^*$ of arbitrary length[1] and output a digest D of bitlengths 256, and 384, respectively. They employ the well-known sponge construction, which is instantiated with SPARKLE permutations and parameterized by the rate r and the capacity c. The slim version is used during both absorption and squeezing. The big version is used in between the two phases.

In both ESCH256 and ESCH384, the rate r is fixed to 128. This means that the message M has to be padded such that its length in bit becomes a multiple of

[1] More rigorously, all bitlengths under a given (very large) threshold are supported.

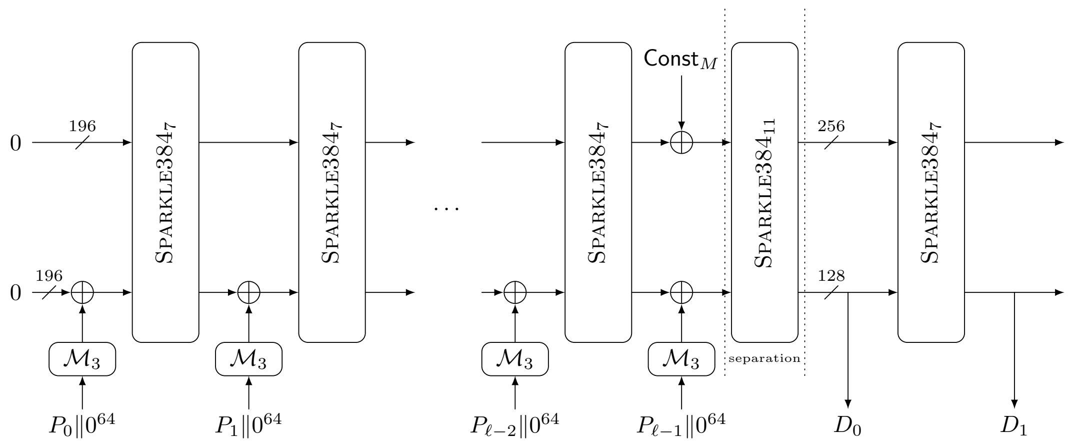


FIGURE 11.3: The Hash Function ESCH256 with rate $r = 128$ and capacity $c = 256$.

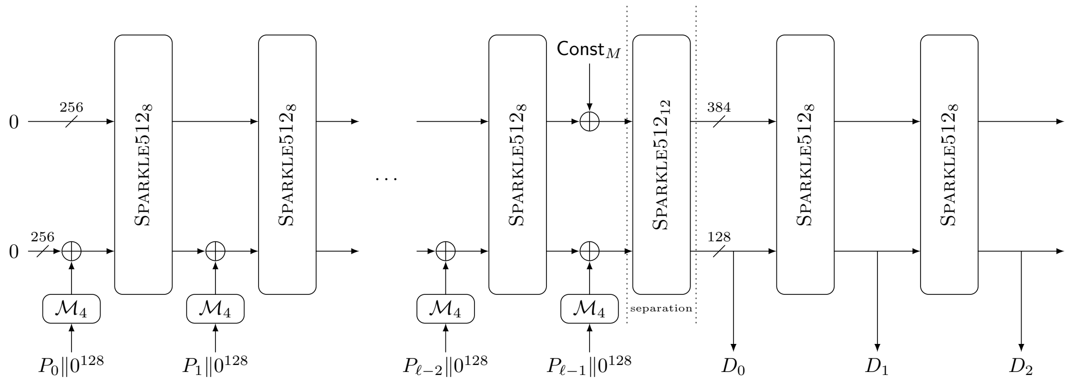


FIGURE 11.4: The Hash Function ESCH384 with rate $r = 128$ and capacity $c = 384$.

128. For this, we use the simple padding rule that appends 10^*. The algorithms are depicted in Figure 11.3 and Figure 11.4, respectively. Note that the 128 bits of message blocks are injected *indirectly.* They are first padded with zeros and transformed via $\mathcal{M}_3$ in ESCH256, respectively, $\mathcal{M}_4$ in ESCH384, and the resulting image is XORed to the *leftmost* branches of the state. We stress that this tweak can still be expressed in the regular sponge mode. Instead of injecting the messages through $\mathcal{M}_{h_b}$, one can use an equivalent representation in which the message is injected as usual and the permutation is defined by prepending $\mathcal{M}_{h_b}$ and appending $\mathcal{M}_{h_b}^{-1}$ to SPARKLE$_{n_b}$.

A message with a length that is a multiple of r is not padded. To prevent trivial collisions, we borrow the technique introduced in [Hir16] and add $\mathsf{Const}_\mathsf{M}$ in the capacity, where $\mathsf{Const}_\mathsf{M}$ is different depending on whether the message was padded or not.

	n	r	c	$\|K\|$	$\|N\|$	$\|T\|$	security	data limit
SCHWAEMM128-128	256	128	128	128	128	128	120	2^{68}
SCHWAEMM256-128	384	256	128	128	256	128	120	2^{68}
SCHWAEMM192-192	384	192	192	192	192	192	184	2^{68}
SCHWAEMM256-256	512	256	256	256	256	256	248	2^{133}

TABLE 11.3: The instances we provide for authenticated encryption together with their (joint) security level in bit with regard to confidentiality and integrity and the limitation in the data (in bytes) to be processed.

11.2.3 The SCHWAEMM Authenticated Ciphers

We propose four instances for authenticated encryption with associated data:

SCHWAEMM128-128, SCHWAEMM192-192, SCHWAEMM256-256,
and SCHWAEMM256-128

which, for a given key K and nonce N allow to process associated data A and messages M of arbitrary length (up to a certain threshold) and output a ciphertext C with $|C| = |M|$ and an authentication tag T. For given (K, N, A, C, T), the decryption procedure returns the decryption M of C if the tag T is valid, otherwise it returns the error symbol $\perp$. All instances use (a slight variation of) the BEETLE mode of operation presented in [CDNY18], which is based on the well-known duplexed sponge construction. The difference between the instances is the version of the underlying SPARKLE permutation (and thus the rate and capacity is different) and the size of the authentication tag. As a naming convention, we used SCHWAEMMr-c, where r refers to the size of the rate and c to the size of the capacity in bits. Similar as for hashing, we use the big version of SPARKLE for initialization, separation between processing of associated data and secret message, and finalization, and the slim version of SPARKLE for updating the intermediate state. Table 11.3 gives an overview of the parameters of the SCHWAEMM instances. The data limits correspond to 2^{64} blocks of r bits rounded up to the closest power of two, except for the high security SCHWAEMM256-256 for which it is $r \times 2^{128}$ bits.

The main difference between the BEETLE mode and duplexed sponge modes is the usage of a combined feedback ρ to differentiate the ciphertext blocks and the rate part of the states. This combined feedback is created by applying the function $\mathsf{FeistelSwap}$ to the rate part of the state, which is computed as

$$\mathsf{FeistelSwap}(S) = S_2 \| (S_2 \oplus S_1) \ ,$$

where $S \in \mathbb{F}_2^r$ and $S_1 \| S_2 = S$ with $|S_1| = |S_2| = \frac{r}{2}$. The feedback function $\rho \colon (\mathbb{F}_2^r \times \mathbb{F}_2^r) \to (\mathbb{F}_2^r \times \mathbb{F}_2^r)$ is defined as $\rho(S, D) = (\rho_1(S, D), \rho_2(S, D))$, where

$$\rho_1 \colon (S, D) \mapsto \mathsf{FeistelSwap}(S) \oplus D, \quad \rho_2 \colon (S, D) \mapsto S \oplus D \ .$$

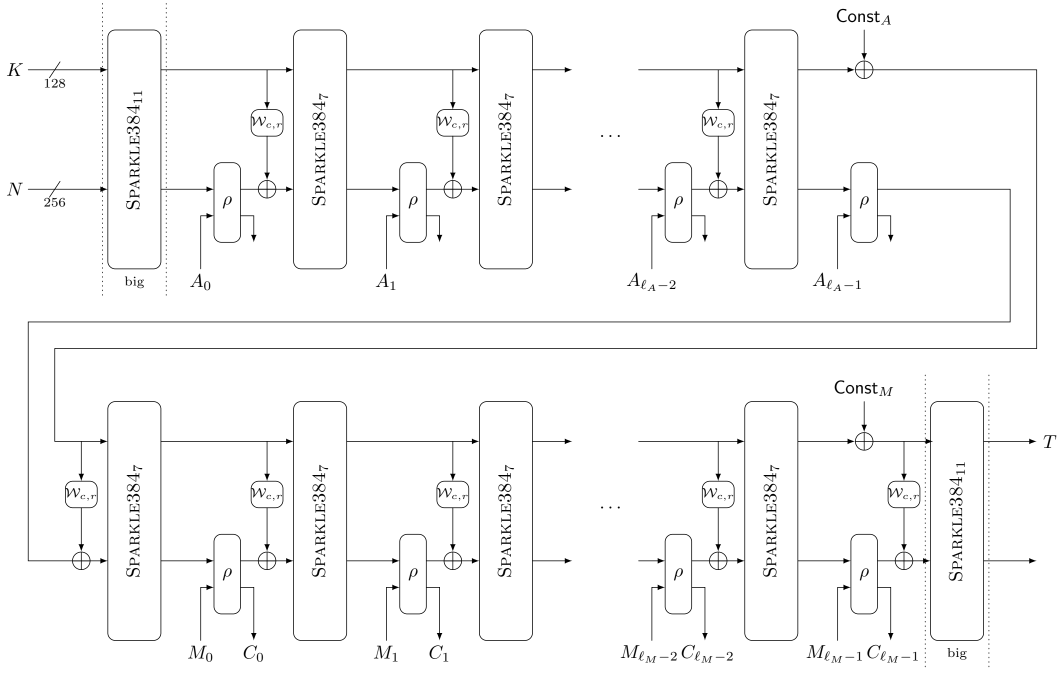


FIGURE 11.5: The Authenticated Encryption Algorithm SCHWAEMM192-192 with rate $r = 192$ and capacity $c = 192$.

For decryption, we have to use the inverse feedback function $\rho' \colon (\mathbb{F}_2^r \times \mathbb{F}_2^r) \to (\mathbb{F}_2^r \times \mathbb{F}_2^r)$ defined as $\rho'(S, D) = (\rho'_1(S, D), \rho'_2(S, D))$, where

$$\rho'_1 \colon (S, D) \mapsto \mathsf{FeistelSwap}(S) \oplus S \oplus D, \quad \rho'_2 \colon (S, D) \mapsto S \oplus D\ .$$

After each application of ρ and the additions of the domain separation constants, i.e., before each call to the SPARKLE permutation except the one for initialization, we prepend a *rate whitening* layer which XORs the value of $\mathcal{W}_{c,r}(S_R)$ to the rate, where S_R denotes the internal state corresponding to the capacity part. For the SCHWAEMM instances with $r = c$, we define $\mathcal{W}_{c,r} \colon \mathbb{F}_2^c \to \mathbb{F}_2^r$ as the identity (i.e., we just XOR the capacity part to the rate part). For SCHWAEMM256-128, we define $\mathcal{W}_{128,256}(x, y) = (x, y, x, y)$, where $x, y \in \mathbb{F}_2^{64}$. Note that this tweak can still be described in the BEETLE framework as the prepended rate whitening can be considered to be part of the definition of the underlying permutation.

Figure 11.5 depicts the mode for our primary member SCHWAEMM192-192.

11.3 Linearization of ARX-boxes

In recent attacks against Keccak instances [QSLG17, SLG17] the S-Box linearization technique is used. The idea is to find a subset of inputs (often an affine subspace), such that the S-Box acts linearly on this set. I attempted to linearize the ARX-boxes by finding all inputs, for which all four modular additions inflict no carry bits and thus are equivalent to XOR. For the addition of two random independent 32-bit words, the probability of having all carry bits equal to zero is equal to $(3/4)^{31}$. Indeed, for each bit position, if no carry comes in, then the outgoing carry will occur only if both input bits are equal to 1. Furthermore, the carry bit from the most significant bits is ignored. Assuming independence of the additions in an ARX-box, $2^{64}(3/4)^{124} \approx 2^{12.5}$ inputs are expected to satisfy the linearization trail. In order to find all such inputs, a quadratic equation system has to be solved.

11.3.1 Quadratic Equation System

The following proposition formalizes the linearization of addition modulo 2^n.

Proposition 11.3. *For any $x, y \in \mathbb{F}_2^n$, the addition $(x + y) \mod 2^n$ is equal to the XOR $x \oplus y$ if and only if $x_i y_i = 0$ for all $i \in [2 \ldots n]$. In particular,*

$$\Pr_{x,y \in \mathbb{F}_2^n}[(x+y) \mod 2^n = x \oplus y] = \left(\frac{3}{4}\right)^{n-1}.$$

Proof. By induction for i from n to 1, $x_i \wedge y_i = 0$ implies that there are no carries, except maybe the carry after the addition of x_1 and y_1. For the other direction, observe that a single carry for any $i \geq 2$ would necessarily modify at least one bit. Furthermore, all positions are independent, since the carries are fixed, and the probability $3/4$ is simply multiplied. □

In order to find all inputs satisfying the linearization in all four rounds, we have to solve a system of quadratic equations. Indeed, for the first round with inputs $x, y \in \mathbb{F}_2^{32}$, we obtain 31 quadratic bit equations from Proposition 11.3. Since the conditions ensures that the output of the first round is linear, similar quadratic equations are obtained for the second round, except that x and y are replaced by the corresponding linear functions. In total we obtain 124 quadratic equations of the form

$$l(x, y) \cdot r(x, y) = 0,$$

where $l, r\colon (\mathbb{F}_2^{32})^2 \to \mathbb{F}_2$ are affine.

The linear functions $l(x, y), r(x, y)$ can be simplified by changing the basis of the equation system. Let $x, y \in \mathbb{F}_2^{32}$ denote the branches of the state before the second constant addition (see Figure 11.6). In order to perform computations from x, y, the first two modular additions must be replaced by modular *subtractions*. The linearization of subtraction is formalized by the following proposition.

Proposition 11.4. *For any $x, y \in \mathbb{F}_2^n$, the subtraction $(x - y) \mod 2^n$ is equal to the XOR $x \oplus y$ if and only if $(x_i \oplus 1)y_i = 0$ for all $i \in [2 \ldots n]$.*

Proof. Observe that

$$x - y = 2^n - 1 - (2^n - 1 - x + y) = \neg((\neg x) + y).$$

As we want to ensure $x - y = x \oplus y$, we get

$$(\neg x) + y = \neg(x \oplus y) = (\neg x) \oplus y.$$

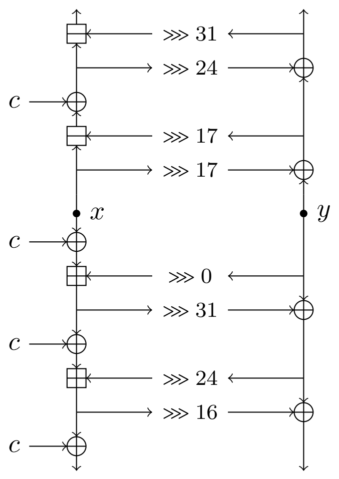


FIGURE 11.6: The variables x, y in the middle of the ARX-box A_c.

Therefore, the subtraction constraints are equivalent to the addition constraints for $(\neg x) + y$. The result follows from Proposition 11.3. $\square$

The two modular subtractions and the two modular additions provide the following 124 equations for an ARX-box A_c.

Proposition 11.5. *Consider the ARX-box A_c with a constant $c \in \mathbb{F}_2^{32}$. All four its modular additions are equivalent to XORs if and only if the intermediate values $x, y \in \mathbb{F}_2^{32}$ as shown in Figure 11.6 satisfy the following 124 equations:*

$$\begin{aligned}(1 \oplus c_{2+i} \oplus x_{2+i} \oplus x_{32+i} \oplus y_{17+i}) \cdot (c_{11+i} \oplus x_{11+i} \oplus x_{18+i} \oplus x_{9+i} \oplus y_{26+i} \oplus y_{3+i}) = 0,\\ (1 \oplus x_{2+i}) \cdot (x_{32+i} \oplus y_{17+i}) = 0,\\ (c_{2+i} \oplus x_{2+i}) \cdot (y_{2+i}) = 0,\\ (x_{2+i} \oplus y_{2+i}) \cdot (c_{11+i} \oplus x_{11+i} \oplus y_{10+i} \oplus y_{11+i}) = 0,\end{aligned}$$

where $i \in [0 \dots 31]$.

11.3.2 Guess and Determine Algorithm

Hardness of the Equation Type

All the quadratic equations are of the form

$$l(x, y) \cdot r(x, y) = 0,$$

which is equivalent to any of the two implications

$$l(x, y) \Rightarrow r(x, y) \oplus 1, r(x, y) \Rightarrow l(x, y) \oplus 1.$$

If all l, r were single variables, then the system would be equivalent to a 2-SAT problem and would be efficiently solvable. However, due to l, r being linear functions of the secret variables, it is NP-complete.

Proposition 11.6. *The problem of finding $x \in \mathbb{F}_2^n$ such that $l_i(x) \cdot r_i(x) = 0$ for all given affine functions $l_i, r_i \colon \mathbb{F}_2^n \to \mathbb{F}_2$ is NP-complete.*

Proof. Let us reduce a general 3-SAT instance to this problem. It is sufficient to show en encoding of a 3-SAT clause $(a \vee b \vee c)$. Let us introduce a new variable v_{bc} such that it should be equal to $b \vee c$. This can be ensured by two implications:

$$(b \Rightarrow v_{bc}) \wedge (v_{bc} \oplus c \Rightarrow b).$$

Then the following encodings are equivalent:

$$\begin{aligned}(a \vee b \vee c) &\Leftrightarrow (a \vee v_{bc}) \wedge (b \Rightarrow v_{bc}) \wedge (v_{bc} \oplus c \Rightarrow b) \Leftrightarrow \\ &\Leftrightarrow (\neg a \cdot \neg v_{bc} = 0) \wedge (b \cdot \neg v_{bc} = 0) \wedge ((v_{bc} \oplus c) \cdot \neg b = 0)).\end{aligned}$$

□

Due to sparsity of the affine maps l, r in the linearization problem, the guess-and-determine approach may work well enough.

Generating More Equations

In order to improve the efficiency, we first generate more equations of the same form by combining the equations in the following way. For each affine map l_i (or r_i) used in the equations, we attempt to find an affine function α and a subset of equations $\{l_j(x,y) \cdot r_j(x,y) = 0\}_{j \in J}$ such that the function

$$l'_i \coloneqq \alpha \cdot l_i \oplus \bigoplus_{j \in J} l_j \cdot r_j$$

has algebraic degree 1. This can be done using a basic linear algebra. We then obtain a new equation

$$l'_i(x,y) \cdot r(x,y) = \alpha(x,y) \cdot l(x,y) \cdot r(x,y) = 0.$$

Example 5. Let $c_2 = 0$ and consider the equations

$$x_2 \cdot y_2 = 0, \quad (1 \oplus x_2) \cdot (x_{32} \oplus y_{17}) = 0.$$

Let $\alpha = x_{32} \oplus y_{17}$, and let the subset of equations consist of the second equation. Then

$$\alpha \cdot x_2 \oplus (1 \oplus x_2) \cdot (x_{32} \oplus y_{17}) = (x_{32} \oplus y_{17}) \cdot x_2 \oplus (1 \oplus x_2) \cdot (x_{32} \oplus y_{17}) = x_{32} \oplus y_{17}.$$

We obtain a new equation

$$(x_{32} \oplus y_{17}) \cdot y_2 = 0.$$

The number of generated equations depends on the chosen round constant c and in our case varies from 36 to 75 for the constants used in SPARKLE, and 186 generated equations for the zero constant. The exact numbers are reported in Table 11.4.

Guess-and-Determine Approach

Due to the implicational nature of equations, guess-and-determine method may work reasonably well. Our approach is based on two ideas. The first idea is to choose an order in which variables will be guessed by using a simple heuristic. The second idea is to verify the consistency of each current guess by checking the consistency of all linear equations that appear once some l_i or r_i becomes equal to one in the equation $l_i \cdot r_i = 0$. Finally, when only linear equations are left, the solutions are easily enumerated by basic linear algebra methods. Using these simple ideas, I managed to exhaustively find all solutions to the equation systems for several ARX-boxes. The algorithm implementation runs in an hour on a modern laptop, for a single ARX-box.

First, I describe the guessing order heuristic. Consider the equations $l_i \cdot r_i = 0$ where both l_i, r_i are non-constant. Let t_1 denote the minimum number of variables involved in such a function l_i or r_i. Let t_2 denote the second minimum such number, t_3 denote the third minimum such number, etc. Then, each variable x_j is assigned a vector $C_j = (c_1, c_2, c_3, \ldots)$ where c_k is the number of considered functions l_i or r_i involving t_k variables including x_j. Then the variable x_j with the largest such vector is selected, where the comparison is done lexicographically. The selected variable is added to the guess order, eliminated from equations and the process repeats until all variables are eliminated.

Example 6. Assume that we consider equations

$$(x_1)\cdot(x_2) = 0, (x_1\oplus 1)\cdot(x_1\oplus x_2) = 0, (x_2)\cdot(x_2\oplus x_3) = 0, (x_1\oplus x_3\oplus x_4\oplus x_5)\cdot(x_3\oplus 1) = 0.$$

Initially, we get

$$C_1 = (2,1,1), C_2 = (2,2,0), C_3 = (1,1,1), C_4 = (0,0,1), C_5 = (0,0,1).$$

We select x_2 as the first variable to guess. After elimination, we obtain that the first and the third equations are removed, because x_2 becomes constant. Next, we get

$$C_1 = (2,0,1), C_3 = (1,0,1), C_4 = (0,0,1), C_5 = (0,0,1).$$

We select x_1 as the second variable to guess. After the elimination, only the last equation remains, and x_3 is selected as the last variable to guess. We do not need to guess x_4, x_5 as the system at this point will be linear.

The full approach is described in Algorithm 11.4.

11.3.3 Generalization to Arbitrary Carry Patterns

Note that an ARX-box is linearized not only in the case when all carries are zero. In fact, it is sufficient that all carries are fixed. For an isolated modular addition, the fraction of inputs leading to a particular carry mask depends on the number of adjacent bits in the carry mask that are equal.

Algorithm 11.4 Guess-and-Determine algorithm for ARX-box Linearization.

```
Input: a system $E$ of equations $\{l_i(x) \cdot r_i(x) = 0\}$,
       where $l_i, r_i \colon \mathbb{F}_2^n \to \mathbb{F}_2$ are affine;
Output: all solutions $x \in \mathbb{F}_2^n$ to the system.

    ▷ generate more equations:
 1: for all $(l_i, r_i) \in E$, $(r_i, l_i) \in E$ do
 2:     $L \leftarrow \mathrm{span}(x_1 \cdot l_i, \dots, x_n \cdot l_i, l_i)$
 3:     $E \leftarrow E \cup \{l' \cdot r_i = 0 \mid l' \in L/E, \deg l' \leq 1\}$        ▷ Linear algebra

    ▷ compute a guessing order heuristically:
 4: $order \leftarrow []$
 5: $E' \leftarrow E$
 6: while $E'$ is not linear do
 7:     $C_j \leftarrow \left( |\{(l_i, r_i) \in E' \text{ or } (r_i, l_i) \in E' \mid \deg l_i = \deg r_i = 2, |l_i| = t, x_j \in l_i\}| \right)_{t \in \mathbb{Z}_+}$,
               where $|l_i|$ is the number of variables in $l_i$,
                     $x_j \in l_i$ means that $x_j$ is involved in $l_i$
 8:     $j \leftarrow \arg\max_{j \in [1 \dots n], j \notin order} C_j$
 9:     $E' \leftarrow E'\big|_{x_j = 0}$        ▷ elimination of $x_j$
10:     append $j$ to $order$

    ▷ enumerate solutions:
11: function GuessAndDetermine($guessed, E$)
12:     $i \leftarrow order_{|guessed|}$
13:     for all $v \in \{0, 1\}$ do
14:         $E' \leftarrow E\big|_{x_i = v}$
15:         $L \leftarrow$ all linear equations from $E'$
16:         if $L = E'$ then
17:             yield solutions based on $guessed$ and $L$
18:         else if $L$ is consistent then
19:             $GuessAndDetermine(guessed \cup \{x_i = v\}, E')$
20: $GuessAndDetermine(\{\}, E)$
```

Proposition 11.7. *Let $e \in \mathbb{F}_2^n$. For any $x, y \in \mathbb{F}_2^n$, the addition $(x+y) \mod 2^n$ is equal to the* XOR *$x \oplus y \oplus e$ if and only if $e_n = 0$ and for all $i \in [1 \dots n-1]$,*

$$e_i \oplus e_{i+1} = (x_{i+1} \oplus e_{i+1}) \cdot (y_{i+1} \oplus e_{i+1}).$$

In particular, for any $e \in \mathbb{F}_2^n$ with $e_n = 0$,

$$\Pr_{x,y \in \mathbb{F}_2^n}[(x+y) \mod 2^n = x \oplus y \oplus e] = \frac{3^m}{4^{n-1}},$$

where $m = \sum_{i=1}^{n-1}(e_i \oplus e_{i+1} \oplus 1)$ (the sum is over integers).

Proof. Note that e denotes the carry vector, and the addition simply XORs the operands and the carry vector. It is left to ensure that the carry vector is correct. This requires only local constraints. The carry e_i is computed as

$$e_i = maj_3(e_{i+1}, x_{i+1}, y_{i+1}) = \\ x_{i+1}y_{i+1} \oplus x_{i+1}e_{i+1} \oplus y_{i+1}e_{i+1} = (x_{i+1} \oplus e_{i+1}) \cdot (y_{i+1} \oplus e_{i+1}) \oplus e_{i+1}.$$

where maj_3 is the majority function.

If $e_i \oplus e_{i+1} = 0$, we obtain a quadratic equation of the form $x_{i+1} \cdot y_{i+1} = 0$, which has 3 solutions. If $e_i \oplus e_{i+1} = 0$, we obtain a quadratic equation of the form $x_{i+1} \cdot y_{i+1} = 1$, which has 1 solution, and is equivalent to two linear equations $x_{i+1} = 1$ and $y_{i+1} = 1$. For all positions the constraints on x, y are independent. □

Example 7. The all-zero carry patterns are the most probable, and the probability is equal to $(3/4)^{n-1}$. The second most probable patterns are those with one adjacent difference, i.e. of the form $e = (1, 1, \dots, 1, 0, \dots, 0)$. Their probability is equal to $(3/4)^{n-1}/3$.

As observed in the proof, a difference in adjacent carry bits results in lower probability of linearization, and results in two linear equations instead of one quadratic equation. Therefore, carry patterns with more differences in adjacent bits should result in easier equation systems but also in a lower number of solutions. In general, an extra adjacent difference reduces the probability by a factor of 3.

11.3.4 Linearization Results

I implemented the algorithm in SageMath [SD19] and applied it to all 8 constants used in the ARX-boxes in Sparkle. In addition, I ran the algorithm on the all-zero and the all-one constants. The results are given in Table 11.4. For all constants except the all-one constant, the equation generation took a couple of minutes and the solving part took around an hour. The all-one constant had no extra equations and, due to an unusually large number of solutions, the computations have not finished even after 200 hours, yielding more than 4 millions of inputs. The evaluation was performed on a single core of a 2.8 GHz CPU on a laptop.

constant	hexadecimal	# equations	# inputs	example
c_0	B7E15162	199	13	(05600000, 70000225)
c_1	BF715880	199	11	(2A001990, 00188000)
c_2	38B4DA56	196	18	(1000C000, 144A0528)
c_3	324E7738	196	3	(1000E620, 04270080)
c_4	BB1185EB	193	10	(001C8181, 10808201)
c_5	4F7C7B57	160	340	(08301013, 28265722)
c_6	CFBFA1C8	178	105	(801D8000, 2FD10085)
c_7	C2B3293D	199	76	(00220110, 20001804)
0	00000000	310	8	(00000000, 40200080)
$2^{32}-1$	FFFFFFFF	124	$\geq 2^{22}$	(0B11CC51, 72770942)

TABLE 11.4: The number of inputs for ARX-boxes inflicting no carries in all four rounds, for different round constants.

The first interesting observation is that the number of solutions is much smaller than $2^{12.5} \approx 5900$ predicted under the round independence assumption. For 5 out of 8 used constants, the number of solutions is less than 20, and the maximum number of solutions among constants used in SPARKLE is 340. The second observation is that, for the zero constant, the number of solutions is also extremely low. We find it rather counter-intuitive, since in absence of constants many low-weight vectors can be expected to pass through the ARX-box without inflicting any carries. We suggest that this happens due to strong choice of rotation amounts, leading to faster diffusion. Finally, it turns out that the all-one constant leads to a huge number of solutions.

I observed similar behaviour and verified correctness of the algorithm on 8-bit words, where an exhaustive search over all ARX-box inputs is feasible.

I also applied the algorithm to a carry pattern with a single difference in adjacent carry bits, namely when the carry pattern in the first round is

$$e = (1, 1, \dots, 1, 0) \in \mathbb{F}_2^{32}$$

and in other rounds the carry pattern is zero. I generated the equations and ran the guess-and-determine algorithm on ARX-boxes with constants c_0 and c_5. The two linear equations that appear due to the carry difference allow to generate much more quadratic equations. Note that the algorithm for generating equations can be further improved to use linear relations in various ways. For the ARX-box A_{c_0}, 301 total equations are obtained, whereas for A_{c_5}, the system contains 409 equations. Though, the running time was still about 1 hour. The results for this ARX-boxes and the described carry pattern are given in Table 11.5.

11.4 Truncated Differential Analysis of SPARKLE

We performed exhaustive search of all structural truncated trails in SPARKLE, i.e. when each branch can be either active or inactive. Our approach consists of

constant	hexadecimal	# equations	# inputs	example
c_0	`B7E15162`	301	41	(`1F5D7FF5`, `B2D168B5`)
c_5	`4F7C7B57`	409	6	(`7ED77B73`, `A3DCCEE7`)

TABLE 11.5: The number of inputs for ARX-boxes inflicting the carry pattern $(1, \dots, 1, 0)$ in the first round an no carries in the other rounds, for round constants c_0 and c_5.

two steps. The first step is to generate the matrix of probabilities of all truncated transitions through the linear layer. I propose a new generic and precise method for this step. The second step is a simple iterative search, where for each round and for each truncated pattern at this round we keep the best truncated trail leading tho this pattern.

11.4.1 Generating Truncated Trail Matrix of a Linear Layer

I describe a generic method to generate the matrix of probabilities of truncated transitions from the binary matrix of the analyzed linear layer.

Definition 11.8. *Let $L : (\mathbb{F}_2^m)^t \to (\mathbb{F}_2^m)^t$ be a linear bijective mapping. An* exact truncated transition *over L is a pair of vectors from $\{0, +\}^t$. A* loose truncated transition *over L is a pair of vectors from $\{0, *\}^t$. A truncated transition α, β over L is denoted $\alpha \xrightarrow{L} \beta$.*

Definition 11.9 (Support Sets)**.**
The support *of a symbol $\gamma_i \in \{0, *, +\}$ is defined as the set*

$$p(\gamma_i) := \begin{cases} \{0\}, & \text{if } \gamma_i = 0, \\ \mathbb{F}_2^m, & \text{if } \gamma_i = *, \\ \mathbb{F}_2^m \setminus \{0\}, & \text{if } \gamma_i = +. \end{cases}$$

The support *of a vector $\gamma \in \{0, *, +\}^t$ is defined as the set*

$$p(\gamma) := p(\gamma_0) \times \ldots \times p(\gamma_{t-1}).$$

The support *of a truncated transition $\alpha \xrightarrow{L} \beta$ is defined as the set*

$$p(\alpha \xrightarrow{L} \beta) := \{(x, L(x))| \mid x \in p(\alpha), L(x) \in p(\beta)\} \subseteq (\mathbb{F}_2^{mt})^2.$$

Definition 11.10. *The* cardinality *of a truncated transition $\alpha \xrightarrow{L} \beta$ is defined as the cardinality of its support:*

$$|\alpha \xrightarrow{L} \beta| := |p(\alpha \xrightarrow{L} \beta)| = |L(p(\alpha)) \cap p(\beta)|.$$

The probability *of a truncated transition $\alpha \xrightarrow{L} \beta$ is defined as*

$$\Pr\Big[\alpha \xrightarrow{L} \beta\Big] := \Pr_{x \in p(\alpha)}[L(x) \in p(\beta)] = \frac{|\alpha \xrightarrow{L} \beta|}{|p(\alpha)|}.$$

Remark 23. The cardinality is a property of the *graph* of L, whereas the probability differentiates the input from the output. In particular, the problem of computing cardinalities of truncated transitions is equivalent to the problem of finding the number of codewords fitting the truncated mask $(\alpha, \beta) \in \mathbb{F}_2^{2mt}$ in the linear code with the generator matrix $\left[I_{mt \times mt} \mid L\right]$. The matrix of exact truncated transitions also trivially allows to compute the branching number of the matrix.

Finally, the table of truncated transition probabilities can be formally defined.

Definition 11.11 (TTT)**.** *Let $L\colon (\mathbb{F}_2^m)^t \to (\mathbb{F}_2^m)^t$ be a linear bijective mapping. The* table of truncated transitions *(TTT) of L is the $2^t \times 2^t$ matrix TTT_L given by*

$$\mathsf{TTT}_L[a, b] = \Pr\left\{\alpha \xrightarrow{L} \beta\right\},$$

where $a, b \in \mathbb{F}_2^t$ are mapped naturally to $\alpha', \beta' \in \{0, +\}^t$ respectively.

Since $|p(\alpha)|$ is easy to compute, we focus on computing the cardinalities of loose and exact truncated transitions.

Cardinalities of Loose Transitions. The first step is to compute the cardinalities of *all* possible *loose* truncated transitions over L. Let $\alpha \xrightarrow{L} \beta$ be a loose transition over L. Observe that $p(\alpha), p(\beta)$ are linear subspaces of $\mathbb{F}_2^{mt}$. The cardinality $|\alpha \xrightarrow{L} \beta| = L(p(\alpha)) \cap p(\beta)$ can be computed as follows.

A vector $b \in (\mathbb{F}_2^m)^t$ belongs to $p(\beta)$ if and only if $\beta_i = 0$ implies $b_i = 0$ for all i. Let

$$\pi_\beta(b)\colon (\mathbb{F}_2^m)^t \to p(\beta)^\perp$$

be the linear map returning the part of the vector b consisting of all elements b_i for which $\beta_i = 0$. The dimension of the space $L(p(\alpha)) \cap p(\beta)$ can be computed as the nullity of the linear map

$$\phi\colon p(\alpha) \to p(\beta)^\top, \qquad \phi \coloneqq \pi_\beta \circ L.$$

Let L be given as a block matrix $(L_{i,j})_{i,j \in [1 \ldots t]}$ with blocks of size $m \times m$. The vector space $L(p(\alpha))$ is spanned by bit-columns of the submatrix

$$(L_{i,j}), \text{ where } i, j \in [1 \ldots t], \alpha_j = *.$$

Furthermore, the map π_β simply truncates its input vector to positions where $\beta_i = 0$. It follows that $\phi \coloneqq \pi_\beta \circ L$ can be given by the submatrix

$$L\big|_{\alpha=*,\beta=0} \coloneqq (L_{i,j}) \text{ where } i, j \in [1 \ldots t], \beta_i = 0, \alpha_j = *.$$

Given the binary rank of this matrix, the nullity of ϕ is computed as

$$\dim p(\alpha) - \operatorname{rank} L\big|_{\alpha=*,\beta=0}.$$

We conclude that $|\alpha \xrightarrow{L} \beta|$ can be computed as

$$|\alpha \xrightarrow{L} \beta| = 2^{\dim p(\alpha) - \operatorname{rank} L\big|_{\alpha=*,\beta=0}}.$$

Cardinalities of Exact Transitions. The second step is to compute the probabilities of all *exact* truncated transitions over L. Observe that a loose truncated truncated trail can be seen as a union of precise truncated trails. For example,

$$(*, 0) \xrightarrow{L} (0, *)$$

is equivalent to the following union of disjoint transitions:

$$\Big\{(0,0) \xrightarrow{L} (0,0), \quad (0,0) \xrightarrow{L} (0,+), \quad (+,0) \xrightarrow{L} (0,0), \quad (+,0) \xrightarrow{L} (0,+)\Big\},$$

i.e. the cardinalities are summed.

Lemma 11.12. *Let $\alpha \xrightarrow{L} \beta \neq \alpha' \xrightarrow{L} \beta'$ be exact truncated transitions. Then their supports are disjoint.*

Proof. The lemma follows from the fact that $p(+)$ and $p(0)$ are disjoint. □

Definition 11.13. *For any symbols/vectors/truncated transitions γ, γ' let*

$$\gamma \preceq \gamma' \text{ if and only if } p(\gamma) \subseteq p(\gamma').$$

Lemma 11.14. *Let $\alpha_* \xrightarrow{L} \beta_*$ be a loose truncated transition. Then the set*

$$\mathbf{P}(\alpha_* \xrightarrow{L} \beta_*) := \Big\{p(\alpha'_+ \xrightarrow{L} \beta'_+) \mid \alpha'_+, \beta'_+ \in \{0,+\}^t, (\alpha'_+ \xrightarrow{L} \beta'_+) \preceq (\alpha_* \xrightarrow{L} \beta_*)\Big\}$$

is a partition of $p(\alpha \xrightarrow{L} \beta)$.

Proof. It follows from Lemma 11.12 and the fact that $p(*) = p(+) \sqcup p(0)$, where $\sqcup$ denotes the disjoint union. □

Corollary 11.15. *Let $\alpha_+ \xrightarrow{L} \beta_+$ be an exact truncated transition and let $\alpha_* \xrightarrow{L} \beta_*$ be the same transition, but with all symbols $+$ replaced by $*$. Then*

$$|\alpha_+ \xrightarrow{L} \beta_+| = |\alpha_* \xrightarrow{L} \beta_*| - \sum_{\alpha'_+, \beta'_+ \in \{0,+\}^t, (\alpha'_+ \xrightarrow{L} \beta'_+) \preceq (\alpha_* \xrightarrow{L} \beta_*)} |p(\alpha'_+ \xrightarrow{L} \beta'_+)|.$$

This corollary gives an efficient way to compute cardinalities of all exact truncated transitions. By computing the cardinalities in the lexicographic order of transitions, we can ensure that all sub-transitions are processed before processing the current transition.

Given the cardinalities of exact transitions, it is easy to compute the probabilities of exact transitions, and thus, the matrix TTT_L.

Complexity. The time complexity of the naive implementation is $\mathcal{O}(4^t(tm)^3 + 4^t 4^t))$, where the first term corresponds to the complexity of the rank computation for all block-aligned submatrices of L, and the second term corresponds to the complexity of the summing over "subtransitions". The latter step can be done in one extra pass in time $\mathcal{O}(t \cdot 4^t)$ by an algorithm similar to the well-known algorithm for the Möbius transform. Then the complexity becomes fully dominated by the rank computations: $\mathcal{O}(4^t \cdot (tm)^3)$. The algorithm can be directly improved if a better algorithm for computing the ranks of all block-aligned submatrices exists.

11.4.2 Iterative Algorithm for Truncated Trail Search

The trails of truncated transitions that are the most useful for cryptanalysis, should have probabilities significantly higher than the probability of sampling the final truncated difference uniformly at random. Such trails may be used to distinguish the analyzed structure from an ideal primitive.

Definition 11.16. *A truncated trail $\alpha_0 \xrightarrow{L} \dots \xrightarrow{L} \alpha_r$ over $L : (\mathbb{F}_2^m)^t \to (\mathbb{F}_2^m)^t$ is said to be* effective *if all $\alpha_i \neq 0$ and*

$$\Pr\left[\alpha_0 \xrightarrow{L} \dots \xrightarrow{L} \alpha_r\right] > \Pr_{\delta \in (\mathbb{F}_2^m)^t \setminus \{0\}} [\delta \in p(\alpha_r)] = p(\alpha_r)/(2^{mt} - 1).$$

where the trail probability is equal to the

Given the TTT a linear layer, it is easy to compute the best effective truncated trails in an iterative way. The method is based on dynamic programming. For each round r, we keep for each truncated output mask the best probability of reaching this mask from arbitrary input mask over r rounds. Extension to $r+1$ rounds is done by enumerating best output masks for r rounds and extending them using the TTT. The algorithm sketch is given in Algorithm 11.5.

Algorithm 11.5 Search for best truncated trails.

Input: a binary matrix of $L : (\mathbb{F}_2^m)^t \to (\mathbb{F}_2^m)^t$, $\mathsf{TTT}_L : \mathbb{F}_2^t \times \mathbb{F}_2^t \to \mathbb{Q}$;
Output: the map $\alpha_r \mapsto q$ for the best effective truncated trails $\alpha_0 \xrightarrow{L} \dots \xrightarrow{L} \alpha_r$

```
1: $d_0 \leftarrow \{\alpha \mapsto 1 \mid \alpha \in \{0,+\}^t \setminus (0,\dots,0)\}$
2: for all $r \in [1\dots]$ do
3:     $d_r \leftarrow \{\alpha \mapsto 0 \mid \alpha \in \{0,+\}^t\}$
4:     $effective \leftarrow false$
5:     for all $\alpha \xrightarrow{L} \beta \in \mathsf{TTT}_L$ do
6:         $d_r(\beta) \leftarrow \max\left(d_r(\beta), d_{r-1}(\alpha) \cdot \Pr\left[\alpha \xrightarrow{L} \beta\right]\right)$
7:         if $d_r(\beta) > p(\beta)/(2^{mt} - 1)$ then        ▷ $p(\beta)$ is the support of $\beta$
8:             $effective \leftarrow true$
9:     if not $effective$ then
10:        return $d_r$                                        ▷ Trail recovery can be added
```

Using precise arithmetics over rationals, the precise TTT can be computed. For example, consider the AES MixColumn matrix $L\colon (\mathbb{F}_2^8)^4 \to (\mathbb{F}_2^8)^4$. The algorithm finds the following two-round effective trail:

$$(+,0,0,0) \xrightarrow{L} (+,+,+,+) \xrightarrow{L} (+,0,0,0),$$

which has probability

$$1/16581375 \approx 2^{-23.983}$$

which is greater than the probability of sampling the difference fitting $(+,0,0,0)$ uniformly at random from all non-zero differences, which is equal to

$$(2^8-1)/(2^{32}-1) = 1/16843009 \approx 2^{-24.006}.$$

The position of active word in the initial and output masks does not matter. The first transition has probability one due to the fact that L has branching number 5 (note that the algorithm was given only the binary matrix of L). The second transition is possible due to uneven distribution of weights of differences in the image of $p((+,+,+,+))$ under L. This is an interesting observation, though the difference between the probabilities is too small for exploiting it for the cryptanalysis purpose.

11.4.3 Truncated Trails in SPARKLE

For most linear layers used in practice, the probabilities of truncated transitions over the linear layer are usually close to powers of 2 raised to the word size. The error term is ignored as insignificant. Indeed, since the S-Boxes are fixed, the assumed independence between sequential truncated transitions does not hold.

Consider the linear layer of SPARKLE as a mapping of $(\mathbb{F}_2^{64})^{n_b}$ to itself. For the analysis of SPARKLE, we also utilize the assumption, as all the transition probabilities over the linear layer are very close to 2^{-64k} for some k. We strengthen the definition of an effective trail by requiring that the trail probability is higher than $2^{-64k+0.01}$, where k is the number of inactive words in the output mask.

For SPARKLE256, the longest effective truncated differential trail covers two steps and has probability 1. It can be described as follows, where + indicates an active branch and 0 indicates an inactive branch:

$$\begin{array}{rcccc} \text{input}: & 0 & 0 & 0 & + \\ \text{step 1}: & + & 0 & 0 & 0 \\ \text{step 2}: & + & + & + & 0 \end{array}.$$

Another similar one can be obtained using the input 00+0. When restricting the input difference to be only in the left branches (i.e., for the setting in SCHWAEMM128-128), the longest effective truncated differential trail covers only one step (and probability 1):

$$\begin{array}{rcccc} \text{input}: & + & 0 & 0 & 0 \\ \text{step 1}: & + & + & + & 0 \end{array}.$$

For SPARKLE384, the longest effective truncated differential trail also covers two steps and has probability 1:

$$\begin{array}{lcccccc} \text{input}: & 0 & 0 & 0 & + & 0 & 0 \\ \text{step } 1: & 0 & 0 & + & 0 & 0 & 0 \\ \text{step } 2: & + & + & + & 0 & 0 & + \end{array}.$$

Two similar ones can be obtained using inputs $0000+0$ and $00000+$.

For SPARKLE512, the longest effective truncated differential trail covers three steps and has probability close to 2^{-64}:

$$\begin{array}{lcccccccc} \text{input}: & 0 & 0 & 0 & 0 & 0 & + & 0 & + \\ \text{step } 1: & + & 0 & + & 0 & 0 & 0 & 0 & 0 \\ \text{step } 2: & 0 & + & 0 & + & + & 0 & + & 0 \\ \text{step } 3: & + & + & + & + & 0 & + & 0 & + \end{array},$$

where we associate a probability of 2^{-64} for the transition between step 1 and step 2.

11.5 Division Property Analysis

11.5.1 Division Property of the ARX-box Structure

I performed MILP-aided bit-based division property analysis [Tod15, TM16] on the ARX-box structure. The MILP encoding is rather straightforward. For the modular addition operation I used the following method.

Addition modulo 2^{32}. Let us encode the modular addition by encoding the carry propagation. For any $a, b, c \in \mathbb{F}_2$, let $c' = maj_3(a, b, c) \in \mathbb{F}_2$ and $y = a \oplus b \in \mathbb{F}_2$, where maj_3 is the majority function. Then, all possible such 5-tuples $(a, b, c, c', y) \in \mathbb{F}_2^5$ can be characterized by the two following integer inequalities:

1. $-a - b - c + 2c' + y \geq 0$,
2. $a + b + c - 2c' - 2y \geq 1$.

For any bit position, summing the input bits a, b with the input carry c results in the output bit y and the new carry c'. In my experiments, these two inequalities applied per each bit position generated precisely the correct division property table of addition modulo 2^n for n up to 7. There were some redundant transitions though, which do not affect the result.

First, I evaluated the general algebraic degree of the ARX-box structure based on the division property. The 5^{th} and 6^{th} rounds rotation constants were chosen as the 1^{st} and 2^{nd} rounds rotation constants respectively, as this will happen when two ARX-boxes will be chained. The inverse ARX-box structure starts with 4^{th} round rotation constants, then 3^{rd}, 2^{nd}, 1^{st}, 4^{th}, etc. The minimum and maximum degree among coordinates of the ARX-box structure and its inverse are given in Table 11.6. Even though these are just upper bounds, I expect that they are close to the actual values, as the division property was

ARX-Box rounds	1	2	3	4	ARX-Box inverse rounds	1	2	3	4
min	1	10	42	63	min	1	2	32	46
max	32	62	63	63	max	32	62	63	63

TABLE 11.6: The upper bounds on the minimum and maximum degree of the coordinates of the ARX-box and its inverse.

shown to be rather precise [TM16]. Thus, the ARX-box structure may have full degree in all its coordinates, but the inverse of the ARX-box has a coordinate of degree 46.

The block-size level division property of the ARX-box is such that, for any $1 \leq k \leq 62$, $\mathcal{D}^{64}_{k}$ maps to $\mathcal{D}^{64}_{1}$ after two rounds, and $\mathcal{D}^{64}_{63}$ maps to $\mathcal{D}^{64}_{2}$ after two rounds and to $\mathcal{D}^{64}_{1}$ after three rounds. The same holds for the inverse of the ARX-box.

The longest integral characteristic found with bit-based division property is for 6-round ARX-box, where the input has 63 active bits and the inactive bit is at the index 44 (i.e., there are 44 active bits from the left and 19 active bits from the right), and in the output 16 bits are balanced:

input active bits:
`11111111111111111111111111111111,11111111111101111111111111111111`,
balanced bits after 6-round ARX-box:
`????????????????????????BBBBBBBB,?????????BBBBBBBB???????????????`.

The inactive bit can be moved to indexes $45, 46, 47, 48$ as well, the balanced property after 6 round stays the same. For the 7-round ARX-box we did not find any integral distinguishers.

For the inverse ARX-box, the longest integral characteristic is for 5 rounds:

input active bits:
`11111111111111111111111111110111,11111111111111111111111111111111`,
balanced bits after 5-round ARX-box inverse:
`???????????????????????????????B,???????BBBBBBBB?????????????????`.

For ARX-box inverse with 6-rounds we did not find any integral characteristic.

As a conclusion, even though a single ARX-box has integral characteristics, for two chained ARX-boxes there are no integral characteristics that can be found using the state-of-the-art division property method.

11.5.2 Division Property of the SPARKLE Permutations

I performed MILP-aided bit-based division property analysis [Tod15, SWW16] on the SPARKLE permutation family.

For the MILP encoding of the linear layer, I used the original simple method from [SWW16]. Note that in [ZR17] it was shown that this method is imprecise and may result in extra trails and weaker distinguisher. The linear layer of Sparkle can be viewed as 16 independent linear layers of dimensions from 16×16 in Sparkle256 to 32×32 in Sparkle512. For these dimensions it may be possible to apply the precise encoding method from [ZR17]. However, due to the large state size, I found it to be still infeasible.

I performed bit-based division property evaluation of the reduced-round Sparkle permutations. Let there be $b-1$ active bits with the inactive bit at index 44 or $44+b/2$, as offset 44 results in the best bit-based integral characteristic for the ARX-box structure. Furthermore, the branch choice for the inactive bit does not affect the result, due to the rotational branch symmetry (inside each half of the state). The best integral characteristic I found is for 4 steps and an extra ARX-box layer, for all three Sparkle versions. Let us encrypt the half of the codebook, such that one bit in the left half of the input is constant and all other bits are taking all possible values. Then, after 4 steps and the ARX-box layer from the 5-th step, the right half of the state is balanced (i.e. sums to zero). I state and prove this characteristic using structural division property and show that, in fact, fewer active input bits are required. Namely, $64 \cdot n_b + 65$ active bits instead of $2 \cdot 64 \cdot n_b$.

Proposition 11.17. *Consider a* Sparkle*-like permutation of $\mathbb{F}_2^{2h}$, with arbitrary bijective ARX-Boxes permuting $\mathbb{F}_2^m$, arbitrary linear Feistel function and at least 4 branches, i.e. $n_b = 2h/m \geq 4$. Then, the following division property transition is satisfied over 4 steps and an extra ARX-box layer:*

$$\mathcal{D}^{(m,\dots,m),(m,\dots,m)}_{(0,\dots,0,1,m),(m,\dots,m)} \xrightarrow{A\circ(L\circ A)^4} \mathcal{D}^{(h),(h)}_{(1),(0)} \cup \mathcal{D}^{(h),(h)}_{(0),(2)},$$

where A denotes the ARX-box layer and L denotes the linear layer. In other words, the right half of the output sums to zero.

Proof. Without loss of generality, we assume that there is no rotation of branches. Indeed, any permutation of branches inside a half is equivalent to reordering ARX-boxes inside halves and to modifying the Feistel linear layer, which is not constrained in this proposition.

Step 1. The properties $\mathcal{D}^m_1$ and $\mathcal{D}^m_m$ are retained through the ARX-boxes. The right half is fully active, therefore the linear layer does not have mixing effect yet. The following division trail is unique:

$$\mathcal{D}^{(m,\dots,m),(m,\dots,m)}_{(0,\dots,0,1,m),(m,\dots,m)} \xrightarrow{A} \mathcal{D}^{(m,\dots,m),(m,\dots,m)}_{(0,\dots,0,1,m),(m,\dots,m)} \xrightarrow{L} \mathcal{D}^{(m,\dots,m),(m,\dots,m)}_{(m,\dots,m),(0,\dots,0,1,m)}.$$

Step 2. The ARX-box layer does not change anything again. The linear layer allows multiple division trails. Note that at most $(n_b-1)m-1$ active bits can be transferred through the Feistel linear function until the right half is saturated to fully active. Therefore, at least $m+1$ bits remain active in the left half. In particular, at least two branches remain active. As we will show, this is the only requirement to show the proposed balanced property. We reduce active bits to these two in order to cover all possible trails and simplify the proof. Up to permutation of branches inside the state halves,

$$\mathcal{D}^{(m,\dots,m),(m,\dots,m)}_{(m,\dots,m),(0,\dots,0,1,m)} \xrightarrow{A} \mathcal{D}^{(m,\dots,m),(m,\dots,m)}_{(m,\dots,m),(0,\dots,0,1,m)} \xrightarrow{L} \mathcal{D}^{(m,\dots,m),(m,\dots,m)}_{(0,\dots,0),(1,1,0,\dots,0)}.$$

Step 3. The two active branches remain active through the third step, since there is no mixing between them:

$$\mathcal{D}^{(m,\dots,m),(m,\dots,m)}_{(0,\dots,0),(1,1,0,\dots,0)} \xrightarrow{A} \mathcal{D}^{(m,\dots,m),(m,\dots,m)}_{(0,\dots,0),(1,1,0,\dots,0)} \xrightarrow{L} \mathcal{D}^{(m,\dots,m),(m,\dots,m)}_{(1,1,0,\dots,0),(0,\dots,0)}.$$

Step $4+A$. Similarly, the two active branches stay active after the ARX-box layer of the fourth step:

$$\mathcal{D}^{(m,\dots,m),(m,\dots,m)}_{(1,1,0\dots,0),(0,\dots,0)} \xrightarrow{A} \mathcal{D}^{(m,\dots,m),(m,\dots,m)}_{(1,1,0\dots,0),(0,\dots,0)}.$$

In the linear layer, there are several possibilities. The two active bits from the left half can be transferred to a single branch in the right half by the Feistel function. Then $\mathcal{D}^{(h),(h)}_{(2),(0)}$ is obtained that is mapped through the final ARX-box layer to $\mathcal{D}^{(h),(h)}_{(1),(0)}$, i.e., the left half is possibly not balanced. If one of the active bits is transferred by the linear layer, then $\mathcal{D}^{(h),(h)}_{(1),(1)}$ is obtained, which is covered by the previous case. Otherwise, two active branches remain after the linear layer and after the final ARX-box layer. The output division property in this case is $\mathcal{D}^{(h),(h)}_{(0),(2)}$. The following trails cover all possible trails up to branch permutations in each half:

$$\mathcal{D}^{(m,\dots,m),(m,\dots,m)}_{(1,1,0\dots,0),(0,\dots,0)} \xrightarrow{L} \mathcal{D}^{(m,\dots,m),(m,\dots,m)}_{(2,0,\dots,0),(0,0,\dots,0)}, \xrightarrow{A} \mathcal{D}^{(m,\dots,m),(m,\dots,m)}_{(1,0,\dots,0),(0,0,\dots,0)} \implies \mathcal{D}^{(h),(h)}_{(1),(0)},$$
$$\mathcal{D}^{(m,\dots,m),(m,\dots,m)}_{(1,1,0,\dots,0),(0,\dots,0)} \xrightarrow{L} \mathcal{D}^{(m,\dots,m),(m,\dots,m)}_{(0,\dots,0),(1,1,0,\dots,0)}, \xrightarrow{A} \mathcal{D}^{(m,\dots,m),(m,\dots,m)}_{(0,\dots,0),(1,1,0,\dots,0)} \implies \mathcal{D}^{(h),(h)}_{(0),(2)}.$$

It follows that the following division trail is impossible:

$$\mathcal{D}^{(m,\dots,m),(m,\dots,m)}_{(0,\dots,0,1,m),(m,\dots,m)} \xrightarrow{A\circ(L\circ A)^4} \mathcal{D}^{(h),(h)}_{(0),(1)}$$

Therefore, the right output half is balanced. □

Note that in the proof, a lot of active bits were omitted for simplicity, in order to cover all possible trails by a single one. However, as the bit-based division property analysis suggests, a more careful analysis does not yield any longer integral characteristic.

I evaluated also the inverses of the Sparkle permutations. Similarly, the bit-based division property with only one inactive bit (at offset 27 in the left or in the right half) suggested only a general structural distinguisher, similar to the one from Proposition 11.17.

Proposition 11.18. *Consider a* Sparkle*-like permutation as in Proposition 11.17. The following division property transition is satisfied over 4 steps in the reverse* *direction:*

$$\mathcal{D}^{(m,\dots,m),(m,\dots,m)}_{(m,\dots,m),(0,\dots,0,1,m)} \xrightarrow{(A^{-1}\circ L^{-1})^4} \mathcal{D}^{(h),(h)}_{(2),(0)} \cup \mathcal{D}^{(h),(h)}_{(0),(1)}.$$

Proof. In a similar way to the Proposition 11.17, the following division trail covers all division trails:

$$\begin{aligned}\mathcal{D}^{(m,\dots,m),(m,\dots,m)}_{(m,\dots,m),(0,\dots,0,1,m)} &\xrightarrow{L^{-1}} \mathcal{D}^{(m,\dots,m),(m,\dots,m)}_{(0,\dots,0,1,m),(m,\dots,m)} \xrightarrow{A^{-1}} \mathcal{D}^{(m,\dots,m),(m,\dots,m)}_{(0,\dots,0,1,m),(m,\dots,m)}\\ &\xrightarrow{L^{-1}} \mathcal{D}^{(m,\dots,m),(m,\dots,m)}_{(1,1,0,\dots,0),(0,\dots,0,1,m)} \xrightarrow{A^{-1}} \mathcal{D}^{(m,\dots,m),(m,\dots,m)}_{(1,1,0,\dots,0),(0,\dots,0,1,m)}\\ &\xrightarrow{L^{-1}} \mathcal{D}^{(m,\dots,m),(m,\dots,m)}_{(0,\dots,0),(1,1,0\dots,0)} \xrightarrow{A^{-1}} \mathcal{D}^{(m,\dots,m),(m,\dots,m)}_{(0,\dots,0),(1,1,0\dots,0)},\end{aligned}$$

And in the last step the same cases take place as in Proposition 11.17.

$$\begin{aligned}&\mathcal{D}^{(m,\dots,m),(m,\dots,m)}_{(0,\dots,0),(1,1,0\dots,0)} \xrightarrow{L^{-1}} \mathcal{D}^{(m,\dots,m),(m,\dots,m)}_{(0,\dots,0),(2,0,\dots,0)}, \xrightarrow{A^{-1}} \mathcal{D}^{(m,\dots,m),(m,\dots,m)}_{(0,\dots,0),(1,0,\dots,0)} \implies \mathcal{D}^{(h),(h)}_{(0),(1)},\\ &\mathcal{D}^{(m,\dots,m),(m,\dots,m)}_{(0,\dots,0),(1,1,0\dots,0)} \xrightarrow{L^{-1}} \mathcal{D}^{(m,\dots,m),(m,\dots,m)}_{(1,1,0,\dots,0),(0,\dots,0)}, \xrightarrow{A^{-1}} \mathcal{D}^{(m,\dots,m),(m,\dots,m)}_{(1,1,0,\dots,0),(0,\dots,0)} \implies \mathcal{D}^{(h),(h)}_{(2),(0)}.\end{aligned}$$

The following trail is impossible:

$$\mathcal{D}^{(m,\dots,m),(m,\dots,m)}_{(m,\dots,m),(0,\dots,0,1,m)} \xrightarrow{(A^{-1}\circ L^{-1})^4} \mathcal{D}^{(h),(h)}_{(1),(0)}$$

Therefore, the left output half is balanced. □

11.6 Cryptanalysis of SCHWAEMM

Diffusion is relatively fast in SPARKLE and we expect guess-and-determine attack to be infeasible. This section shows several attacks on round-reduced variants of SCHWAEMM. The attacks are summarized in Table 11.7.

Instance	Steps	Whitening	Method	Time	Data
SCHWAEMM128-128	3.5	no	data trade-off	2^{64}	2^{64}
SCHWAEMM192-192	3.5	no	data trade-off	2^{128}	2^{64}
SCHWAEMM256-256	3.5	no	data trade-off	2^{192}	2^{64}
SCHWAEMM256-256	3.5	no	guess and det.	2^{192}	1
SCHWAEMM128-128	4.5	no	birthday diff.	$2^{96+\epsilon}$	$2^{96-\epsilon}$
SCHWAEMM192-192	4.5	no	birthday diff.	$2^{128+\epsilon}$	$2^{128-\epsilon}$
SCHWAEMM256-256	4.5	no	birthday diff.	$2^{192}+2^{160+\epsilon}$	$2^{160-\epsilon}$
SCHWAEMM256-256	3.5	yes	birthday diff.	$2^{224+\epsilon}$	$2^{224-\epsilon}$

TABLE 11.7: Guess and determine attacks on SCHWAEMM instances. ϵ is an arbitrary positive parameter. 0.5 step denotes an extra layer of ARX-boxes.

Notation used in the Attacks

Consider an instance of SCHWAEMM. Let $\mathbf{A}_j^i$ denote j-th ARX-box at the left half of the state at step i together with the step constant addition:

$$\mathbf{A}_j^i(x) = \begin{cases} A_{c_0}(x \oplus c_i), & \text{if } j = 0, \\ A_{c_1}(x \oplus i), & \text{if } j = 1, \\ A_{c_j}(x), & \text{if } 2 \le j < h_b. \end{cases}$$

Let $\mathbf{B}_j^i$ denote the j-th ARX-box at the right half of the state at step i: $\mathbf{B}_j^i = A_{c_{h_b+j}}$. Let $\mathbf{A}^i$ denote the parallel application of $\mathbf{A}_0^i, \dots, \mathbf{A}_{h_b-1}^i$; $\mathbf{B}^i$ denote the parallel application of $\mathbf{B}_0^i, \dots, \mathbf{B}_{h_b-1}^i$.

Let $\mathbf{X}[a]$ denote the map $x \mapsto (x \oplus a)$. Let $\mathbf{M}$ denote the linear Feistel map $\mathcal{M}_{h_b}$ and let ℓ' denote the linear feed-forward function used in $\mathbf{M}$:

$$\ell'((x_1||x_2),(y_1||y_2)) = (y_2||y_1 \oplus y_2),(x_2||x_1 \oplus x_2), \qquad \text{where } x_1, x_2, y_1, y_2 \in \mathbb{F}_2^{16}.$$

Let $\mathbf{R}$ denote the rotation of h_b branches to the left by one position:

$$\mathbf{R}(x_0, \dots, x_{h_b-1}) = (x_1, \dots, x_{h_b-1}, x_0).$$

A high-level structure of a 4-round SPARKLE in a SCHWAEMM instance using the described notations is depicted in Figure 11.7.

Consider a known-plaintext scenario. The rate part of the state becomes known before and after a call to a (round-reduced) SPARKLE permutation. Let m_{in} be the initial rate part and m_{out} be the final rate part. We call the ARX-box layer a *half-step*. Note that in the considered scenario, any attack on t full steps can be trivially extended to $t + 1/2$ steps, since the final ARX-boxes in the rate part can be easily inverted.

Differential Assumptions on the ARX-boxes

A single isolated ARX-box does not have a strong resistance against differential attacks. Indeed, there is a differential trail with probability 2^{-6}. In our attacks, we assume that particular problems about differential transitions involving random differences can be solved efficiently, even though we do not propose concrete algorithms. For example, consider the problem of checking whether a random differential transition over an ARX-box is possible. A naive approach would require 2^{64} evaluations. However, we can expect that with a meet-in-the-middle method it can be done much more efficiently. Indeed, an ARX-box has only 4 rounds. We further assume that the difference distribution table (DDT) of an ARX-box is very sparse and such problems about differential transitions have few solutions on average.

The problems we consider are about finding all solutions of the following differential transition types:

Problem 1. $a \xrightarrow{A} b$, where $a, b \in \mathbb{F}_2^{64}$ are known random differences, A is an ARX-box or the inverse of an ARX-box,

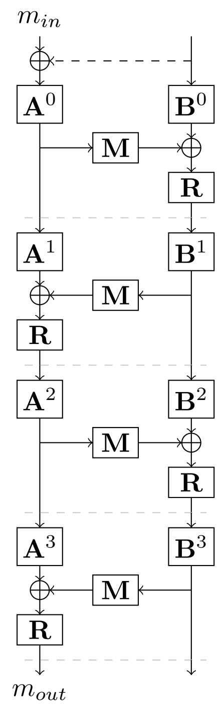


FIGURE 11.7: The High-level structure of SPARKLE with 4 steps.

Problem 2. $a \xrightarrow{A} \alpha, b \xrightarrow{B} \alpha$, where $a, b \in \mathbb{F}_2^{64}$ are known random differences, A, B are ARX-boxes or their inverses, $\alpha \in \mathbb{F}_2^{64}$ is an unknown difference.

Problem 3. $\alpha \xrightarrow{A} \beta, \alpha \xrightarrow{B} \beta + a$, where $a \in \mathbb{F}_2^{64}$ is a known random difference, A, B are ARX-boxes or their inverses, $\alpha, \beta \in \mathbb{F}_2^{64}$ are unknown differences.

We denote the average ratio of solutions to a problem by ν, and the average time to enumerate all solutions by τ_f.

11.6.1 Birthday-Differential Attacks

Encryptions with unique nonces can be expected to be completely independent. Therefore, a nonce-respecting adversary can not easily inject differences in the state in the encryption queries. Indeed, the difference between two encryptions in any part of the state can be expected to be random, and independent of the message due to the state randomization by the initialization with unique nonces. However, any fixed difference in n-bit part of the state may be obtained randomly among approximately $2^{n/2}$ random states. Therefore, with $2^{n/2}$ data, we can expect to have a pair satisfying an n-bit differential constraint. However, the procedure of finding this pair in the pool of encryptions has to be efficient.

The most useful differentials for this attack method are zero differences on full branches. They propagate to zero difference through ARX-boxes. It is also desirable that this differences imply the zero difference of some function of observable parts of the state (i.e. m_{in}, m_{out}). Then a hash table can be used to filter pairs from the data pool efficiently.

Proposition 11.19. *Assume that 64η bits in the encryption process are chosen such that for a pair of encryptions having zero difference in those 64η bits,*

1. *64μ bits can be efficiently computed from m_{in}, m_{out} (denote the function by π), such that they also have zero difference;*

2. *pairs of encryptions that satisfying the zero difference can be further filtered in time τ_f, keeping a fraction of most $\nu^{\eta-\mu}$ pairs (denote the function by* filter

3. *given such a pair, the full state can be recovered in time τ_r (denote the function by* recover*).*

Then, the full state can be recovered using $2^{64\eta/2+1/2}$ data and $2^{64(\eta-\mu)}(\tau_f + \nu^{\eta-\mu}\tau_r)$ time. The general attack procedure is given in Algorithm 11.6.

Proof. There are $2^{64\eta}$ pairs in the encryption pool and we can expect to have a pair having the required zero difference with a high probability. The complexity of the initial filtering by π can be neglected. Therefore, we assume that all $2^{64(\eta-\mu)}$ pair candidates (on average) can be enumerated efficiently. For each candidate, the verification and, in case of verification success, the state recovery take time $\tau_f + \nu^t\tau_r$. □

Algorithm 11.6 Birthday-Differential attack procedure.

```
collect $2^{64\eta/2+1/2}$ known-plaintext encryptions
compute corresponding rate parts $m_{in}, m_{out}$
store $\pi(m_{in}, m_{out})$ for each encryption in a hash table
for all $(m_{in}, m_{out}), (m'_{in}, m'_{out})$ such that $\pi(m_{in}, m_{out}) = \pi(m'_{in}, m'_{out})$ do
    if filter$((m_{in}, m_{out}), (m'_{in}, m'_{out}))$ then
        $s \leftarrow$ recover$((m_{in}, m_{out}), (m'_{in}, m'_{out}))$
        return $s$
```

Attacks of this type typically have quite large data complexity, violating the data limit set in the specification. However, it should be noted, that the actual key used does not matter as each state is always expected to be random and independent. Therefore, re-keying does not prevent the attack. If the required difference is achieved by a pair of encryptions under different keys, then both states are recovered by the attack.

An adversary can further exploit this fact. The data complexity may be reduced by performing a precomputation. The adversary encrypts 2^{64t} data (t may be fractional), and forms a pool in the same way as in the normal attack. Then, $2^{64(\eta-t)}$ data is collected from encryptions under the unknown secret key. Among too pools, there are $2^{64\eta}$ pairs and at least one pair will satisfy the zero difference with a high probability. Note that the data reduction starts only with $t > \eta/2$ and is costly in the time and memory complexity.

11.6.2 Attack on 3.5-step SCHWAEMM Instances without Rate Whitening

Consider an instance of SCHWAEMM with the rate equal to the capacity (i.e. one of SCHWAEMM128-128, SCHWAEMM192-192, SCHWAEMM256-256), which uses the SPARKLE permutation reduced to 3 steps and has no rate whitening.

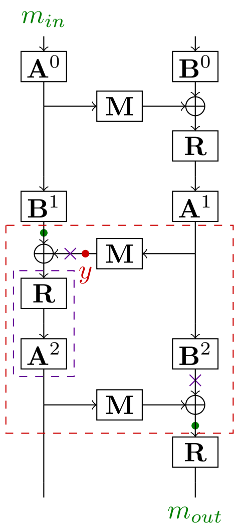


FIGURE 11.8: Attack on 3.5-step SPARKLE without whitening. The green dots show known values, the purple crosses show zero differences in the birthday-differential attack. The red dashed area highlights the part being attacked, the purple dashed area shows the part with the target differential transition in the birthday-differential attack.

Let y denote the output of the linear Feistel function $\mathbf{M}$ in second step (as shown in Figure 11.8). It lies on the following *cyclic* structure (marked with dashed red rectangle):

$$y = \mathbf{M} \circ (\mathbf{B}^2)^{-1} \circ \mathbf{X}\left[\mathbf{R}^{-1}(m_{out})\right] \circ \mathbf{M} \circ \mathbf{A}^2 \circ \mathbf{R} \circ \mathbf{X}\left[\mathbf{B}^1(\mathbf{A}^0(m_{in}))\right] \circ (y).$$

Let

$$m'_{in} = \mathbf{B}^1(\mathbf{A}^0(m_{in})),$$
$$m'_{out} = \mathbf{M}^{-1}(\mathbf{R}^{-1}(m_{out})).$$

Then

$$y = \Big(\mathbf{M} \circ (\mathbf{B}^2)^{-1} \circ \mathbf{M}\Big) \circ \Big(\mathbf{X}\left[m'_{out}\right] \circ \mathbf{A}^2 \circ \mathbf{R} \circ \mathbf{X}\left[m'_{in}\right]\Big)(y), \text{ and} \tag{11.2}$$

$$\mathbf{M}^{-1} \circ \mathbf{B}^2 \circ \mathbf{M}^{-1}(y) = \mathbf{X}\left[m'_{out}\right] \circ \mathbf{A}^2 \circ \mathbf{R} \circ \mathbf{X}\left[m'_{in}\right](y). \tag{11.3}$$

Note that Equation 11.2 shows that the unknown part of the state y is a *fixed point* of a particular bijective structure using the constants m'_{in}, m'_{out}. This is an interesting formulation of the constraint on the unknown part of the state.

Precomputation/data trade-off attack. Note that the left part of Equation 11.3 is independent of m'_{in}, m'_{out}. Moreover, the right part consists of independent ARX-boxes. Therefore, guessing one 64-bit branch of y leads to knowledge of an input and an output 64-bit branches of the function from the left-hand side. A data trade-off attack follows. The trade-off parameterized by an integer r, $0 < r \leq 64$.

We start by the precomputation phase. Let $z = \mathbf{M}^{-1} \circ \mathbf{B}^2 \circ \mathbf{M}^{-1}(y)$. We iterate over all $y_1 \in \mathbb{F}_2^{64-r}$ and all $y_i \in \mathbb{F}_2^{64}$ for $i \neq 1$, and generate the table mapping (y_1, z_0) to all values y satisfying the constraint. On average, we expect $2^{64h_b-r}/2^{64} = 2^{64(h_b-1)-r}$ candidates per each (y_1, z_0) in the table. This step requires 2^{64h_b-r} time and memory blocks.

In the online phase, we collect 2^r known plaintexts-ciphertext pairs and compute the corresponding m'_{in}, m'_{out} for each pair. Then, for each $y_1 \in \mathbb{F}_2^{64-r}$ we compute

$$z_0 = (m'_{out})_0 \oplus \mathbf{A}_0^2((m'_{in})_1 \oplus y_1).$$

For each preimage candidate of (y_1, z_0) in the precomputed table, we recover the full state in the middle of the second step. We then check if the corresponding state correctly connects m_{in}, m_{out} and possibly recover the secret key by inverting the sponge operation.

If a considered plaintext-ciphertext pair is such that the leftmost r bits of y_1 are equal to zero, then the attack succeeds. Indeed, then, for one of the guesses of y_1, the pair (y_1, z_0) corresponds to the correct preimage. For each of 2^r plaintext-ciphertext pairs we guess 2^{64-r} values of (y_1, z_0). Correct y_1 identifies a table mapping the z_0 to all possible y. Therefore, on average, there will be $2^{64-r} \cdot 2^{64(h_b-1)-r} = 2^{64h_b-2r}$ total candidates. The time required to check a candidate and to recover the secret key is negligible.

The online phase requires 2^r different 2-block plaintext-ciphertext pairs, 2^{64h_b-2r} time and negligible amount of extra memory.

The following attacks on SCHWAEMM instances follow:

1. SCHWAEMM128-128: with $r = 64$, the full attack requires 2^{64} time, memory and data; with $r = 32$, the full attack requires 2^{96} time and memory, and 2^{32} data.

2. SCHWAEMM192-192: with $r = 64$, the full attack requires 2^{128} time, memory and 2^{64} data.

3. SCHWAEMM256-256: with $r = 64$, the full attack requires 2^{192} time, memory and 2^{64} data.

Low-data variant of the attack on SCHWAEMM256-256. Due to the high branching number of $\mathbf{M}$, it is hard to exploit the structure of the function $\mathbf{M}^{-1} \circ \mathbf{B}^2 \circ \mathbf{M}^{-1}$ by guessing several branches. However, for the largest instance SCHWAEMM256-256, a simple attack requiring one known-plaintext and 2^{192} time is possible.

The key observation is that when $\ell'(x)$ is fixed, $\mathbf{M}(x)$ splits into h_b independent xors with $\ell'(x)$. In the attack, we simply guess the corresponding ℓ' for the two calls to $\mathbf{M}$. Precisely, let $\ell'_y = \ell'(\mathbf{M}^{-1}(y))$ and $\ell'_z = \ell'(\mathbf{M}^{-1} \circ \mathbf{B}^2 \circ \mathbf{M}^{-1}(y))$.

The computations from Equation 11.2 then split into one large cycle:

$$\begin{aligned}
y_0 &= \mathbf{X}\left[\ell'_y\right] \circ (\mathbf{B}_0^2)^{-1} \circ \mathbf{X}\left[\ell'_z\right] \circ \mathbf{X}\left[(m'_{out})_0\right] \circ \mathbf{A}_0^2 \circ \mathbf{X}\left[(m'_{in})_1\right](y_1),\\
y_1 &= \mathbf{X}\left[\ell'_y\right] \circ (\mathbf{B}_1^2)^{-1} \circ \mathbf{X}\left[\ell'_z\right] \circ \mathbf{X}\left[(m'_{out})_1\right] \circ \mathbf{A}_1^2 \circ \mathbf{X}\left[(m'_{in})_2\right](y_2),\\
y_2 &= \mathbf{X}\left[\ell'_y\right] \circ (\mathbf{B}_2^2)^{-1} \circ \mathbf{X}\left[\ell'_z\right] \circ \mathbf{X}\left[(m'_{out})_2\right] \circ \mathbf{A}_2^2 \circ \mathbf{X}\left[(m'_{in})_3\right](y_3),\\
y_3 &= \mathbf{X}\left[\ell'_y\right] \circ (\mathbf{B}_3^2)^{-1} \circ \mathbf{X}\left[\ell'_z\right] \circ \mathbf{X}\left[(m'_{out})_3\right] \circ \mathbf{A}_3^2 \circ \mathbf{X}\left[(m'_{in})_0\right](y_0).
\end{aligned}$$

Let us guess y_0 and compute the whole cycle. If the result matches guessed y_0, then we obtain a candidate for the full $y = (y_0, y_1, y_2, y_3)$. On average, we can expect to find one false-positive candidate.

The attack requires 1 known plaintext-ciphertext pair, negligible amount of memory, and 2^{192} time.

Birthday-differential attack. A birthday-differential attack can be mounted too. We are looking for a pair having zero difference in y. Then the expression

$$\mathbf{X}\left[m'_{out}\right] \circ \mathbf{A}^2 \circ \mathbf{R} \circ \mathbf{X}\left[m'_{in}\right](y) \tag{11.4}$$

has zero difference in the input y and zero difference in the output. Therefore, the difference in m'_{in} is transformed into the difference in $\mathbf{R}(m'_{out})$ by an ARX-box layer. This is the first problem we noted in Section 11.6.

Note that the amount of pairs of encryptions in the pool has to be greater than 2^{64h_b} in order for a pair with zero difference in y to exist. Therefore, enumeration of all pairs and checking the possibility of the differential transition $m'_{in} \xrightarrow{\mathbf{A}^2\circ\mathbf{R}} m'_{out}$ results in an ineffective attack.

As described in the birthday-differential attack framework, we further strengthen the constraints in order to obtain an efficient initial filtering. We require that t branches of m'_{in} starting from the second branch have zero difference too, $0 < t < h_b$. Then, m'_{out} must have zero difference in the first t branches. This allows us to obtain initial filtering with $\mu = 2t$, i.e. with the probability $2^{-64\cdot 2t}$. In total we need zero difference in $h_b + t$ branches. Therefore, we need $2^{64(h_b+t)/2+1/2}$ data and we expect to keep $2^{64(h_b+t)} \cdot 2^{-64\cdot 2t} = 2^{64(h_b-t)}$ pairs on average after the initial filtering procedure.

The second filtering step is based on filtering possible differential transitions. In the correct pair, the differences $\Delta m'_{in}$ and $\Delta m'_{out}$ of the values m'_{in} and m'_{out} respectively are related by the layer $\mathbf{A}^2$ of ARX-boxes. More precisely, for all $i, 0 \le i < h_b$, the following differential transition holds:

$$(\Delta m'_{in})_{i+1} \xrightarrow{\mathbf{A}_i^2} (\Delta m'_{out})_i.$$

Verifying a pair requires checking whether a differential transition over an ARX-box is possible or not (see Problem 1 in Section 11.6). We assume that there only a fraction ν of all differential transitions over an ARX-box is possible, and that for any differential transitions all solutions can be found in time τ_f on average.

The branch values corresponding to zero difference transitions can be found exhaustively in time $\tau_r \leq 2^{64t}$ or more efficiently by exploiting the structure further.

We estimate the final complexity of the attack by $2^{64(h_b+t)/2+1/2}$ data and $2^{64(h_b-t)}(\tau_f + \nu^{h_b-t}\tau_r)$ time. Assuming low values of ν, τ_f and τ_r, we estimate the following attack complexities for different instances of SCHWAEMM:

1. SCHWAEMM128-128: with $t = 1$, the attack requires $2^{96.5}$ data, and slightly more than 2^{64} time. By the precomputation cost of $2^{96+\epsilon}$ time and memory, the data requirement can be reduced to $2^{96-\epsilon}$ for any $\epsilon < 32$.
2. SCHWAEMM192-192: with $t = 1$, the attack requires $2^{128.5}$ data, and slightly more than 2^{128} time. By the precomputation cost of $2^{128+\epsilon}$ time and memory, the data requirement can be reduced to $2^{128-\epsilon}$ for any $\epsilon < 64$.
3. SCHWAEMM256-256: with $t = 1$, the attack requires $2^{160.5}$ data, and slightly more than 2^{192} time. By the precomputation cost of $2^{160+\epsilon}$ time and memory, the data requirement can be reduced to $2^{160-\epsilon}$ for any $\epsilon < 96$.

11.6.3 Attack on 4.5-step SCHWAEMM without Rate Whitening

Consider an instance of SCHWAEMM with the rate equal to the capacity (i.e. one of SCHWAEMM128-128, SCHWAEMM192-192, SCHWAEMM256-256), which uses the SPARKLE permutation reduced to 4 steps and has no rate whitening.

Let y be the input to the linear Feistel layer $\mathbf{M}$ in the third step (see Figure 11.9). We aim to mount a birthday-differential attack with zero-difference in y. The parts of the structure with zero difference are marked with purple crosses in the figure. It follows that differences of the observed rate parts can be propagated and connected by independent branches. More formally, let

$$\begin{aligned} m'_{in} &= \mathbf{M}^{-1}(\mathbf{B}^1(\mathbf{A}^0(m_{in}))), \\ m'_{out} &= \mathbf{M}^{-1}(\mathbf{R}(m_{out})). \end{aligned}$$

Denote the difference in m'_{in} by $\Delta m'_{in}$, and the difference in m'_{out} by $\Delta m'_{out}$. It follows that the difference $\Delta m'_{in}$ propagates through $\mathbf{B}^2$ into the same difference as the difference $\Delta m'_{out}$ propagates through $\mathbf{R}^{-1} \circ (\mathbf{A}^3)^{-1}$. Note that they are connected by h_b independent 64-bit branches:

$$(\Delta m'_{in})_i \xrightarrow{\mathbf{B}_i^2} \alpha_i \xleftarrow{(\mathbf{A}_{i-1}^3)^{-1}} (\Delta m'_{out})_{i-1}, \tag{11.5}$$

where α is the unknown intermediate difference.

In order to make the birthday-differential attack, we further strengthen the zero-difference constraint in order to perform an efficient initial filtering. We require that $(\Delta m'_{in})_i = \alpha_i = (\Delta m'_{out})_{i-1} = 0$ for all $i < t$ for an integer t, $0 < t < h_b$. This constraints allows us to filter the pairs efficiently by the zero-difference parts of m'_{in} and m'_{out}.

The second filtering step is based on checking the possibility of the differential transitions from Equation 11.5. This is Problem 2 mentioned in Section 11.6.

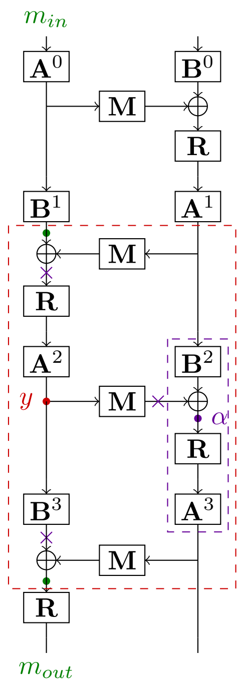


FIGURE 11.9: Attack on 4.5-step SPARKLE without whitening. The green dots show known values, the purple crosses show zero differences. The red dashed area highlights the part being attacked, the purple dashed area shows the part with the target differential transition.

Similarly to the previous attack, we the final complexity of the attack is estimated by $2^{64(h_b+t)/2+1/2}$ data and $2^{64(h_b-t)}(\tau_f + \nu^{h_b-t}\tau_r)$ time. Under the assumption of low values of ν, τ_f and τ_r, the following attacks are derived:

1. SCHWAEMM128-128: with $t = 1$, the attack requires $2^{96.5}$ data, and more than 2^{64} time. By the precomputation cost of $2^{96+\epsilon}$ time and memory, the data requirement can be reduced to $2^{96-\epsilon}$ for any $\epsilon < 32$.

2. SCHWAEMM192-192: with $t = 1$, the attack requires $2^{128.5}$ data, and more than 2^{128} time. By the precomputation cost of $2^{128+\epsilon}$ time and memory, the data requirement can be reduced to $2^{128-\epsilon}$ for any $\epsilon < 64$.

3. SCHWAEMM256-256: with $t = 1$, the attack requires $2^{160.5}$ data, and more than 2^{192} time. By the precomputation cost of $2^{160+\epsilon}$ time and memory, the data requirement can be reduced to $2^{160-\epsilon}$ for any $\epsilon < 96$.

11.6.4 Attack on 3.5-step SCHWAEMM256-256

Consider SCHWAEMM256-256, which uses the SPARKLE permutation reduced to 3 steps and has the rate whitening.

Let y be the input to the linear Feistel function $\mathbf{M}$ in the second step (see Figure 11.10). We aim to find a pair of encryptions with zero difference in y. We further restrict the input and the output difference of $\mathbf{M}$ in the first round to have form $\boldsymbol{\alpha} = (\alpha, \alpha, 0, 0$ for any $\alpha \in \mathbb{F}_2^{64}$. Note that this happens in the fraction

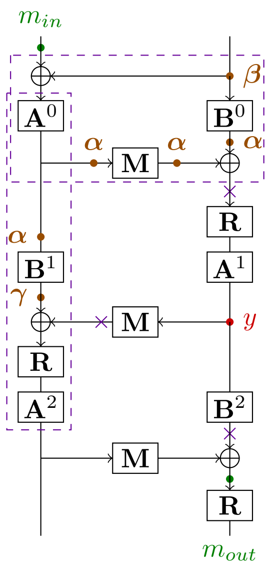


FIGURE 11.10: Attack on 3.5-step SPARKLE with whitening. The green dots show known values, the purple crosses show zero differences. The purple dashed areas shows the parts with the target differential transitions.

$2^{-3\cdot 64}$ of all inputs to M, because $(\alpha, \alpha, 0, 0)$ is always mapped to $(\alpha, \alpha, 0, 0)$ by M. In total, we require 7 independent branches to have zero difference.

First, observe that for some $\boldsymbol{\beta} \in (\mathbb{F}_2^{64})^4 = (\beta_0, \beta_1, 0, 0)$, the following differential transitions hold (see the topmost purple area in Figure 11.10):

$$\boldsymbol{\alpha} \xrightarrow{\mathbf{A}^0} \beta \oplus \Delta m_{in},$$
$$\boldsymbol{\alpha} \xrightarrow{\mathbf{B}^0} \beta,$$

where Δm_{in} is the difference in m_{in}. It follows that $(\Delta m_{in})_2 = (\Delta m_{in})_3 = 0$, because $\boldsymbol{\alpha}_2 = \boldsymbol{\alpha}_3 = 0$. For $i = 0$ and $i = 1$ we obtain an instance of Problem 3 from Section 11.6:

$$\alpha \xrightarrow{\mathbf{A}_i^0} \beta_i \oplus (m_{in})_i,$$
$$\alpha \xrightarrow{\mathbf{B}_i^0} \beta_i.$$

Note that here the same unknown $\alpha \in \mathbb{F}_2^{64}$ appears in two instances of the problem, thus adding more constraints on α. Consider the leftmost purple area in Figure 11.10. It describes another differential transition:

$$\boldsymbol{\alpha} \xrightarrow{\mathbf{B}^1} \boldsymbol{\gamma} \xrightarrow{\mathbf{A}^2 \circ \mathbf{R}} \Delta m'_{out},$$

where $\boldsymbol{\gamma} \in (\mathbb{F}_2^{64})^4 = (\gamma_0, \gamma_1, 0, 0)$ and $\Delta m'_{out}$ is the difference of $m'_{out} = \mathbf{M}^{-1}(\mathbf{R}^{-1}(m_{out}))$. It follows that $(\Delta m'_{out})_1 = (\Delta m'_{out})_2 = 0$ and for $i = 0$ and $i = 1$, the following differential transition holds:

$$\alpha \xrightarrow{\mathbf{B}_i^1} \boldsymbol{\gamma}_i \xrightarrow{\mathbf{A}_{i-1}^2} (\Delta m'_{out})_{i-1}.$$

In total, $\eta = 7$ branches are constrained to have zero difference and $\mu = 4$ branches with zero differences can be observed from m_{in}, m_{out}, providing strong initial filter. Using $2^{64\eta/2+1/2}$ data, we expect to get $2^{64(\eta-\mu)} = 2^{64\cdot 3}$ encryption pairs after the initial filtering. Furthermore, we assume that the constraints on the unknown difference $\alpha \in \mathbb{F}_2^{64}$ are very strong and are enough to significantly reduce the number of possible encryption pairs. We assume it can be done efficiently, since a precomputation time of $264 \cdot 3$ is available. After values of branches involved in differential transitions with α are recovered, the rest of the state can be recovered in negligible time.

Therefore, we estimate the data complexity of the attack by $2^{224.5}$ and same time complexity (the heavy filtering step has to filter 2^{192} pairs). By precomputations costing $2^{224+\epsilon}$ time and memory, the data complexity may be reduced to $2^{224-\epsilon}$, for any $\epsilon < 32$.

This attack does not directly apply to SCHWAEMM128-128, SCHWAEMM192-192 since the constraint on the linear map $\mathbf{M}$ in the first step is too costly. For SCHWAEMM$192-192$ with $\boldsymbol{\alpha} = (\alpha, \alpha, 0)$ we would obtain $\eta = 5, \mu = 2$ leaving with 2^{192} pair candidates, which is too much to filter in time 2^{192}. Therefore, a stronger initial filter is required.

Bibliography

[AA04] Sergey Agievich and Andrey Afonenko. Exponential S-boxes. Cryptology ePrint Archive, Report 2004/024, 2004. http://eprint.iacr.org/2004/024.

[AA05] S.V. Agievich and A.A. Afonenko. О свойствах экспоненциальных подстановок [On properties of the exponential S-Boxes]. In *Вести НАН Беларуси [News of The National Academy of Sciences of Belarus]*, volume 1, pages 106–112. National Academy of Sciences of Belarus, 2005. Available at http://elib.bsu.by/handle/123456789/24138.

[ABD+] Elena Andreeva, Andrey Bogdanov, Nilanjan Datta, Atul Luykx, Bart Mennink, Mridul Nandi, Elmar Tischhauser, and Kan Yasuda. COLM v1. CAESAR candidate. http://competitions.cr.yp.to/round3/deoxysv141.pdf.

[ABL+13] Elena Andreeva, Andrey Bogdanov, Atul Luykx, Bart Mennink, Elmar Tischhauser, and Kan Yasuda. Parallelizable and Authenticated Online Ciphers. In Kazue Sako and Palash Sarkar, editors, *Advances in Cryptology - ASIACRYPT 2013 - 19th International Conference on the Theory and Application of Cryptology and Information Security, Bengaluru, India, December 1-5, 2013, Proceedings, Part I*, volume 8269 of *Lecture Notes in Computer Science*, pages 424–443. Springer, 2013.

[AFF+14] Farzaneh Abed, Scott R. Fluhrer, Christian Forler, Eik List, Stefan Lucks, David A. McGrew, and Jakob Wenzel. Pipelineable On-line Encryption. In Carlos Cid and Christian Rechberger, editors, *Fast Software Encryption - 21st International Workshop, FSE 2014, London, UK, March 3-5, 2014. Revised Selected Papers*, volume 8540 of *Lecture Notes in Computer Science*, pages 205–223. Springer, 2014.

[AGMK02] S.V. Agievich, V.A. Galinskij, N.D. Mikulich, and Y.S. Kharin. Алгоритм блочного шифрования BelT [Block Cipher BelT]. In *Управление защитой информации [Information Security Management]*, volume 6(4), pages 407–412. Belarusian State University, 2002. Available at http://elib.bsu.by/handle/123456789/24140.

[AJN14] Jean-Philippe Aumasson, Philipp Jovanovic, and Samuel Neves. Analysis of NORX: Investigating Differential and Rotational

Properties. In Diego F. Aranha and Alfred Menezes, editors, *Progress in Cryptology - LATINCRYPT 2014 - Third International Conference on Cryptology and Information Security in Latin America, Florianópolis, Brazil, September 17-19, 2014, Revised Selected Papers*, volume 8895 of *Lecture Notes in Computer Science*, pages 306–324. Springer, 2014.

[AJN15] Jean-Philippe Aumasson, Philipp Jovanovic, and Samuel Neves. NORX8 and NORX16: Authenticated Encryption for Low-End Systems. *Trustworthy Manufacturing and Utilization of Secure Devices—TRUDEVICE*, 2015.

[AJN16] Jean-Philippe Aumasson, Philipp Jovanovic, and Samuel Neves. NORX v3.0. CAESAR candidate. https://competitions.cr.yp.to/round3/norxv30.pdf, https://norx.io/data/norx.pdf, 2016.

[BB17] Paul Bottinelli and Joppe W. Bos. Computational aspects of correlation power analysis. *Journal of Cryptographic Engineering*, 7(3):167–181, Sep 2017.

[BBdS$^+$19a] Christof Beierle, Alex Biryukov, Luan Cardoso dos Santos, Johann Großschädl, Léo Perrin, Aleksei Udovenko, Vesselin Velichkov, and Qingju Wang. Schwaemm and Esch: Lightweight Authenticated Encryption and Hashing using the Sparkle Permutation Family, 2019.

[BBdS$^+$19b] Christof Beierle, Alex Biryukov, Luan Cardoso dos Santos, Johann Großschädl, Léo Perrin, Aleksei Udovenko, Vesselin Velichkov, and Qingju Wang. Schwaemm and Esch: Weightless Authenticated Encryption and Cheap Hashing for Micro Controllers using the Sparkle Permutation Family, 2019. To appear.

[BBF16] Arnaud Bannier, Nicolas Bodin, and Eric Filiol. Partition-Based Trapdoor Ciphers. Cryptology ePrint Archive, Report 2016/493, 2016. https://eprint.iacr.org/2016/493.

[BBI$^+$15] Subhadeep Banik, Andrey Bogdanov, Takanori Isobe, Kyoji Shibutani, Harunaga Hiwatari, Toru Akishita, and Francesco Regazzoni. Midori: A Block Cipher for Low Energy. In Tetsu Iwata and Jung Hee Cheon, editors, *Advances in Cryptology - ASIACRYPT 2015 - 21st International Conference on the Theory and Application of Cryptology and Information Security, Auckland, New Zealand, November 29 - December 3, 2015, Proceedings, Part II*, volume 9453 of *Lecture Notes in Computer Science*, pages 411–436. Springer, 2015.

[BBIJ17] Subhadeep Banik, Andrey Bogdanov, Takanori Isobe, and Martin Jepsen. Analysis of Software Countermeasures for Whitebox Encryption. *IACR Transactions on Symmetric Cryptology*, 2017(1):307–328, Mar. 2017.

[BBK14a] Alex Biryukov, Charles Bouillaguet, and Dmitry Khovratovich. Cryptographic Schemes Based on the ASASA Structure: Black-Box, White-Box, and Public-Key. In Palash Sarkar and Tetsu Iwata, editors, *Advances in Cryptology – ASIACRYPT 2014*, pages 63–84, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.

[BBK14b] Alex Biryukov, Charles Bouillaguet, and Dmitry Khovratovich. Cryptographic Schemes Based on the ASASA Structure: Black-Box, White-Box, and Public-Key (Extended Abstract). In Palash Sarkar and Tetsu Iwata, editors, *Advances in Cryptology - ASIACRYPT 2014 - 20th International Conference on the Theory and Application of Cryptology and Information Security, Kaoshiung, Taiwan, R.O.C., December 7-11, 2014. Proceedings, Part I*, volume 8873 of *Lecture Notes in Computer Science*, pages 63–84. Springer, 2014.

[BBMT18] Estuardo Alpirez Bock, Chris Brzuska, Wil Michiels, and Alexander Treff. On the Ineffectiveness of Internal Encodings - Revisiting the DCA Attack on White-Box Cryptography. In Bart Preneel and Frederik Vercauteren, editors, *Applied Cryptography and Network Security - 16th International Conference, ACNS 2018, Leuven, Belgium, July 2-4, 2018, Proceedings*, volume 10892 of *Lecture Notes in Computer Science*, pages 103–120. Springer, 2018.

[BBU18] Christof Beierle, Alex Biryukov, and Aleksei Udovenko. On Degree-d Zero-Sum Sets of Full Rank. Cryptology ePrint Archive, Report 2018/1194, 2018. https://eprint.iacr.org/2018/1194.

[BC13] Christina Boura and Anne Canteaut. On the Influence of the Algebraic Degree of F^{-1} on the Algebraic Degree of $G \circ F$. *IEEE Trans. Information Theory*, 59(1):691–702, 2013.

[BC16] Christina Boura and Anne Canteaut. Another View of the Division Property. 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 I*, volume 9814 of *Lecture Notes in Computer Science*, pages 654–682. Springer, 2016.

[BCBP03] Alex Biryukov, Christophe De Cannière, An Braeken, and Bart Preneel. A Toolbox for Cryptanalysis: Linear and Affine Equivalence Algorithms. In Eli Biham, editor, *Advances in Cryptology - EUROCRYPT 2003, International Conference on the Theory and Applications of Cryptographic Techniques, Warsaw, Poland, May 4-8, 2003, Proceedings*, volume 2656 of *Lecture Notes in Computer Science*, pages 33–50. Springer, 2003.

[BCD06] Julien Bringer, Herve Chabanne, and Emmanuelle Dottax. White Box Cryptography: Another Attempt. Cryptology ePrint Archive, Report 2006/468, 2006. http://eprint.iacr.org/2006/468.

[BCL18] Christof Beierle, Anne Canteaut, and Gregor Leander. Nonlinear Approximations in Cryptanalysis Revisited. *IACR Trans. Symmetric Cryptol.*, 2018(4):80–101, 2018.

[BCLR17] Christof Beierle, Anne Canteaut, Gregor Leander, and Yann Rotella. Proving Resistance Against Invariant Attacks: How to Choose the Round Constants. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part II*, volume 10402 of *Lecture Notes in Computer Science*, pages 647–678. Springer, 2017.

[BDMW10] KA Browning, JF Dillon, MT McQuistan, and AJ Wolfe. An APN permutation in dimension six. *Finite Fields: theory and applications*, 518:33–42, 2010.

[BDP$^+$16] Guido Bertoni, Joan Daemen, Michaël Peeters, Gilles Van Assche, and Ronny Van Keer. CAESAR submission: Ketje v2. CAESAR candidate. http://competitions.cr.yp.to/round3/ketjev2.pdf, https://keccak.team/files/Ketjev2-doc2.0.pdf, 2016.

[BDPA06] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. RadioGatún, a belt-and-mill hash function. Cryptology ePrint Archive, Report 2006/369, 2006. https://eprint.iacr.org/2006/369.

[BDPA11] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Duplexing the Sponge: Single-Pass Authenticated Encryption and Other Applications. In Ali Miri and Serge Vaudenay, editors, *Selected Areas in Cryptography - 18th International Workshop, SAC 2011, Toronto, ON, Canada, August 11-12, 2011, Revised Selected Papers*, volume 7118 of *Lecture Notes in Computer Science*, pages 320–337. Springer, 2011.

[BDPVA07] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche. Sponge functions, May 2007. Ecrypt Hash Workshop 2007. Available at https://keccak.team/files/SpongeFunctions.pdf.

[BDPVA11] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche. The keccak reference, version 3.0, 2011. https://keccak.team/keccak.html.

[BDPVA12a] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Duplexing the Sponge: Single-Pass Authenticated

Encryption and Other Applications. In Ali Miri and Serge Vaudenay, editors, *Selected Areas in Cryptography: 18th International Workshop, SAC 2011, Toronto, ON, Canada, August 11-12, 2011, Revised Selected Papers*, pages 320–337, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[BDPVA12b] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Permutation-based encryption, authentication and authenticated encryption. *Directions in Authenticated Ciphers*, 2012.

[BDPVA12c] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Permutation-based encryption, authentication and authenticated encryption. Presented at DIAC 2012, 05–06 July 2012, Stockholm, Sweden., 2012.

[Bel11] Belarusian State University, National Research Center for Applied Problems of Mathematics and Informatics. Информационные технологии. Защита информации. Криптографические алгоритмы шифрования и контроля целостности [Information technologies. Information security. Cryptographic algorithms for encryption and integrity control.]. State Standard of Republic of Belarus (STB 34.101.31-2011), 2011. http://apmi.bsu.by/assets/files/std/belt-spec27.pdf.

[Bey18] Tim Beyne. Block Cipher Invariants as Eigenvectors of Correlation Matrices. In Thomas Peyrin and Steven D. Galbraith, editors, *Advances in Cryptology - ASIACRYPT 2018 - 24th International Conference on the Theory and Application of Cryptology and Information Security, Brisbane, QLD, Australia, December 2-6, 2018, Proceedings, Part I*, volume 11272 of *Lecture Notes in Computer Science*, pages 3–31. Springer, 2018.

[BGEC05] Olivier Billet, Henri Gilbert, and Charaf Ech-Chatbi. Cryptanalysis of a White Box AES Implementation. In Helena Handschuh and M. Anwar Hasan, editors, *Selected Areas in Cryptography: 11th International Workshop, SAC 2004, Waterloo, Canada, August 9-10, 2004, Revised Selected Papers*, pages 227–240, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[BHMT16] Joppe W. Bos, Charles Hubain, Wil Michiels, and Philippe Teuwen. Differential Computation Analysis: Hiding Your White-Box Designs is Not Enough. In Benedikt Gierlichs and Axel Y. Poschmann, editors, *Cryptographic Hardware and Embedded Systems - CHES 2016 - 18th International Conference, Santa Barbara, CA, USA, August 17-19, 2016, Proceedings*, volume 9813 of *Lecture Notes in Computer Science*, pages 215–236. Springer, 2016.

[Bir06] Alex Biryukov. The Design of a Stream Cipher LEX. In Eli Biham and Amr M. Youssef, editors, *Selected Areas in Cryptography, 13th International Workshop, SAC 2006, Montreal, Canada, August 17-18, 2006 Revised Selected Papers*, volume 4356 of *Lecture Notes in Computer Science*, pages 67–75. Springer, 2006.

[BKP16] Alex Biryukov, Dmitry Khovratovich, and Léo Perrin. Multiset-Algebraic Cryptanalysis of Reduced Kuznyechik, Khazad, and secret SPNs. *IACR Trans. Symmetric Cryptol.*, 2016(2):226–247, 2016.

[BKP17] Alex Biryukov, Dmitry Khovratovich, and Léo Perrin. Multiset-Algebraic Cryptanalysis of Reduced Kuznyechik, Khazad, and secret SPNs. *IACR Transactions on Symmetric Cryptology*, 2016(2):226–247, 2017.

[BLP15] Alex Biryukov, Gaëtan Leurent, and Léo Perrin. Cryptanalysis of Feistel Networks with Secret Round Functions. In Orr Dunkelman and Liam Keliher, editors, *Selected Areas in Cryptography - SAC 2015 - 22nd International Conference, Sackville, NB, Canada, August 12-14, 2015, Revised Selected Papers*, volume 9566 of *Lecture Notes in Computer Science*, pages 102–121. Springer, 2015.

[BM18] Leif Both and Alexander May. Decoding Linear Codes with High Error Rate and Its Impact for LPN Security. In Tanja Lange and Rainer Steinwandt, editors, *Post-Quantum Cryptography*, pages 25–46, Cham, 2018. Springer International Publishing.

[BN08] Mihir Bellare and Chanathip Namprempre. Authenticated Encryption: Relations among Notions and Analysis of the Generic Composition Paradigm. *J. Cryptology*, 21(4):469–491, 2008.

[BP15] Alex Biryukov and Léo Perrin. On Reverse-Engineering S-Boxes with Hidden Design Criteria or Structure. In Gennaro and Robshaw [GR15], pages 116–140.

[BPU16] Alex Biryukov, Léo Perrin, and Aleksei Udovenko. Reverse-Engineering the S-Box of Streebog, Kuznyechik and STRIBOBr1. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology - EUROCRYPT 2016 - 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part I*, volume 9665 of *Lecture Notes in Computer Science*, pages 372–402. Springer, 2016.

[BS01] Alex Biryukov and Adi Shamir. Structural Cryptanalysis of SASAS. In Birgit Pfitzmann, editor, *Advances in Cryptology – EUROCRYPT 2001*, volume 2045 of *Lecture Notes in Computer Science*, pages 395–405. Springer Berlin Heidelberg, 2001.

[BSS$^+$13] Ray Beaulieu, Douglas Shors, Jason Smith, Stefan Treatman-Clark, Bryan Weeks, and Louis Wingers. The SIMON and SPECK Families of Lightweight Block Ciphers. *IACR Cryptology ePrint Archive*, 2013:404, 2013.

[BU18a] Alex Biryukov and Aleksei Udovenko. Attacks and Countermeasures for White-box Designs. In Thomas Peyrin and Steven D. Galbraith, editors, *Advances in Cryptology - ASIACRYPT 2018 - 24th International Conference on the Theory and Application of Cryptology and Information Security, Brisbane, QLD, Australia, December 2-6, 2018, Proceedings, Part II*, volume 11273 of *Lecture Notes in Computer Science*, pages 373–402. Springer, 2018.

[BU18b] Alex Biryukov and Aleksei Udovenko. White-box Tools, 2018. https://github.com/cryptolu/whitebox.

[BUV17] Alex Biryukov, Aleksei Udovenko, and Vesselin Velichkov. Analysis of the NORX Core Permutation. *IACR Cryptology ePrint Archive*, 2017:34, 2017.

[BVLC16] Alex Biryukov, Vesselin Velichkov, and Yann Le Corre. Automatic Search for the Best Trails in ARX: Application to Block Cipher Speck. In Thomas Peyrin, editor, *Fast Software Encryption*, volume 3557 of *Lecture Notes in Computer Science*, page To Appear. Springer Berlin Heidelberg, 2016.

[Can05] D. Canright. A Very Compact S-Box for AES. In Josyula R. Rao and Berk Sunar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2005*, pages 441–455, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[Car07] Claude Carlet. Boolean Functions for Cryptography and Error Correcting Codes. In Yves Crama and Peter Hammer, editors, *Boolean Methods and Models*. Cambridge University Press, 2007.

[Car10a] Claude Carlet. *Boolean Functions for Cryptography and Error-Correcting Codes*, pages 257–397. Encyclopedia of Mathematics and its Applications. Cambridge University Press, 2010.

[Car10b] Claude Carlet. Vectorial Boolean functions for cryptography. *Boolean models and methods in mathematics, computer science, and engineering*, 134:398–469, 2010.

[CDNY18] Avik Chakraborti, Nilanjan Datta, Mridul Nandi, and Kan Yasuda. Beetle Family of Lightweight and Secure Authenticated Encryption Ciphers. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2018(2):218–241, 2018.

[CDP17] Anne Canteaut, Sébastien Duval, and Léo Perrin. A Generalisation of Dillon's APN Permutation With the Best Known Differential and Nonlinear Properties for All Fields of Size 2^{4k+2}. *IEEE Trans. Information Theory*, 63(11):7575–7591, 2017.

[CEJvO02a] Stanley Chow, Philip A. Eisen, Harold Johnson, and Paul C. van Oorschot. A White-Box DES Implementation for DRM Applications. In Joan Feigenbaum, editor, *Security and Privacy in Digital Rights Management, ACM CCS-9 Workshop, DRM 2002, Washington, DC, USA, November 18, 2002, Revised Papers*, volume 2696 of *Lecture Notes in Computer Science*, pages 1–15. Springer, 2002.

[CEJvO02b] Stanley Chow, Philip A. Eisen, Harold Johnson, and Paul C. van Oorschot. White-Box Cryptography and an AES Implementation. In Kaisa Nyberg and Howard M. Heys, editors, *Selected Areas in Cryptography, 9th Annual International Workshop, SAC 2002, St. John's, Newfoundland, Canada, August 15-16, 2002. Revised Papers*, volume 2595 of *Lecture Notes in Computer Science*, pages 250–270. Springer, 2002.

[CFG+17] Colin Chaigneau, Thomas Fuhr, Henri Gilbert, Jérémy Jean, and Jean-René Reinhard. Cryptanalysis of NORX v2.0. *IACR Trans. Symmetric Cryptol.*, 2017(1):156–174, 2017.

[CMR17] Brent Carmer, Alex J. Malozemoff, and Mariana Raykova. 5Gen-C: Multi-input Functional Encryption and Program Obfuscation for Arithmetic Circuits. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, pages 747–764, New York, NY, USA, 2017. ACM.

[Com19] The CAESAR Committee. Competition for Authenticated Encryption: Security, Applicability, and Robustness, 2014–2019. http://competitions.cr.yp.to/caesar-submissions.html.

[Cop94] Don Coppersmith. The Data Encryption Standard (DES) and its strength against attacks. *IBM Journal of Research and Development*, 38(3):243–250, 1994.

[Cou04] Nicolas Courtois. Feistel Schemes and Bi-linear Cryptanalysis. In Matthew K. Franklin, editor, *Advances in Cryptology - CRYPTO 2004, 24th Annual International CryptologyConference, Santa Barbara, California, USA, August 15-19, 2004, Proceedings*, volume 3152 of *Lecture Notes in Computer Science*, pages 23–40. Springer, 2004.

[CPT18] Anne Canteaut, Léo Perrin, and Shizhu Tian. If a Generalised Butterfly is APN then it Operates on 6 Bits. Cryptology ePrint Archive, Report 2018/1036, 2018. https://eprint.iacr.org/2018/1036.

[DBG+15] Daniel Dinu, Alex Biryukov, Johann Großschädl, Dmitry Khovratovich, Yann Le Corre, and Léo Perrin. FELICS - Fair Evaluation of Lightweight Cryptographic Systems. In *NIST Workshop on Lightweight Cryptography*, volume 128, 2015.

[DCK$^+$15] Daniel Dinu, Yann Le Corre, Dmitry Khovratovich, Léo Perrin, Johann Großschädl, and Alex Biryukov. Triathlon of Lightweight Block Ciphers for the Internet of Things. Cryptology ePrint Archive, Report 2015/209, 2015. https://eprint.iacr.org/2015/209.

[DCK$^+$16] Daniel Dinu, Yann Le Corre, Dmitry Khovratovich, Léo Perrin, Johann Großschädl, and Alex Biryukov. FELICS - Fair Evaluation of Lightweight Cryptographic Systems, 2016. https://www.cryptolux.org/index.php/FELICS.

[DDKL15] Itai Dinur, Orr Dunkelman, Thorsten Kranz, and Gregor Leander. Decomposing the ASASA Block Cipher Construction. Cryptology ePrint Archive, Report 2015/507, 2015. http://eprint.iacr.org/.

[DDKS15] Itai Dinur, Orr Dunkelman, Nathan Keller, and Adi Shamir. New Attacks on Feistel Structures with Improved Memory Complexities. In Gennaro and Robshaw [GR15], pages 433–454.

[DEMS16] Christoph Dobraunig, Maria Eichlseder, Florian Mendel, and Martin Schläffer. Ascon v1.2. CAESAR candidate. http://competitions.cr.yp.to/round3/asconv12.pdf, 2016.

[Dil09] John F Dillon. APN polynomials: an update. In *Fq9, The 9th International Conference on Finite Fields and Applications*, 2009.

[Din18] Itai Dinur. An Improved Affine Equivalence Algorithm for Random Permutations. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology - EUROCRYPT 2018 - 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29 - May 3, 2018 Proceedings, Part I*, volume 10820 of *Lecture Notes in Computer Science*, pages 413–442. Springer, 2018.

[DLPR13] Cécile Delerablée, Tancrède Lepoint, Pascal Paillier, and Matthieu Rivain. White-Box Security Notions for Symmetric Encryption Schemes. In Tanja Lange, Kristin E. Lauter, and Petr Lisonek, editors, *Selected Areas in Cryptography - SAC 2013 - 20th International Conference, Burnaby, BC, Canada, August 14-16, 2013, Revised Selected Papers*, volume 8282 of *Lecture Notes in Computer Science*, pages 247–264. Springer, 2013.

[DMWP10] Yoni De Mulder, Brecht Wyseur, and Bart Preneel. Cryptanalysis of a Perturbated White-Box AES Implementation. In Guang Gong and Kishan Chand Gupta, editors, *Progress in Cryptology - INDOCRYPT 2010: 11th International Conference on Cryptology in India, Hyderabad, India, December 12-15, 2010. Proceedings*, pages 292–310, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[DN16] Nilanjan Datta and Mridul Nandi. Proposal of ELmD v2.1. CAESAR candidate. `http://competitions.cr.yp.to/round2/elmdv21.pdf`, 2016.

[DPU$^+$16] Daniel Dinu, Léo Perrin, Aleksei Udovenko, Vesselin Velichkov, Johann Großschädl, and Alex Biryukov. Design Strategies for ARX with Provable Bounds: Sparx and LAX. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *Advances in Cryptology - ASIACRYPT 2016 - 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part I*, volume 10031 of *Lecture Notes in Computer Science*, pages 484–513, 2016.

[DR98] Joan Daemen and Vincent Rijmen. The Block Cipher Rijndael. In Jean-Jacques Quisquater and Bruce Schneier, editors, *Smart Card Research and Applications, This International Conference, CARDIS '98, Louvain-la-Neuve, Belgium, September 14-16, 1998, Proceedings*, volume 1820 of *Lecture Notes in Computer Science*, pages 277–284. Springer, 1998.

[DR02] Joan Daemen and Vincent Rijmen. AES and the Wide Trail Design Strategy. In Lars R. Knudsen, editor, *Advances in Cryptology - EUROCRYPT 2002, International Conference on the Theory and Applications of Cryptographic Techniques, Amsterdam, The Netherlands, April 28 - May 2, 2002, Proceedings*, volume 2332 of *Lecture Notes in Computer Science*, pages 108–109. Springer, 2002.

[DV18] F. Betül Durak and Serge Vaudenay. Generic Round-Function-Recovery Attacks for Feistel Networks over Small Domains. In Bart Preneel and Frederik Vercauteren, editors, *Applied Cryptography and Network Security - 16th International Conference, ACNS 2018, Leuven, Belgium, July 2-4, 2018, Proceedings*, volume 10892 of *Lecture Notes in Computer Science*, pages 440–458. Springer, 2018.

[ECR17] ECRYPT-CSA consortium. CHES 2017 Capture The Flag Challenge. The WhibOx Contest, 2017. `http://whibox.cr.yp.to/`.

[EKM17] Andre Esser, Robert Kübler, and Alexander May. LPN Decoded. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology – CRYPTO 2017*, pages 486–514, Cham, 2017. Springer International Publishing.

[Fed12] Federal Agency on Technical Regulation and Metrology. GOST R 34.11-2012: Streebog Hash Function, 2012. `https://www.streebog.net/`.

[FFW17] Shihui Fu, Xiutao Feng, and Baofeng Wu. Differentially 4-Uniform Permutations with the Best Known Nonlinearity from

Butterflies. *IACR Trans. Symmetric Cryptol.*, 2017(2):228–249, 2017.

[FJVP13] P. J. S. G. Ferreira, B. Jesus, J. Vieira, and A. J. Pinho. The Rank of Random Binary Matrices and Distributed Storage Applications. *IEEE Communications Letters*, 17(1):151–154, January 2013.

[GGH$^+$13] S. Garg, C. Gentry, S. Halevi, M. Raykova, A. Sahai, and B. Waters. Candidate Indistinguishability Obfuscation and Functional Encryption for all Circuits. In *2013 IEEE 54th Annual Symposium on Foundations of Computer Science*, pages 40–49, Oct 2013.

[GLSV14] Vincent Grosso, Gaëtan Leurent, François-Xavier Standaert, and Kerem Varici. LS-Designs: Bitslice Encryption for Efficient Masked Software Implementations. In Carlos Cid and Christian Rechberger, editors, *Fast Software Encryption - 21st International Workshop, FSE 2014, London, UK, March 3-5, 2014. Revised Selected Papers*, volume 8540 of *Lecture Notes in Computer Science*, pages 18–37. Springer, 2014.

[Göl15] Faruk Göloglu. Almost perfect nonlinear trinomials and hexanomials. *Finite Fields and Their Applications*, 33:258–282, 2015.

[GPRW17] Louis Goubin, Pascal Paillier, Matthieu Rivain, and Junwei Wang. Reveal Secrets in Adoring Poitras. A victory of reverse engineering and cryptanalysis over challenge 777, 2017. CHES 2017 Rump Session, slides. https://ches.2017.rump.cr.yp.to/a905c99d1845f2cf373aad564ac7b5e4.pdf.

[GPRW18] Louis Goubin, Pascal Paillier, Matthieu Rivain, and Junwei Wang. How to Reveal the Secrets of an Obscure White-Box Implementation. Cryptology ePrint Archive, Report 2018/098, 2018. https://eprint.iacr.org/2018/098.

[GPT15] Henri Gilbert, Jérôme Plût, and Joana Treger. Key-Recovery Attack on the ASASA Cryptosystem with Expanding S-Boxes. In Rosario Gennaro and Matthew Robshaw, editors, *Advances in Cryptology – CRYPTO 2015: 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part I*, pages 475–490, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg.

[GR15] Rosario Gennaro and Matthew Robshaw, editors. *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part I*, volume 9215 of *Lecture Notes in Computer Science*. Springer, 2015.

[Gra18] Lorenzo Grassi. MixColumns Properties and Attacks on (Round-Reduced) AES with a Single Secret S-Box. In Nigel P. Smart, editor, *Topics in Cryptology - CT-RSA 2018 - The Cryptographers' Track at the RSA Conference 2018, San Francisco, CA, USA, April 16-20, 2018, Proceedings*, volume 10808 of *Lecture Notes in Computer Science*, pages 243–263. Springer, 2018.

[GW18] Chun Guo and Lei Wang. Revisiting Key-Alternating Feistel Ciphers for Shorter Keys and Multi-user Security. In Thomas Peyrin and Steven D. Galbraith, editors, *Advances in Cryptology - ASIACRYPT 2018 - 24th International Conference on the Theory and Application of Cryptology and Information Security, Brisbane, QLD, Australia, December 2-6, 2018, Proceedings, Part I*, volume 11272 of *Lecture Notes in Computer Science*, pages 213–243. Springer, 2018.

[HBE+16] Charles Hubain, Joppe Bos, Michael Eder, Paul Bottinelli, Philippe Teuwen, Van Huynh Le, and Wil Michiels. Side-Channel Marvels, 2016. https://github.com/SideChannelMarvels.

[Hir16] Shoichi Hirose. Sequential hashing with minimum padding. In *NIST Workshop on Lightweight Cryptography 2016*. National Institute of Standards and Technology (NIST), 2016.

[HKM95] Carlo Harpes, Gerhard G. Kramer, and James L. Massey. A Generalization of Linear Cryptanalysis and the Applicability of Matsui's Piling-Up Lemma. In Louis C. Guillou and Jean-Jacques Quisquater, editors, *Advances in Cryptology - EUROCRYPT '95, International Conference on the Theory and Application of Cryptographic Techniques, Saint-Malo, France, May 21-25, 1995, Proceeding*, volume 921 of *Lecture Notes in Computer Science*, pages 24–38. Springer, 1995.

[HLK+13] Deukjo Hong, Jung-Keun Lee, Dong-Chan Kim, Daesung Kwon, Kwon Ho Ryu, and Donggeon Lee. LEA: A 128-Bit Block Cipher for Fast Encryption on Common Processors. In Yongdae Kim, Heejo Lee, and Adrian Perrig, editors, *Information Security Applications - 14th International Workshop, WISA 2013, Jeju Island, Korea, August 19-21, 2013, Revised Selected Papers*, volume 8267 of *Lecture Notes in Computer Science*, pages 3–27. Springer, 2013.

[HSH+06] Deukjo Hong, Jaechul Sung, Seokhie Hong, Jongin Lim, Sangjin Lee, Bonseok Koo, Changhoon Lee, Donghoon Chang, Jesang Lee, Kitae Jeong, Hyun Kim, Jongsung Kim, and Seongtaek Chee. HIGHT: A New Block Cipher Suitable for Low-Resource Device. In Louis Goubin and Mitsuru Matsui, editors, *Cryptographic Hardware and Embedded Systems - CHES 2006, 8th International Workshop, Yokohama, Japan, October 10-13, 2006,* *Proceedings*, volume 4249 of *Lecture Notes in Computer Science*, pages 46–59. Springer, 2006.

[HSS99] A.S. Hedayat, N.J.A. Sloane, and J. Stufken. *Orthogonal Arrays*. Springer Series in Statistics. Springer New York, 1999.

[HW78] A Hedayat and WD Wallis. Hadamard Matrices and Their Applications. *The Annals of Statistics*, 6(6):1184–1238, 1978.

[IPSW06] Yuval Ishai, Manoj Prabhakaran, Amit Sahai, and David Wagner. Private Circuits II: Keeping Secrets in Tamperable Circuits. In Serge Vaudenay, editor, *Advances in Cryptology - EUROCRYPT 2006: 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, St. Petersburg, Russia, May 28 - June 1, 2006. Proceedings*, pages 308–327, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[ISW03] Yuval Ishai, Amit Sahai, and David Wagner. Private Circuits: Securing Hardware against Probing Attacks. In Dan Boneh, editor, *Advances in Cryptology - CRYPTO 2003: 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003. Proceedings*, pages 463–481, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[Jea16] Jérémy Jean. TikZ for Cryptographers. https://www.iacr.org/authors/tikz/, 2016.

[Jé16] Jérémy Jean, Ivica Nikolić, Thomas Peyrin and Yannick Seurin. Deoxys v1.41. CAESAR candidate. http://competitions.cr.yp.to/round3/deoxysv141.pdf, 2016.

[Kar10] Mohamed Karroumi. Protecting White-Box AES with Dual Ciphers. In Kyung Hyune Rhee and DaeHun Nyang, editors, *Information Security and Cryptology - ICISC 2010 - 13th International Conference, Seoul, Korea, December 1-3, 2010, Revised Selected Papers*, volume 6829 of *Lecture Notes in Computer Science*, pages 278–291. Springer, 2010.

[Knu98] Lars Ramkilde Knudsen. DEAL-A 128-bit Block Cipher, AES submission, 1998.

[KR96] Lars R. Knudsen and Matthew J. B. Robshaw. Non-Linear Approximations in Linear Cryptanalysis. In Ueli M. Maurer, editor, *Advances in Cryptology - EUROCRYPT '96, International Conference on the Theory and Application of Cryptographic Techniques, Saragossa, Spain, May 12-16, 1996, Proceeding*, volume 1070 of *Lecture Notes in Computer Science*, pages 224–236. Springer, 1996.

[KR16] Ted Krovetz and Philip Rogaway. OCB (v1.1). CAESAR candidate. http://competitions.cr.yp.to/round3/ocbv11.pdf, 2016.

[KT70] T. Kasami and N. Tokura. On the weight structure of Reed-Muller codes. *IEEE Transactions on Information Theory*, 16(6):752–759, November 1970.

[KTA76] Tadao Kasami, Nobuki Tokura, and Saburo Azumi. On the weight enumeration of weights less than 2.5d of Reed—Muller codes. *Information and Control*, 30(4):380 – 395, 1976.

[LAAZ11] Gregor Leander, Mohamed Ahmed Abdelraheem, Hoda AlKhzaimi, and Erik Zenner. A Cryptanalysis of PRINTcipher: The Invariant Subspace Attack. In Phillip Rogaway, editor, *Advances in Cryptology - CRYPTO 2011 - 31st Annual Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2011. Proceedings*, volume 6841 of *Lecture Notes in Computer Science*, pages 206–221. Springer, 2011.

[Lai94] Xuejia Lai. Higher Order Derivatives and Differential Cryptanalysis. In Richard E. Blahut, Daniel J. Costello, Ueli Maurer, and Thomas Mittelholzer, editors, *Communications and Cryptography: Two Sides of One Tapestry*, pages 227–233. Springer US, Boston, MA, 1994.

[Lec71] Robert J. Lechner. Harmonic Analysis of Switching Functions. In Amar Mukhopadhyay, editor, *Recent Developments in Switching Theory*, Academic Press electrical science series. New York: Academic Press, 1971.

[LR88] Michael Luby and Charles Rackoff. How to Construct Pseudorandom Permutations from Pseudorandom Functions. *SIAM J. Comput.*, 17:373–386, 1988.

[LR13] Tancrède Lepoint and Matthieu Rivain. Another Nail in the Coffin of White-Box AES Implementations. Cryptology ePrint Archive, Report 2013/455, 2013. http://eprint.iacr.org/2013/455.

[LRW02] Moses Liskov, Ronald L. Rivest, and David A. Wagner. Tweakable Block Ciphers. In Moti Yung, editor, *Advances in Cryptology - CRYPTO 2002, 22nd Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 2002, Proceedings*, volume 2442 of *Lecture Notes in Computer Science*, pages 31–46. Springer, 2002.

[LS14] Rodolphe Lampe and Yannick Seurin. Security Analysis of Key-Alternating Feistel Ciphers. In Carlos Cid and Christian Rechberger, editors, *Fast Software Encryption - 21st International*

Workshop, FSE 2014, London, UK, March 3-5, 2014. Revised Selected Papers, volume 8540 of *Lecture Notes in Computer Science*, pages 243–264. Springer, 2014.

[LTYW18] Yongqiang Li, Shizhu Tian, Yuyin Yu, and Mingsheng Wang. On the Generalization of Butterfly Structure. *IACR Trans. Symmetric Cryptol.*, 2018(1):160–179, 2018.

[LW14] Yongqiang Li and Mingsheng Wang. Constructing S-boxes for Lightweight Cryptography with Feistel Structure. 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 127–146. Springer, 2014.

[Mat93] Mitsuru Matsui. Linear Cryptanalysis Method for DES Cipher. In Tor Helleseth, editor, *Advances in Cryptology - EUROCRYPT '93, Workshop on the Theory and Application of of Cryptographic Techniques, Lofthus, Norway, May 23-27, 1993, Proceedings*, volume 765 of *Lecture Notes in Computer Science*, pages 386–397. Springer, 1993.

[Mat94] Mitsuru Matsui. On Correlation Between the Order of S-boxes and the Strength of DES. In Alfredo De Santis, editor, *Advances in Cryptology - EUROCRYPT '94, Workshop on the Theory and Application of Cryptographic Techniques, Perugia, Italy, May 9-12, 1994, Proceedings*, volume 950 of *Lecture Notes in Computer Science*, pages 366–375. Springer, 1994.

[Mat97] Mitsuru Matsui. New Block Encryption Algorithm MISTY. In Eli Biham, editor, *Fast Software Encryption, 4th International Workshop, FSE '97, Haifa, Israel, January 20-22, 1997, Proceedings*, volume 1267 of *Lecture Notes in Computer Science*, pages 54–68. Springer, 1997.

[MDFK15] Brice Minaud, Patrick Derbez, Pierre-Alain Fouque, and Pierre Karpman. Key-Recovery Attacks on ASASA. In Tetsu Iwata and Jung Hee Cheon, editors, *Advances in Cryptology - ASIACRYPT 2015*, volume 8270 of *Lecture Notes in Computer Science*, page To appear. Springer Berlin Heidelberg, 2015.

[MGH09] Wil Michiels, Paul Gorissen, and Henk D. L. Hollmann. Cryptanalysis of a Generic Class of White-Box Implementations. In Roberto Maria Avanzi, Liam Keliher, and Francesco Sica, editors, *Selected Areas in Cryptography: 15th International Workshop, SAC 2008, Sackville, New Brunswick, Canada, August 14-15, Revised Selected Papers*, pages 414–428, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[MMH$^+$14] Nicky Mouha, Bart Mennink, Anthony Van Herrewege, Dai Watanabe, Bart Preneel, and Ingrid Verbauwhede. Chaskey: An Efficient MAC Algorithm for 32-bit Microcontrollers. In Antoine Joux and Amr M. Youssef, editors, *Selected Areas in Cryptography - SAC 2014 - 21st International Conference, Montreal, QC, Canada, August 14-15, 2014, Revised Selected Papers*, volume 8781 of *Lecture Notes in Computer Science*, pages 306–323. Springer, 2014.

[MPC04] Willi Meier, Enes Pasalic, and Claude Carlet. Algebraic Attacks and Decomposition of Boolean Functions. In Christian Cachin and Jan Camenisch, editors, *Advances in Cryptology - EUROCRYPT 2004, International Conference on the Theory and Applications of Cryptographic Techniques, Interlaken, Switzerland, May 2-6, 2004, Proceedings*, volume 3027 of *Lecture Notes in Computer Science*, pages 474–491. Springer, 2004.

[MV04] D. McGrew and J. Viega. The Galois/Counter Mode of Operation (GCM), 2004. Submission to NIST Modes of Operation Process.

[NIS12] NIST. SHA-3 Competition, 2007-2012. `http://csrc.nist.gov/groups/ST/hash/sha3/index.html`.

[NIS19] NIST. Lightweight Cryptography. Call for Algorithms. `https://csrc.nist.gov/Projects/Lightweight-Cryptography`, 2019.

[NL18] Yoav Nir and Adam Langley. ChaCha20 and Poly1305 for IETF Protocols. *RFC*, 8439:1–46, 2018.

[Nyb93] Kaisa Nyberg. Differentially Uniform Mappings for Cryptography. In Tor Helleseth, editor, *Advances in Cryptology - EUROCRYPT '93, Workshop on the Theory and Application of of Cryptographic Techniques, Lofthus, Norway, May 23-27, 1993, Proceedings*, volume 765 of *Lecture Notes in Computer Science*, pages 55–64. Springer, 1993.

[Pat01] Jacques Patarin. Generic Attacks on Feistel Schemes. In Colin Boyd, editor, *Advances in Cryptology - ASIACRYPT 2001, 7th International Conference on the Theory and Application of Cryptology and Information Security, Gold Coast, Australia, December 9-13, 2001, Proceedings*, volume 2248 of *Lecture Notes in Computer Science*, pages 222–238. Springer, 2001.

[Pat04] Jacques Patarin. Security of Random Feistel Schemes with 5 or More Rounds. In Matthew K. Franklin, editor, *Advances in Cryptology - CRYPTO 2004, 24th Annual International CryptologyConference, Santa Barbara, California, USA, August 15-19, 2004, Proceedings*, volume 3152 of *Lecture Notes in Computer Science*, pages 106–122. Springer, 2004.

[Per19] Léo Perrin. Partitions in the S-Box of Streebog and Kuznyechik. *IACR Transactions on Symmetric Cryptology*, 2019(1), Mar. 2019. To appear.

[PG97] Jacques Patarin and Louis Goubin. Asymmetric cryptography with S-Boxes. In Yongfei Han, Tatsuaki Okamoto, and Sihan Qing, editors, *Information and Communication Security, First International Conference, ICICS'97, Beijing, China, November 11-14, 1997, Proceedings*, volume 1334 of *Lecture Notes in Computer Science*, pages 369–380. Springer, 1997.

[PRV06] Kevin T. Phelps, Josep Rifà, and Mercè Villanueva. Hadamard Codes of Length 2ts (s Odd). Rank and Kernel. In Marc P. C. Fossorier, Hideki Imai, Shu Lin, and Alain Poli, editors, *Applied Algebra, Algebraic Algorithms and Error-Correcting Codes*, pages 328–337, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[PS16] Thomas Peyrin and Yannick Seurin. Counter-in-Tweak: Authenticated Encryption Modes for Tweakable Block Ciphers. 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 I*, volume 9814 of *Lecture Notes in Computer Science*, pages 33–63. Springer, 2016.

[PU16] Léo Perrin and Aleksei Udovenko. Algebraic Insights into the Secret Feistel Network. In Thomas Peyrin, editor, *Fast Software Encryption - 23rd International Conference, FSE 2016, Bochum, Germany, March 20-23, 2016, Revised Selected Papers*, volume 9783 of *Lecture Notes in Computer Science*, pages 378–398. Springer, 2016.

[PU17] Léo Perrin and Aleksei Udovenko. Exponential S-Boxes: a Link Between the S-Boxes of BelT and Kuznyechik/Streebog. *IACR Transactions on Symmetric Cryptology*, 2016(2):99–124, Feb. 2017.

[PUB16] Léo Perrin, Aleksei Udovenko, and Alex Biryukov. Cryptanalysis of a Theorem: Decomposing the Only Known Solution to the Big APN Problem. 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 93–122. Springer, 2016.

[QSLG17] Kexin Qiao, Ling Song, Meicheng Liu, and Jian Guo. New Collision Attacks on Round-Reduced Keccak. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *Advances in Cryptology - EUROCRYPT 2017 - 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Paris,* *France, April 30 - May 4, 2017, Proceedings, Part III*, volume 10212 of *Lecture Notes in Computer Science*, pages 216–243, 2017.

[RBBK01] Phillip Rogaway, Mihir Bellare, John Black, and Ted Krovetz. OCB: a block-cipher mode of operation for efficient authenticated encryption. In Michael K. Reiter and Pierangela Samarati, editors, *CCS 2001, Proceedings of the 8th ACM Conference on Computer and Communications Security, Philadelphia, Pennsylvania, USA, November 6-8, 2001.*, pages 196–205. ACM, 2001.

[RP97] Vincent Rijmen and Bart Preneel. A Family of Trapdoor Ciphers. In Eli Biham, editor, *Fast Software Encryption, 4th International Workshop, FSE '97, Haifa, Israel, January 20-22, 1997, Proceedings*, volume 1267 of *Lecture Notes in Computer Science*, pages 139–148. Springer, 1997.

[RR13] Matthieu Rivain and Thomas Roche. SCARE of Secret Ciphers with SPN Structures. In Kazue Sako and Palash Sarkar, editors, *Advances in Cryptology - ASIACRYPT 2013 - 19th International Conference on the Theory and Application of Cryptology and Information Security, Bengaluru, India, December 1-5, 2013, Proceedings, Part I*, volume 8269 of *Lecture Notes in Computer Science*, pages 526–544. Springer, 2013.

[RW19] Matthieu Rivain and Junwei Wang. Analysis and Improvement of Differential Computation Attacks against Internally-Encoded White-Box Implementations. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2019(2), 2019. To appear.

[Saa14] Markku-Juhani O. Saarinen. STRIBOB: Authenticated Encryption from GOST R 34.11-2012 LPS Permutation. *IACR Cryptology ePrint Archive*, 2014:271, 2014.

[Sar14] Palash Sarkar. Modes of operations for encryption and authentication using stream ciphers supporting an initialisation vector. *Cryptography and Communications*, 6(3):189–231, 2014.

[SD19] The Sage Developers. *SageMath, the Sage Mathematics Software System (Version 8.6)*, 2019. `https://www.sagemath.org`.

[Sha49] Claude E Shannon. Communication theory of secrecy systems. *Bell system technical journal*, 28(4):656–715, 1949.

[SLG17] Ling Song, Guohong Liao, and Jian Guo. Non-full Sbox Linearization: Applications to Collision Attacks on Round-Reduced Keccak. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part II*, volume 10402 of *Lecture Notes in Computer Science*, pages 428–451. Springer, 2017.

[SMG16] Pascal Sasdrich, Amir Moradi, and Tim Güneysu. White-Box Cryptography in the Gray Box. In *Revised Selected Papers of the 23rd International Conference on Fast Software Encryption - Volume 9783*, FSE 2016, pages 185–203, New York, NY, USA, 2016. Springer-Verlag New York, Inc.

[Sor84] Arthur Sorkin. Lucifer, a Cryptographic Algorithm. *Cryptologia*, 8(1):22–42, 1984.

[SWP09] Amitabh Saxena, Brecht Wyseur, and Bart Preneel. Towards Security Notions for White-Box Cryptography. In Pierangela Samarati, Moti Yung, Fabio Martinelli, and Claudio Agostino Ardagna, editors, *Information Security, 12th International Conference, ISC 2009, Pisa, Italy, September 7-9, 2009. Proceedings*, volume 5735 of *Lecture Notes in Computer Science*, pages 49–58. Springer, 2009.

[SWW16] Ling Sun, Wei Wang, and Meiqin Wang. MILP-Aided Bit-Based Division Property for Primitives with Non-Bit-Permutation Linear Layers. Cryptology ePrint Archive, Report 2016/811, 2016. `https://eprint.iacr.org/2016/811`.

[TCT15] Deng Tang, Claude Carlet, and Xiaohu Tang. Differentially 4-uniform bijections by permuting the inverse function. *Des. Codes Cryptography*, 77(1):117–141, 2015.

[TKKL15] Tyge Tiessen, Lars R. Knudsen, Stefan Kölbl, and Martin M. Lauridsen. Security of the AES with a Secret S-Box. In Gregor Leander, editor, *Fast Software Encryption - 22nd International Workshop, FSE 2015, Istanbul, Turkey, March 8-11, 2015, Revised Selected Papers*, volume 9054 of *Lecture Notes in Computer Science*, pages 175–189. Springer, 2015.

[TLS16] Yosuke Todo, Gregor Leander, and Yu Sasaki. Nonlinear Invariant Attack - Practical Attack on Full SCREAM, iSCREAM, and Midori64. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *Advances in Cryptology - ASIACRYPT 2016 - 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part II*, volume 10032 of *Lecture Notes in Computer Science*, pages 3–33, 2016.

[TM16] Yosuke Todo and Masakatu Morii. Bit-Based Division Property and Application to Simon Family. In Thomas Peyrin, editor, *Fast Software Encryption - 23rd International Conference, FSE 2016, Bochum, Germany, March 20-23, 2016, Revised Selected Papers*, volume 9783 of *Lecture Notes in Computer Science*, pages 357–377. Springer, 2016.

[Tod15] Yosuke Todo. Structural Evaluation by Generalized Integral Property. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I*, volume 9056 of *Lecture Notes in Computer Science*, pages 287–314. Springer, 2015.

[W$^+$08] Matthijs Joost Warrens et al. *Similarity coefficients for binary data: properties of coefficients, coefficient matrices, multi-way metrics and multivariate coefficients.* Psychometrics and Research Methodology Group, Leiden University Institute for Psychological Research, Faculty of Social Sciences, Leiden University, 2008.

[WP16] Hongjun Wu and Bart Preneel. AEGIS: A Fast Authenticated Encryption Algorithm (v1.1). CAESAR candidate. `http://competitions.cr.yp.to/round3/aegisv11.pdf`, 2016.

[Wu16] Hongjun Wu. ACORN: A Lightweight Authenticated Cipher (v3), 2016. A CAESAR candidate.

[WYWP18] Yongzhuang Wei, Tao Ye, Wenling Wu, and Enes Pasalic. Generalized Nonlinear Invariant Attack and a New Design Criterion for Round Constants. *IACR Trans. Symmetric Cryptol.*, 2018(4):62–79, 2018.

[XL09] Y. Xiao and X. Lai. A Secure Implementation of White-Box AES. In *2009 2nd International Conference on Computer Science and its Applications*, pages 1–6, Dec 2009.

[ZBL$^+$14] Wentao Zhang, Zhenzhen Bao, Dongdai Lin, Vincent Rijmen, Bohan Yang, and Ingrid Verbauwhede. RECTANGLE: A Bit-slice Ultra-Lightweight Block Cipher Suitable for Multiple Platforms. *IACR Cryptology ePrint Archive*, 2014:84, 2014.

[ZR17] Wenying Zhang and Vincent Rijmen. Division Cryptanalysis of Block Ciphers with a Binary Diffusion Layer. Cryptology ePrint Archive, Report 2017/188, 2017. `https://eprint.iacr.org/2017/188`.