RUHR
UNIVERSITÄT
BOCHUM

Design and Analysis of Block Ciphers

Doctoral Dissertation

Thorsten Kranz

July 2020

1st Reviewer:	Prof. Dr. Gregor Leander
2nd Reviewer:	Prof. Dr. Alexander May

Ruhr University Bochum
Faculty of Mathematics

Abstract

In information security systems, block ciphers play a crucial role. They are deployed to protect confidential data and as building blocks in many cryptographic designs. This thesis focuses on the design and analysis of block ciphers.

After motivating the topic and introducing the basics, the thesis is divided into three parts. The first part is a systematization of knowledge of linear cryptanalysis and provides new insights in the areas of key schedule design and tweakable block ciphers. The second part is an analysis of the ASASA structure, which was proposed as a building block for several cryptographic designs. Finally, the third part discusses methods for finding efficient implementations of matrix multiplications and using them to construct building blocks for secure lightweight block ciphers.

Zusammenfassung

In informationssichernden Systemen spielen Blockchiffren eine entscheidende Rolle. Sie werden eingesetzt um vertrauliche Daten zu schützen und als Baustein in vielen kryptographischen Konstruktionen verwendet. Diese Arbeit legt das Augenmerk auf den Entwurf und die Analyse von Blockchiffren.

Nach einer Motivation des Themas und einer Einführung in die Grundlagen teilt sich diese Arbeit in drei Teile. Der erste Teil systematisiert das Wissen über lineare Kryptanalyse und gibt neue Einblicke in den Entwurf von Schlüsselableitungsfunktionen und in das Thema der veränderbaren Blockchiffren. Der zweite Teil ist eine Analyse der ASASA Struktur, welche als Baustein für verschiedene kryptographische Entwürfe vorgeschlagen wurde. Zuletzt behandelt der dritte Teil Methoden, um effiziente Implementierung von Matrix Multiplikationen zu finden und diese zu nutzen, um Bausteine für sichere leichtgewichtige Blockchiffren zu konstruieren.

Acknowledgments

First, I want to thank my co-authors Christof Beierle, Itai Dinur, Orr Dunkelman, Gregor Leander, Ko Stoffelen, and Friedrich Wiemer. It was a great pleasure to work with such extraordinary scientists.

I am deeply grateful to my supervisor Gregor Leander. I always felt his amazing support and motivation. By working with him, I learned a lot about what constitutes a good scientist, a good teacher and a good supervisor. I also want to thank Christof Paar for accepting me as his student in my first year before I followed Gregor to the mathematics department.

I owe special thanks to Marion Reinhardt-Kalender and Irmgard Kühn, who helped me so many times with administrative problems and who have a major contribution to the great atmosphere in their groups. Many thanks go to Christof Beierle and Friedrich Wiemer who are not just former colleagues but real friends to me. Also, I want to say thank you to all the other former colleagues from the Horst Görtz Institute for IT-Security. I want to thank Thomas Peyrin for accepting me as a visiting student in his group at Nanyang Technological University in Singapore and I want to thank Siang Meng Sim for spending much of his time helping me getting along in Singapore. These three months were an incredible experience for me.

I would like to thank Anne Canteaut and Kaisa Nyberg for interesting discussions and valuable comments on the topics of Chapter 2. Regarding Chapter 4, I would like to thank Thomas Peyrin for some valuable discussions on the notion of the XOR count. I would also like to thank Joan Boyar, René Peralta, Chiara Schiavo, and Andrea Visconti for valuable comments on implementations and other practical details of their heuristics in Chapter 5. Furthermore, my work was partly supported by the DFG Research Training Group GRK 1817 Ubicrypt and by the BMBF Project UNIKOPS (01BY1040).

I am highly grateful for the support and love of my family, especially of my wife Sandra, my children Charlotte and Phil, my parents Helga and Ingomar and my brothers Axel, Ingomar, and Uwe.

Thorsten Kranz

Reken, July 2020

Notation

In the following, we present a list of symbols and a list of abbreviations. The list of symbols contains the most important symbols that occur throughout the whole thesis. More specific ones will be introduced in the text as needed.

List of Symbols

E_k	A block cipher.
D_k	The inverse of a block cipher.
m,x	A plaintext.
c,y	A ciphertext.
n	The block size of a block cipher.
k	The key of a block cipher.
κ	The key length.
S	An S-box, i. e. a non-linear bijection.
m	The size of an S-box.
t	The number of S-boxes. / The tweak of a tweakable block cipher.
τ	The tweak length.
L	A linear map.
L^T	The adjoint of a linear map L.
$\mathbb{F}_\ell$	The finite field with ℓ elements.
$\mathbb{F}_\ell^n$	The n-dimensional vector space over $\mathbb{F}_\ell$.
$\langle x, y\rangle$	The canonical scalar product of two vectors, i. e. $\sum x_i y_i$.
$x\|y$	Concatenation of the binary strings x and y.
$\mathrm{hw}(\cdot)$	The number of non-zero entries of a matrix or vector. In the case of a polynomial, the number of non-zero coefficients.
$x \oplus y$	Bitwise exclusive-or of x and y.

List of Abbreviations

AES	Advanced Encryption Standard
ARK	AddRoundKey
ARX	Addition-Rotation-XOR
CCA	Chosen-ciphertext attack
CDF	Cumulative distribution function
CPA	Chosen-plaintext attack
DDT	Difference distribution table
DES	Data Encryption Standard
DRM	Digital rights management
ECB	Electronic Codebook
FPGA	Field Programmable Gate Array
HSM	Hardware Security Module
IoT	Internet of Things
IV	Initialization vector
KPA	Known-plaintext attack
LAT	Linear approximation table
MC	MixColumns
MDS	Maximum Distance Separable
NIST	National Institute of Standards and Technology
PRP	Pseudorandom permutation
SB	SubBytes
SPN	Substitution-permutation network
SR	ShiftRows
XOR	Exclusive-or

Contents

1 Introduction

Today, information security is of major importance for many aspects of our everyday life. For example, it is fundamental for the privacy of individuals, for business success of companies, or for preventing crimes. The study of mathematical techniques related to aspects of information security is called cryptography or cryptology[1] [MVO96, Definition 1.1].

Cryptology offers a wide range of tools for enabling secure communication and data storage. Typical examples are encryption algorithms that yield confidentiality or digital signatures that can be used to assure data integrity or user authentication. Such basic tools are called cryptographic primitives.

The field of cryptography is typically divided into two categories: asymmetric cryptography and symmetric cryptography. This categorization is best described by a short example of how encryption works in each category for a sender who wants to send a message to a recipient via an insecure channel. In symmetric cryptography, the sender uses a secret key to encrypt a message. The recipient then needs the same key to decrypt the encrypted message. Whoever knows the key can break the confidentiality. Thus, the obvious problem is that both the sender and the recipient first need to agree on some secret key, but they do not have a secure channel to do that. In asymmetric cryptography, a recipient of a message has a public key and a private/secret key. As the name suggests, the public key is made public and the secret key is kept secret. The sender can use the public key to encrypt the message, and the recipient needs the private key to decrypt the encrypted message. Whoever knows the private key can read the message. So the problem from the symmetric scenario is solved, as the public key may be disseminated via an insecure channel. However, a new problem arises. Namely the question of authenticity. The sender must be sure that the public key really belongs to the intended recipient. This problem is addressed

[1]In a strict sense, cryptology is a generic term for cryptography and cryptanalysis. The former denotes the design and latter the analysis. However, the terms cryptology and cryptography are often used interchangeably in the literature.

by so-called certificates that we will not discuss in this thesis. While asymmetric cryptography solves the problem of key distribution, it comes with the major drawback that the computations are rather slow compared to symmetric cryptography. That is why in practice typically a hybrid approach is chosen. The key distribution or key agreement is done by asymmetric primitives. Afterward, symmetric ciphers are used to carry out the bulk data encryption as fast as possible. Thus, symmetric cryptography builds the backbone of today's information security systems.

There exist different symmetric primitives like block ciphers, stream ciphers, and hash functions. The above-mentioned bulk data encryption is typically carried out by block ciphers. Besides being used for encrypting the major fraction of our sensible data, block ciphers are important building blocks in many cryptographic constructions and protocols.

For the application of block ciphers, the obvious research question is "How can we design secure block ciphers?". This question cannot be answered without also looking at techniques for attacking block ciphers. Besides this, another research question directly connected to the application of block ciphers is "How can we design/implement efficient block ciphers?". While efficiency is always a goal in computing, it becomes imperative in small devices with constrained resources.

These two research questions have been the motivation for the work presented in this thesis.

Contribution and Outline

We first introduce the basics and touch different aspects of the design and analysis of block ciphers. For a more fundamental and broad introduction to cryptology the reader is referred to [BR05; KL14; MVO96; PP09] and for basic textbooks about block ciphers we additionally recommend [DR02] and [KR11].

While Chapter 2 and Chapter 3 are dedicated to the analysis of block ciphers, Chapters 4 and 5 deal with design issues for lightweight cryptography.

Linear Cryptanalysis Chapter 2 serves as a systematization of knowledge of linear cryptanalysis and provides new insights in the areas of key schedule design and tweakable block ciphers. We examine the famous linear hull theorem [DGV95; Nyb95] in a step by step manner in a general and consistent setting. Based on this, we study the influence of the choice of the key scheduling on linear cryptanalysis, a – notoriously difficult – but important subject. Moreover, we investigate how tweakable block ciphers can be analyzed with respect to linear cryptanalysis. These results have been published in [KLW17].

Structural Cryptanalysis of ASASA Afterward, we deal with the analysis of the ASASA structure in Chapter 3 [Din+15]. That is, we consider the problem of recovering the internal specification of a general SP-network consisting of three affine layers (A) interleaved with two S-box layers (S), given only black-box access to the scheme. The decomposition of such general ASASA schemes was first considered at ASIACRYPT 2014 by Biryukov et al. [BBK14] who used the alleged difficulty of this problem to propose several concrete block cipher designs as candidates for white-box cryptography. We present several attacks on general ASASA schemes that significantly outperform the analysis of Biryukov et al. As a result, we are able to break all the proposed concrete ASASA constructions with practical complexity. For example, we can decompose an

ASASA structure that was supposed to provide 64-bit security in roughly 2^{28} steps, and break the scheme that supposedly provides 128-bit security in about 2^{41} steps. Whenever possible, our findings are backed up with experimental verifications.

Lightweight Linear Layers Finally, Chapters 4 and 5 consider the problem efficiently implementing MDS matrices in hardware. Here, “efficiently” means that the area needed for the hardware circuit shall be as small as possible. For measuring this efficiency, we use the so-called XOR count, i. e. the number of XOR operations needed to implement the matrix multiplication. In Chapter 4, we consider the more general and fundamental question of optimizing finite field multiplications with one fixed element. We investigate which field representation, that is which choice of basis, allows for an optimal implementation. Based on the results, we construct new MDS matrices. When these results were published in 2016 [BKL16] they outperformed or were on par with all previous results when focusing on a round-based hardware implementation. Subsequently, a lot of attention was paid to this problem and most of the work also concentrated on locally optimizing the multiplication with single matrix elements with the effect of small further improvements. In Chapter 5, we then shift the focus from the local optimization of single matrix elements to the global optimization of the whole matrix multiplication. It turns out that in a separate line of work, several heuristics were developed to find shortest linear straight-line programs. Solving this problem actually corresponds to globally optimizing multiplications by matrices. We combine this separate line of work with our problem of finding efficiently implementable MDS matrices. As a result, we achieve implementations of known, locally optimized, and new MDS matrices that significantly outperform all implementations from the literature. Interestingly, almost all previous locally optimized constructions behave very similarly with respect to the globally optimized implementation. As a side effect, our work revealed the by that time best implementation of the AES MixColumn operation with respect to the number of XOR operations needed. The according results were published in [Kra+17].

1.1 Block Ciphers

Let us consider one of the most basic and most important cryptographic goals: confidentiality. It is established by so-called ciphers or encryption functions. The input of such a function is typically called the *message* or the *plaintext* and will be denoted by m or x. Any unauthorized party should then be unable to derive any information whatsoever about the plaintext from the output data which is typically called the *ciphertext* and denoted by c or y. Authorization is usually proven by the knowledge of a secret value k, called the *key*.

Above, we presented the example of two parties who want to privately communicate over some insecure channel, e. g. via email. Any other party listening to the communication should then not be able to gain any information about the content of the messages. Another example of the need of encryption is the storage of secret data on some hard drive that might also be accessible to an unauthorized person.

A block cipher is a special kind of encryption function. But it can also be used as a building block to construct other symmetric algorithms such as hash functions or stream ciphers.

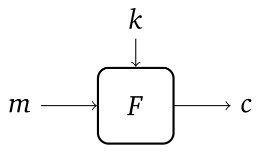


Figure 1.1: *A block cipher.*

Definition 1.1 (Block Cipher)**.** An n-bit block cipher is defined as a function $F : \mathbb{F}_2^n \times \mathbb{F}_2^\kappa \to \mathbb{F}_2^n$ where $n, \kappa > 0$ and for any fixed $k \in \mathbb{F}_2^\kappa$ the function $E_k(\cdot) := F(\cdot, k)$ is bijective. ◊

Thus, as depicted in Figure 1.1, a block cipher can be seen as a family of permutations that is indexed by a key k. Each permutation maps an input block of n bits to an output block of n bits. The according family of inverse permutations is called decryption and denoted by $D_k(\cdot)$.

Typical values for n are 64 and 128. Typical values for κ are 80, 128, and 256. For designing and analyzing a block cipher, we need to define in which case the cipher is considered secure and in which case it is not. This consists of two parts. On the one hand, we need to define what is considered a successful attack. On the other hand, we need to define the abilities of the attacker, the so-called attacker model.

1.1.1 Security Definition

Obviously, if the attacker knows the secret key, he can directly decrypt ciphertexts and learn the according plaintexts. Thus, one might have the first idea of defining a block cipher as secure, if the attacker cannot learn the secret key. However, there is a trivial example showing that this definition is not sufficient. Just imagine a block cipher that computes the identity mapping independently of the key. Such a block cipher is secure against key-recovery, but it is certainly not something that we want to denote as a secure block cipher.

Instead, we use the notion of a distinguishing attack, where the attacker has to find out if he is interacting with the actual block cipher or with a random permutation. If the attacker can distinguish between these two cases, the cipher is considered as broken. We can directly see, that indistinguishability implies security against key-recovery. On the other hand, distinguishability often leads to a key-recovery attack, as we will discuss in Section 1.3.2. The intuition behind this security definition is that a perfect cipher would just be a random permutation for every key and thus if we cannot distinguish our cipher from a random permutation we are very close to this perfect scenario. For this reason, indistinguishability from a random permutation is a well-known concept in provable security, where the security proofs of bigger constructions usually rely on the assumption that the block ciphers can be modeled as (strong) pseudorandom permutations (PRPs), i. e. as functions that cannot be efficiently distinguished from random permutations with a probability that has a non-negligible difference from $1/2$. Typical examples of such bigger constructions are the so-called *modes of operation* that define how the block cipher is used to process a number of bits that differs from the block size. However, so far no practical block cipher could be proven to be a PRP. Thus, in block cipher design, one does not make use of the formal PRP definition which is used in the area of provable security. Since we cannot prove the security, the best we can do is to give good arguments against its insecurity. Here, we consider the block cipher as insecure as soon as we have a distinguisher that works faster than an exhaustive search[2]. This constraint simply comes from the fact that a trivial distinguisher can always be found by searching through the whole key space. This way of defining (in)security directly implies parts of the attacker model presented in the next section, namely the assumption that an attacker can carry out any attack that has lower time complexity than exhaustive search. In the end, a block cipher is believed to be secure if it was thoroughly analyzed by multiple experts for some years and still could not be broken.

1.1.2 Attacker Model

The attacker model defines the resources of the attacker and the scenario in which the cipher is attacked. First of all, there exists the consensus that in every attacker model, the attacker must have full access to the cipher design. This important principle was established by Kerckhoffs in 1883.

Kerckhoffs' Principle [Ker83]. *The security of a cryptosystem should not rely on its secrecy. It should not cause any inconvenience if it falls into the hands of the attacker.* $\Diamond$

The obvious reason for this principle is that it is very hard to keep a cryptographic algorithm secret. Often, the algorithms can be recovered by reverse-engineering techniques. There exist numerous cases in which cryptographic systems were broken because the designers did not adhere to Kerckhoffs' principle. One famous example is the *Content Scramble System* that was used for the protection of DVD-Video content and has been completely broken once the specification became public. Another more recent example is the *Megamos Crypto* transponder which was reverse-engineered and broken in 2012 [VGE15] and is used as an electronic vehicle immobilizer by many car manufacturers, including Audi, Fiat, Honda, Volkswagen, and Volvo.

In addition to Kerckhoffs' principle, another important part of the attacker model is the scenario in which the attacker is interacting with the block cipher. Here, we define the type of data that the attacker can get his hands on and also his potential knowledge about the secret key.

The data that the attacker can use in his attack is plaintexts and ciphertexts. We denote by d the amount of data that the attacker can use. Depending on the way this data is collected, we differentiate between the following scenarios:

- *Ciphertext-only attack.* The attacker knows a set of ciphertexts $\{c_1, \ldots, c_d\}$.
- *Known-plaintext attack (KPA).* The attacker knows pairs $\{(m_1, E_k(m_1)), \ldots, (m_d, E_k(m_d))\}$.
- *Chosen-plaintext attack (CPA).* The attacker can choose plaintexts $\{m_1, \ldots, m_d\}$ and receives $\{E_k(m_1), \ldots, E_k(m_d)$.
- *Chosen-ciphertext attack (CCA).* The attacker can choose ciphertexts $\{c_1, \ldots, c_d\}$ and receives $\{D_k(c_1), \ldots, D_k(c_d)$.

[2]Exhaustive search, also called "brute-force" is the attack that just tries out all keys until finding the correct one. It will be explained in Section 1.3.1.

Additionally, the CPA and the CCA might be adaptive. In an adaptive CPA, the attacker does not need to choose all plaintexts in advance, but he can choose the plaintext m_i with the knowledge of the pairs $\{(m_1, E_k(m_1)), \ldots, (m_{i-1}, E_k(m_{i-1}))\}$. The adaptive CCA works analogously. The ciphertext-only is obviously the weakest attacker model and it is the typical model that one has in mind when thinking about an attacker who eavesdrops on some communication channel. However, also the KPA is a very common attack and even the (adaptive) CPA/CCA might be practically relevant. Namely in situations in which the attacker can query the encryption or decryption function, for example in certain communication protocols or because he has physical access to the cryptographic device. In block cipher design, we have very high standards and typically assume very strong attackers, that is, adaptive CPA/CCA. However, as we will see later, linear cryptanalysis is one of the strongest attacks on block ciphers although it is only a KPA.

When it comes to the attacker's knowledge about the key, the most common scenario is the *secret-key model*, where we assume that the attacker has no information about the key. In the *related-key model* introduced by Biham [Bih94], the attacker knows relations between different keys that are used in different instantiations of the cipher. This model has become very relevant in the context of tweakable block ciphers that will be discussed in Section 1.2.3. Here, the attacker might not only be able to know but actually to choose an XOR-difference in the key. In addition to the secret-key model and the related-key model, there also exists the *known-key model* [KR07] and the *chosen-key model* [BKN09]. These models can be relevant if the block cipher is used to instantiate a hash function. However, in this thesis, we will always assume to be in the secret-key model.

When an attack is presented, it always comes with an attack complexity, which is measured in time, memory, and data. Here, the time complexity gives the number of computations that need to be carried out, memory gives the number of values that need to be stored, and data gives the number of plaintexts or ciphertexts that need to be available to the attacker. Whether or not this attack can be carried out by an attacker depends on the attacker's hardware resources and on the scenario in which the cipher is used. And it also depends on the time the attacker is willing to spend[3]. These points of the attacker model are defined in the block cipher design and are based on the main idea already presented above: that any attack which is better than exhaustive search is considered a successful attack. Note, that this approach is sound because on the one hand exhaustive search always works, so it does not make sense to regard any attack that has higher complexity. On the other hand, good block cipher design can ensure that all known attacks have higher complexity then exhaustive search[4]. This is in contrast to public key cryptography, where the key length must often be chosen much higher because there exist attacks that cannot be avoided and are faster than an exhaustive search. Thus, in a block cipher design, the key length defines the computation power and the time of the attacker, because it is assumed that the attacker can run any attack that needs fewer computations than exhaustive search. The block length gives a natural bound for the amount of data that is available to the attacker.

[3]The attacker's time should not be confused with the *time complexity* which is the number of computations that need to be carried out in a specific attack.

[4]This does not hold anymore in a post-quantum scenario, where Grover's algorithm [Gro96] can be used to reduce the effective key length of block ciphers by a factor of two. Also, as pointed out by Leander and May [LM17], there is not much work about post-quantum symmetric cryptography Grover's algorithm might not be the only threat for post-quantum symmetric cryptography. Throughout this thesis, we will not consider post-quantum scenarios.

However, one might want to explicitly define the amount of data available in the attacker model, for example if there are functional (non-cryptographic) reasons for choosing a particularly high block length. An example of this scenario is the block cipher LowMC [Alb+15] where the authors explicitly decouple the block size from data available in the attacker model because they use large block sizes. Anyway, note that the amount of available data is already naturally bounded by the time complexity because the data needs to be collected first which can also be seen as part of the attack. In the same way, we can argue that the amount of memory is also already bounded by the time complexity.

1.2 Design of Block Ciphers

As already stated in the discussion of security definitions, we are not aware of any method to design a block cipher that can be proven secure while being practical at the same time. Instead, in block cipher design, security is achieved by giving arguments on why the cipher withstands all known attacks. We will present the basic design principles of block cipher design in the following, leading to today's most popular design called substitution-permutation network (SPN). Later, when discussing the specific attack techniques in Section 1.3, we will show how concrete security arguments can be given based on the design. After presenting the basic design principles, we will introduce four areas of block cipher design that will be important for the subsequent chapters.

1.2.1 Design Principles

The basic principles in block cipher design date back to the seminal work of Shannon [Sha49] and his concept of *confusion* and *diffusion*. These notions have been created by Shannon in 1949 and there exist numerous reinterpretations for the design of modern cipher design. Here, we quote a modern description attributed to James L. Massey [KR11]:

Confusion: The ciphertext statistics should depend on the plaintext statistics in a manner too complicated to be exploited by the cryptanalyst.

Diffusion: Each digit of the plaintext and each digit of the secret key should influence many digits of the ciphertext.

As remarked in [KR11], the properties of confusion and diffusion are not absolute, quantifiable concepts. It should rather be seen as basic design principles or as an intuition of what a block cipher designer should strive for. In block cipher design, the concepts of confusion and diffusion are often attributed to specific parts of the cipher. Most block ciphers use the concepts of substitution and permutation. This is, as the name suggests, most obvious for the substitution-permutation network (SPN) which is introduced below. Substitution is an operation that realizes the concept of confusion by carrying out a non-linear operation. Often this is implemented by a lookup table, such that the input is just substituted with a predefined output. An according lookup table is called an S-box. Due to implementation limitations, the S-boxes cannot operate on the full block size. Typical input sizes for S-boxes are 4 or 8 bits. The permutation operation then contributes to the diffusion and is often implemented by a bit-permutation. However, nowadays, many ciphers do not use bit-permutations any more. Instead, the more general

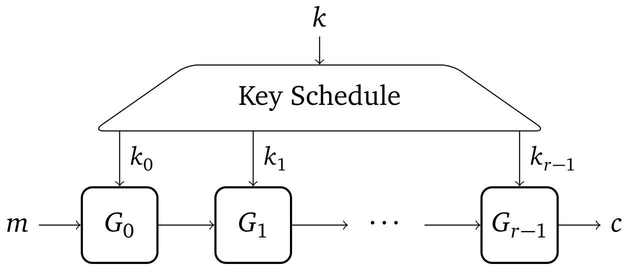


Figure 1.2: *A round-based block cipher.*

operation of a linear function is used which can be nicely implemented by a matrix multiplication and can introduce more diffusion per application than a mere bit-permutation. However, the term "permutation" has become established and is still used, even when we do not talk about a bit-permutation[5].

Virtually all relevant block ciphers nowadays are round-based block ciphers that can be written as $F = G_{r-1} \circ \cdots \circ G_1 \circ G_0$. Here, every round G_i uses some means for confusion and diffusion and some information derived from the key. This information is called the *round key* and will be denoted by k_i for round G_i. To make clear the difference between the round keys and the original key, the latter is sometimes also referred to as the *master key*. The function that derives the round keys from the master key is called the *key schedule*. Fig. 1.2 shows the structure of a round-based block cipher. These ciphers are also often referred to as *product ciphers*.

For a fixed round key, each round function G_i must be an invertible mapping from $\mathbb{F}_2^n$ to $\mathbb{F}_2^n$. Decryption can then be carried out by using the same round keys and computing $F^{-1} = G_0^{-1} \circ \cdots \circ G_{r-2}^{-1} \circ G_{r-1}^{-1}$. If the round functions are identical, i. e. $G_0 = G_1 = \cdots = G_{r-1}$, this is called an *iterated cipher*. Iterated ciphers are very popular because they allow efficient implementations and nice proofs against existing attacks as we will see in Section 1.2.4 and Section 1.3, respectively. The two most common designs for round-based ciphers are Feistel networks and substitution-permutation networks (SPNs). Another important class of round-based block ciphers are so-called *Addition-Rotation-XOR* (ARX) ciphers that only use modular addition, bitwise rotation, and the XOR operation. However, we will not consider ARX designs in this thesis and only introduce Feistel networks and SPNs in the following.

Feistel Ciphers

In a Feistel cipher, each round treats the input as two halves. The output is then computed by applying a key-dependent function f to the right half and adding the result to the left half. Finally, the two halves are swapped and form the input for the next round.

[5]In some sense, this is fine since also a linear function which is not a bit-permutation is a permutation in terms of being bijective. However, this can get confusing, because also bijective S-boxes are often said to be permutations with the same reasoning in mind.

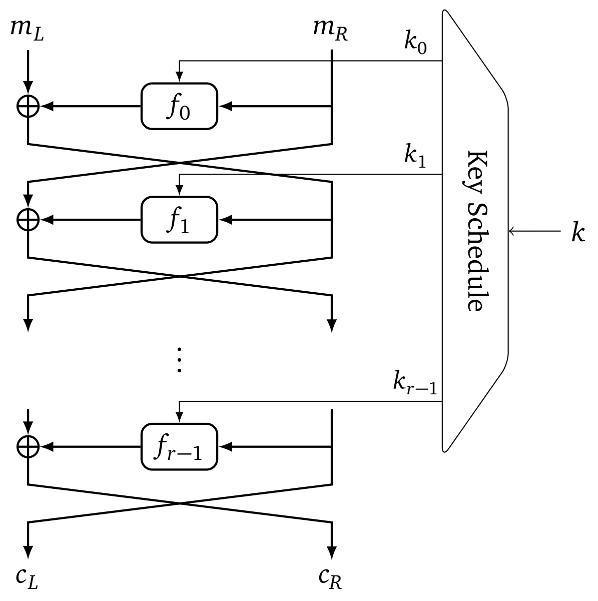


Figure 1.3: *A Feistel cipher.*

Definition 1.2 (Feistel-round). Let $f : \mathbb{F}^{n/2} \times \mathbb{F}^{\kappa} \rightarrow \mathbb{F}^{n/2}$. Now let $x = x_L \| x_R$ be the input where $x_L, x_R \in \mathbb{F}^{n/2}$ and '||' denotes the binary concatenation. The output $y = y_L \| y_R$ is computed as follows:

$$\begin{aligned} y_L &= x_R \\ y_R &= x_L + f(x_R, k) \end{aligned}$$

◊

The design of a Feistel cipher is depicted in Fig. 1.3. Note that the swap in the last round of Fig. 1.3 has no cryptographic relevance and is thus often omitted, for example in the well-known Data Encryption Standard (DES) [DES77]. However, it might also be useful for efficient implementation because if $f_0 = \ldots = f_{r-1}$, we have an iterated cipher and the same round function can then be reused for every round.

The functions f_i are used to implement the principles of confusion and diffusion and to insert the key. They do not have to be invertible, because each round can be inverted without inverting f_i by computing $x_R = y_L$ and $x_L = y_R + f_i(y_L, k)$. The most famous Feistel cipher undoubtedly is the Data Encryption Standard (DES) [DES77]. Due to its short key-length, it should not be used anymore today and was replaced by the Advanced Encryption Standard (AES) [AES01]. However, a variant which is called 3DES solves the problem of the short key length by applying the DES three times in a row and is still widely used in commercial products, for example in the financial sector.

Substitution-permutation networks (SPNs)

The SPN is a very popular structure for block cipher design. It is simple to analyze and enables the designer to give strong arguments for the security against existing attacks as we will learn in Section 1.3. As explained in [DR02, Chapter 5.2], simplicity comes with the additional advantage that it leads to more external analysis because cryptographers, in general, prefer to study simple ciphers rather than obscure ones. In an SPN, each round consists of a substitution layer followed by a permutation layer. These two building blocks were already discussed above when introducing the concepts of confusion and diffusion. Recall that the substitution layer consists of predefined lookup tables that typically operate on small bit sizes, for example 4 or 8 bits. The permutation layer is an $\mathbb{F}_2$-linear function that operates on the whole state[6] and is therefore also called the linear layer.

Definition 1.3 (SP-round)**.** Let $S_i : \mathbb{F}_2^m \mapsto \mathbb{F}_2^m$ be bijections for $0 \le i < t$. Further, let $L : \mathbb{F}_2^n \mapsto \mathbb{F}_2^n$ be an $\mathbb{F}_2$-linear invertible function. Now, given the n-bit input $x = x_0 \| x_1 \| \dots \| x_{t-1}$ with $x_i \in \mathbb{F}_2^m$, the output of the SP-round is computed as

$$y = L(S_0(x_0) \| S_1(x_1) \| \dots \| S_{t-1}(x_{t-1})).$$

$\diamond$

The functions S_i are called m-bit S-boxes. The invertibility of the S-boxes and the linear function implies invertibility of the whole SP-round. Often, all the S-boxes in the whole cipher are the same. This simplifies the algorithm and the proof of security against existing attacks. Also, it enables are more efficient implementation. In Definition 1.3 we directly notice that the influence of the round key is missing. This is because the SPN is a so-called key-alternating block cipher that consists of r key-independent rounds and $r + 1$ round keys that are added at the beginning, at the end, and in between every round.

Definition 1.4 (Key-alternating block cipher)**.** Let $H_i : \mathbb{F}_2^n \mapsto \mathbb{F}_2^n$ be key-independent invertible round functions for $0 \le i < r$. Further, let $k_0, \dots, k_r \in \mathbb{F}_2^n$ be $r + 1$ round keys derived from the master key. A key-alternating block cipher computes the ciphertext as

$$c = H_{r-1}(\cdots(H_1(H_0(m + k_0) + k_1) + k_2)\cdots) + k_r.$$

$\diamond$

Note that we use H for denoting a key-independent round function whereas G was used above to denote a key-dependent round function when describing a round-based block cipher. A key-alternating block cipher can be represented as a special case of a round-based block cipher. A first motivation for the key-alternating structure is that it makes sense to start and to end with introducing key material because according to Kerckhoffs' principle, the attacker knows every detail of the cipher except for the key. Thus, he can always compute forward from the plaintext or backward from the ciphertext until the first or the last round key. Accordingly, any part of

[6]The notion "state" is used to describe the intermediate results of the encryption function and thus the inputs and outputs of the round functions.

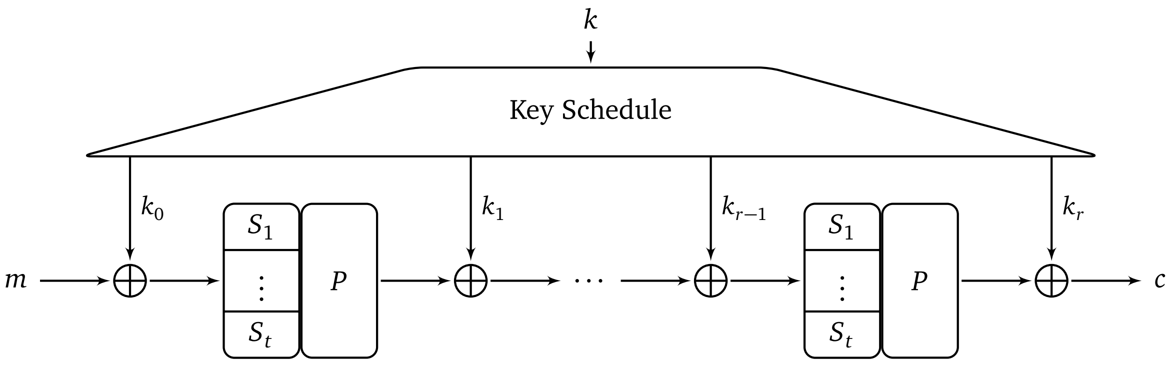


Figure 1.4: *A substitution-permutation network (SPN).*

the block cipher that comes before the first or behind the last round key must be regarded as cryptographically irrelevant. Now, starting the first and ending the last round with a key addition, it seems natural to continue this approach by starting and ending every round with a key addition and indeed it is this simple structure which can be very helpful. A good example is the linear hull theorem which is the basic theorem from linear cryptanalysis and simplifies very much in the case of a key-alternating structure. This will be shown in detail in Chapter 2. Finally, note that addition is understood with respect to the binary strings being represented as elements of the vector space $\mathbb{F}_2^n$. This is sometimes also called exclusive-or (XOR). Especially, it is convention to use the '$\oplus$' symbol in figures for this operation.

We want to point out that SPNs and key-alternating ciphers are not the same. SPNs are a special class of key-alternating ciphers, but one can easily think of key-alternating ciphers that are not SPNs. Actually, many Feistel ciphers are also key-alternating ciphers which can be seen if they are represented in an alternative way[7], as illustrated for example in [DR05, Figure 5].

Fig. 1.4 depicts an SPN that uses the same S-box and the same permutation throughout the whole cipher. The most famous representative of this block cipher class is the already mentioned AES, which is nowadays the most studied block cipher of the world.

The key schedule

We already introduced the key schedule as a function that derives the round keys from the master key. But so far, we have not given any detail on design principles for the key schedule. However, before bothering too much with key schedule design, we might first ask why there needs to be a key schedule at all. Why do we not define all the round keys as the master key and get rid of the key schedule? The reason is that we want to minimize the information that we have to keep secret and, according to Kerckhoffs' principle, this is only the master key. Naturally, keeping for example 128 bits secret is easier than keeping $11 \cdot 128 = 1408$ bits secret what would be the case for a 10-round key-alternating cipher with 128-bit block size. In practice, secret keys are often stored in trust anchors with constrained resources. This could be a smartcard or a Hardware

[7]Note, however, that such an alternative representation typically also restricts the choice of the key schedule of the according key-alternating cipher.

Security Module (HSM). Especially if many keys have to be stored, big keys could then cause problems due to memory constraints. Also, small keys significantly simplify key management processes such as key updates and key imports/exports. Altogether, this leads to the design principle of choosing keys as small as possible but of course as big as necessary for our security requirements. In light of this, it now becomes even clearer why we defined a block cipher as insecure if there exists an attack that is faster than brute-force (see Section 1.1.1).

When it comes to the design of the key schedule, things turn out to be not so easy. In general, it is not clear what properties a good key schedule has to have. There are some general guidelines on what a key schedule should *not* look like. These guidelines are rather basic and ensure mainly that trivial guess-and-determine or meet-in-the-middle attacks are not possible. In a nutshell, it should not be possible to compute large parts of the encryption algorithm, i. e. a large number of rounds in the case of round-based ciphers, without having to know or guess the whole master key. An example of such a trivially bad key schedule is the idea of using two independent keys in order to double the key length, i. e. double encryption.

There are also a few more sophisticated attacks that exploit weaknesses in the key schedule. Accordingly, a key schedule should be designed such that such attacks (e. g. slide-attacks [BW99], invariant subspace attacks [Lea+11; LMR15], non-linear invariant attacks [TLS16]) are not possible. It is often possible to check, for a given key schedule, if it fulfills this criterion, as for example in recent work of Beierle et al. [Bei+17]. The application of round constants is a typical design principle that is used to counter such attacks.

However, the most well-known attacks on block ciphers are differential and linear attacks which will be introduced in Section 1.3.3 and Chapter 2, respectively. Here, the influence of the key schedule is to a large extent completely open. Simply put, for claiming a cipher secure against linear and differential attacks, one has to demonstrate that the cipher does not possess certain statistical irregularities.

To be able to do so, it is in many cases necessary to assume that all round keys are independently and uniformly chosen as we will see later when discussing these attacks in detail. While this is hardly the case for any real cipher, this assumption is on the one hand needed to make the analysis feasible and on the other hand often does not seem problematic as even with the keys not independently and uniformly chosen, most ciphers (experimentally!) do not behave differently from the expectation. Accordingly, different approaches for the design of key schedules exist. Knudsen and Robshaw suggest the following classification of existing key schedules in [KR11, Section 8.5]:

- *Affine key schedules.* Subkeys are derived as an affine transformation of the master key.
- *Non-linear key schedules.* Subkeys are generated as simple non-linear transformations of the master key.
- *Complex key schedules.* The subkeys are generated as complex non-linear transformations of the master key.

For example, DES has an affine key schedule while AES has a non-linear key schedule involving S-boxes. The fact that the decision for one of these approaches in terms of security is basically a matter of taste is highly unsatisfying both from a scientific and from a practical point of view. In

Chapter 2, we address this problem and take some steps forward to increase our understanding of the influence of the key schedule on the security of a block cipher with respect to linear attacks.

1.2.2 Finite Field Representations and the Linear Layer

Many cryptographic schemes are built on finite fields as their underlying mathematic structure, most prominently the AES. In almost all cases, the schemes can be designed without having to specify a concrete representation of the finite field in advance. However, for the sake of interoperability, one necessarily has to choose a particular representation of the finite field when implementing the cipher on the bit-level. In general, this choice does not influence the security of the scheme, but might well influence the performance of the resulting implementation. This is actually a very natural separation of the design of the cipher and its specification (and thus implementation) on the bit-level. This separation has been nicely explained in [DR11] by introducing RIJNDAEL-GF, which is an abstracted version of AES specified using only algebraic operations in $\mathbb{F}_{2^8}$. Now, by choosing a specific representation of the finite field, one can derive AES as an instance of RIJNDAEL-GF. Following [DR11], the choice of basis is to a large extent independent of the design and the security of the cipher. However, the choice of basis might have a significant impact on the efficiency of the cipher on certain platforms.

Let us first recall some basics about finite fields [LN97] and their representations [War94]. $\mathbb{F}_{2^m}$ is the finite field with 2^m elements, often also denoted as GF(2^m). Up to isomorphism, every field with 2^m elements is equal to $\mathbb{F}_2[x]/(q)$, where the $\mathbb{F}_2[x]$ is the polynomial ring over $\mathbb{F}_2$ and (q) denotes the ideal generated by an irreducible polynomial q of degree m: $\mathbb{F}_{2^m} \cong \mathbb{F}_2[x]/(q)$. Although there exists up to isomorphism only one finite field for every possible order, we are interested in the specific representation.

Let $V \cong K^m$ be a finite-dimensional vector space over the field K. Every linear mapping $f : V \to V$ can be described as $v \mapsto A_B v$ by a left-multiplication with a matrix A_B being an $m \times m$ matrix over K. This representation is dependent on the choice of the basis B for V. For instance, if $B = \{b_1, \ldots b_m\}$, the j-th column of A_B consists of the coefficients $a_{1,j}, \ldots, a_{m,j}$ of $f(b_j) = \sum_{i=1}^{m} a_{i,j} b_i$. Thus, changing the basis from B to B' results in a different matrix representation of f.

There is a natural way of representing the elements in a finite field with characteristic 2 as vectors with coefficients in $\mathbb{F}_2$. More precisely, there exists a vector space isomorphism $\Phi_B : \mathbb{F}_{2^m} \to \mathbb{F}_2^m$ which maps elements $\alpha \in \mathbb{F}_{2^m}$ to its vectorial representation over $\mathbb{F}_2$ with regard to a basis B (and Φ_B^{-1} vice versa). Every multiplication by an element $\alpha \in \mathbb{F}_{2^m}$ can then be described by a left-multiplication with a matrix $T_{\alpha,B} \in \mathbb{F}_2^{m\times m}$ as shown in the following diagram.

$$\begin{tikzcd}
\mathbb{F}_{2^m} \arrow[r, "\cdot\alpha"] \arrow[d, "\Phi_B"'] & \mathbb{F}_{2^m} \\
\mathbb{F}_2^m \arrow[r, "T_{\alpha,B}"] & \mathbb{F}_2^m \arrow[u, "\Phi_B^{-1}"']
\end{tikzcd}$$

$T_{\alpha,B}$ is usually called the multiplication matrix of the element α.

Now, when designing a block cipher, one of the most important design strategies is the so-called wide trail strategy, which was initiated in [Dae95] and will shortly be discussed in Section 1.3.3. The main observation of the wide trail strategy is that it is actually the linear layer that is to a large extent responsible for the security of the primitive against linear and differential attacks. Moreover, the wide trail strategy allows a natural decoupling of the design choice for a linear layer and an S-box.

Interestingly, for the linear layer, not many general constructions are known. Two basic approaches can be identified. On the one hand, an ad-hoc approach that requires computer-aided tools. This approach is used for example for the block cipher Serpent [BAK98] or the hash function Keccak [Ber+11]. On the other hand, a code-based approach, where the linear layers are chosen in such a way that they correspond to good (often locally optimal) linear codes. This is, most prominently, the case for AES where a Maximum Distance Separable (MDS) code is implemented via the MixColumns operation.

Even in the theoretically better circumstantiated code-based approach, many fundamental questions are left open. Here, when using an MDS matrix for (parts of) the linear layer, the main challenge is to choose an MDS matrix that is most suitable for an efficient implementation. As those MDS matrices are usually defined over a finite field with characteristic two, i. e. $\mathbb{F}_{2^m}$, one important question is the choice of an $\mathbb{F}_2$-basis of $\mathbb{F}_{2^m}$ and its impact on the implementation efficiency. From a design point of view, one has to choose a linear layer given as a mapping on $\mathbb{F}_{2^m}^t$ and an $\mathbb{F}_2$-basis of $\mathbb{F}_{2^m}$ to concretely specify the primitive.

For hardware implementations, the impact of the choice of basis already becomes apparent when focusing on how to implement the multiplication with one given element α in $\mathbb{F}_{2^m}$. For different choices of bases, the efficiency of implementations of the resulting $\mathbb{F}_2$-linear mappings differs significantly.

In Chapter 4, we study the task of finding a basis for a given element $\alpha \in \mathbb{F}_{2^m}$, such that multiplication can be implemented most efficiently. We then use the results for designing efficient linear layers for block ciphers, i. e. we present linear layers that can be implemented efficiently by an appropriate choice of basis. Afterward, in Chapter 5, we present methods to find efficient implementations of binary matrix multiplications. We then make a paradigm shift by applying these methods not only to the single multiplication matrices $T_{\alpha,B}$, but to the binary representation of the whole MDS matrix. In this way, even more efficient implementations can be obtained.

1.2.3 Tweakable Block Ciphers

This deterministic behavior can lead to critical problems in the application of a block cipher. Block ciphers, as defined in Definition 1.1, are deterministic algorithms, i. e. if the same message is encrypted multiple times, it will result in the same ciphertext every time. Following the naive approach and encrypting a set of plaintext blocks one-by-one independently of each other would lead to a set of ciphertexts in which it is possible to determine which ciphertexts correspond to equal plaintexts, which can lead to information leakage. A classical example is image encryption where the image can still be recognized because areas with the same plaintext color yield identical ciphertext colors. This problem is addressed by choosing a proper mode of operation[8]. But still,

[8]Modes of operation define how to process messages that consist of multiple blocks. For example, the aforementioned naive idea of just encrypting each block one-by-one is called the Electronic Codebook (ECB).

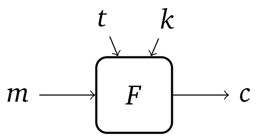


Figure 1.5: *A tweakable block cipher.*

every time the same set of plaintext blocks is encrypted, it will then result in the same set of ciphertext blocks. To deal with this problem, modes of operation usually come with an additional input, typically called initialization vector (IV), that can be changed for eliminating the described deterministic behavior also on the whole set of blocks. Unlike the key, the IV is public.

A tweakable block cipher now brings this concept of an additional randomizing input to the block level. Tweakable block ciphers were introduced by Liskov et al. in 2002 [LRW02]. The traditional definition of a block cipher (see Definition 1.1) is extended by an additional τ-bit input called the *tweak*, denoted by t.

Definition 1.5 (Tweakable Block Cipher). A n-bit tweakable block cipher is defined as a function $F : \mathbb{F}^\kappa \times \mathbb{F}^\tau \times \mathbb{F}^n \to \mathbb{F}^n$ where $n, \kappa, \tau > 0$ and for any fixed $t \in \mathbb{F}^\tau$ and $k \in \mathbb{F}^\kappa$ the function $F(k, t, \cdot)$ is bijective. ◊

Informally, the intuition is that each tweak selects a different block cipher, i. e. a different, unrelated, family of permutations. An according diagram is shown in Fig. 1.5. While the key is, obviously, assumed to be unknown to an attacker, the tweak is assumed to be public. Moreover, following the chosen-plaintext/ciphertext model, we usually assume that also the tweak is under full control of the adversary. That is, the adversary is allowed to query the tweakable block cipher under a plaintext/ciphertext and tweak of his choice.

Tweakable block ciphers have many important applications, e. g. ciphers for memory encryption can use the memory address as a tweak and then decrypt with random access. An example of a cipher that is designed to be suitable for this use case is the block cipher QARMA [Ava17]. A further application is the design of efficient authenticated encryption. For example, the authenticated encryption design DEOXYS-II [Jea+16] is based on the tweakable block cipher DEOXYS-BC [JNP14b] and it was recently selected as a finalist in the CAESAR competition[9]. Another application of tweakable block ciphers are online ciphers [Bel+01; RZ11]. Furthermore, the tweakable block cipher QARMA [Ava17] is used in the *ARMv8.3-A* architecture for hardware-based pointer authentication [Qua17].

When designing a block cipher, one has to argue why the cipher is secure against all the existing methods of cryptanalysis. As we will learn in Section 1.3, the two most prominent attacks are differential and linear cryptanalysis. Now, using a tweak equips the attacker with additional degrees of freedom because he has an additional input in the chosen-plaintext/ciphertext model. It is known and well-understood what this means for differential cryptanalysis and how to take it into account in the design of tweakable block ciphers. However, the impact for linear cryptanalysis was not so clear. In Section 2.5, we address exactly this important problem. Our results can be

[9]CAESAR: Competition for Authenticated Encryption: Security, Applicability, and Robustness. https://competitions.cr.yp.to/caesar.html

helpful for the design of tweakable block ciphers. For example, they were used in the security analysis of the tweakable block cipher family SKINNY [Bei+16].

1.2.4 Lightweight Block Ciphers

When thinking about cryptographic applications, one might first have in mind a typical personal computer that is operated directly by an end-user. However, there also exists a huge market of embedded microcontrollers. Typical applications are smartcards, televisions, home automation, or automobiles. A modern automobile has dozens of microcontrollers to control various systems like the airbags, the brakes, or the engine. The trend of equipping many things in daily life with small microcontrollers and connecting them is often termed the *Internet of things* (IoT) or *ubiquitous computing*. Within this trend, many of these embedded systems also have to run cryptographic functions to ensure privacy and security, for example in the realm of smart homes or connected automobiles. In contrast to powerful general-purpose computer systems, most embedded systems have only very constrained resources. Such resources might be the amount of memory, the battery, or also the area on the chip in case of hardware implementations. Additionally, latency might be a hard requirement if the embedded system controls time-critical applications like an automobile's braking system. Traditional cryptographic primitives are often not suited for embedded systems because they cannot fulfill these special requirements. They occupy too much hardware area, use too much memory, consume too much energy, or have an unacceptably high latency. Therefore, the development of special cryptographic primitives that are dedicated to these situations is very important. The according area of research is called *lightweight cryptography*.

At this point, we want to mention that lightweight cryptography is also sometimes related to a trade-off between key length and efficiency. In some scenarios, a rather small key length might indeed be sufficient, for example if the data does only need to be protected for a rather short period of time. One might even decide to accept the risk of a brute-force attack if the attacker would have to be very powerful and invest lots of money. Actually, such risk assessments are quite normal in the industry. However, designing a cipher with small key length always comes with the risk of it being misused by people who are not aware of the presumed special operating conditions. In the currently ongoing standardization process for lightweight cryptographic algorithms, the United States National Institute of Standards and Technology (NIST) does require lightweight block ciphers to withstand cryptanalytic attacks with up to 2^{112} computations [NIST18] and thus does not allow for such trade-offs.

In the recent two decades, lots of research on lightweight cryptography has been done. Many block ciphers have been designed with different goals like minimizing chip area (e. g. [Bei+16; Bog+07; Guo+11; Shi+11]), program memory (e. g. [Alb+14; Dae+00]), energy (e. g. [Ban+15]), or latency (e. g. [Bei+16; Bor+12]). However, the categorization is not always that easy. Many lightweight block ciphers are designed with multiple goals in mind and often a good software or hardware implementation also refers to an according high throughput. There also exist further design goals besides the classical ones. For example, some ciphers have been designed to allow for efficient masking which is a countermeasure against side-channel attacks [Gér+13; Gro+15; PRC12] . Recently, more extreme designs in this direction have been proposed following the goal of constructing ciphers with a minimal AND count, that is, a cipher that can be implemented with as few AND operations as possible [Alb+15; Dob+18]. Minimizing the AND count is motivated by the fact that for certain applications like for example homomorphic encryption, the computation of an AND operation is especially expensive. Depending on the application, the goal might be to reduce the total number of ANDs, the number of ANDs per bit, or the AND depth (that is, the number of AND operations, that cannot be parallelized). For an overview of lightweight symmetric cryptography including an extensive list and benchmarks of lightweight block ciphers, the reader is referred to [BP17].

When it comes to the goal of minimizing the occupied chip area, a classical approach of designing a block cipher is to use an iterated cipher. The round function than just needs to be implemented once with some additional overhead that realizes a loop. Another typical design strategy to minimize the chip area is to use a simple affine key schedule instead of a complex non-linear one (see Section 1.2.1). In Chapter 2, we examine simple affine and linear key schedules and proof that such key schedules can indeed be used to construct secure ciphers with respect to linear cryptanalysis.

Besides these rather general design strategies, it is also possible to focus on the single building blocks of a block cipher. Using small hardware implementations of S-boxes and linear layers can lead to a significantly reduced hardware area for the whole cipher. Moreover, lightweight implementations of such building blocks can also lead to more efficient implementations in other areas than block ciphers, for example in the area of hash functions. There are two interesting sides to this research. On the one hand, one can search for new building blocks with very small hardware implementations. These building blocks can subsequently be used in the design of new lightweight ciphers. On the other hand, the implementation of existing building blocks can be improved such that it is also possible to find smaller hardware implementations for already existing ciphers. Obviously, both is important, finding very light new designs on the one hand, but also improving lightweight implementations of already established ciphers.

When we try to compare lightweight implementations in terms of small hardware area, an important question that arises is what metric should be used. A typical approach is to compare the gate equivalents needed for an implementation on a field-programmable gate array (FPGA). In this case, we have several aspects influencing the final outcome, namely the written code using some hardware description language, but also the library used for synthesizing and the underlying technology. In the case of linear layers, a more abstract metric called *XOR count* has gained acceptance. The XOR count is the number of used XOR operations, where the whole implementation uses no other operation than XOR. This simplified metric has the advantage of having a simple mathematical description and being independent of the applied technology. The XOR count was first used in [Kho+14] where it also became clear that there is a correlation between low XOR count and small hardware area. In Chapters 4 and 5, we address the problem of optimizing the hardware area of linear layer implementations measured by the XOR count. We present new theoretical results and new implementations that have a significantly lower XOR count compared with previous publications.

1.2.5 White-Box Cryptography

Traditionally, implementations of block ciphers have been regarded as black-box implementations. That is, while the attacker is able to observe ciphertexts, maybe some plaintexts and also knows what encryption algorithm is used, he is not able to observe any intermediate results of the encryption. Neither does he know any details about the implementation such as the source code in case of a software implementation. This black-box model holds for the traditional setup in which the communicating parties are considered to be located in trusted environments. However, this is not always the case in real-world applications. In the late nineties, research about side-channel attacks broke up with the black-box model and considered implementation details like the time required for computation [Koc96] or power consumption that might be measured by an attacker who has physical access to the device running the cryptographic algorithm [KJJ99]. Furthermore, if an attacker has physical access to a cryptographic device, the secret keys must be stored in a secure way such that they cannot easily be read from the memory by an attacker. By now, the area of physical attacks has evolved into a vivid field of research and according countermeasures are implemented in many civil and military devices. Unfortunately, the deployment of hardware security measures is rather expensive. Consequently, we have many platforms in use today that are not equipped with any hardware protection mechanisms, especially in the market of low cost embedded devices. Having such devices at hand, attackers can use reverse engineering methods to obtain information about the source code. They can also manipulate the source code as in the simple key whitening attack where the S-boxes of the last round are reverse-engineered and set to zero, such that the output of the cipher is simply the last round key [KK06].

White-box cryptography is trying to address these problems by offering to protect cryptographic assets in software although assuming the white-box model, i. e. the attacker has full access to the source code and all intermediate results. The first white-box implementations were presented in 2002 in the seminal works of Chow et al. [Cho+02; Cho+03], motivated by digital rights management (DRM) applications. These initial works focused on the prevention of key extraction. Thus, while being aware that an attacker who controls the device can obviously make use of the functionality (e. g. decryption of protected media content in the DRM example), the goal is to prevent the attacker from learning the secret key. This obviously comes with the problem that the attacker can simply extract the whole implementation instead of the secret key, which is called *code lifting*. A suggestion to thwart such an attack, which is for example discussed in [Cho+03], is to apply random input and output encodings as bijections F and G. The white-box implementation then computes $G \circ E_k \circ F^{-1}$ which is useless for a code lifting attacker. The encodings must be added and removed at another place in the system in a way such that an attacker cannot learn them. And here, we see the logical flaw of this suggestion, namely it solves the inherent problems of the white-box scenario by creating a new domain that is able to hide from the white-box access. This is, at least from an academic point of view, an extremely unsatisfying solution. In the aftermath of these seminal papers, lots of practical work on white-box cryptography has been done [Wys09] and more security notions besides key extraction have been suggested [Del+14]. A very important one is *incompressibility*. It basically means that an attacker who has access to the white-box implementation of a block cipher is not able to find a significantly smaller representation of the cipher. If the white-box implementation is sufficiently large, this makes large scale sharing of the implementation very inconvenient. For this security notion, Biryukov et al. [BBK14] present a white-box scheme is based on big lookup tables that are constructed by alternating layers of affine transformations and S-boxes. That is, while there exists a small black-box version of the cipher that uses the affine transformations and the S-boxes, the white-box version only uses the according big lookup table. In Chapter 3, we show that an attacker can compute the single building blocks from the lookup table and thus this white-box scheme is insecure. Developing secure white-box implementation has turned out to be very challenging. All known schemes that are suited for practical use have been broken in academia. Nevertheless, white-box cryptography is used in several products and companies such as Microsoft, Apple or Sony and many more have announced to deploy white-box techniques and/or have shown to have deployed such technology [Wys12]. This stresses the practical importance of the research in this field. The game of white-box schemes being developed, attacked and broken is an important process on the way to a secure design and can also bring new insights to basic cryptographic topics not only limited to white-box cryptography. For example, in his keynote talk about white-box cryptography at FSE 2016 , Henri Gilbert, the head of the Cryptography Laboratory at the French Network and Information Security Agency (ANSSI), called the results presented in Chapter 3 a breakthrough in structural cryptanalysis [Gil16].

1.3 Cryptanalysis of Block Ciphers

In cryptanalysis, we analyze block ciphers from the attacker's point of view. As mentioned before, nobody so far could find an efficient block cipher that can be proven to be secure. However, when designing a new block cipher one should present a detailed security analysis with sound arguments why the cipher is secure against known attacks. Recall from Section 1.1.2 that any attack which is better than exhaustive search is considered a successful attack.

In the following, after shortly treating brute-force attacks, we present the basic idea using a distinguishing property to recover the key. We then introduce the most important attacks for block ciphers. Using the example of the SPN design introduced in Section 1.2.1, we will see how concrete security arguments against single attacks can be derived.

1.3.1 Brute-Force Attack

The brute-force attack is also called exhaustive search and works with only a few plaintext-ciphertext pairs (m_i, c_i). The attacker searches for the single key $k \in \mathbb{F}_2^\kappa$ such that $c_i = E_k(m_i)$ for all i. Recall that a cipher resembles a random family of permutations and thus the probability of m_i being encrypted to c_i under a fixed wrong key should be 2^{-n} where n is the block size. Thus, indeed a few plaintext-ciphertext pairs will suffice to find the key. In the worst case, the attacker needs to search through all 2^κ keys.

The only way to make a block cipher withstand this attack (besides the not recommendable way of constructing a really bad cipher that behaves similarly under different keys) is to choose κ so large that a brute-force attack becomes practically infeasible. As noted above, attacks with complexity larger then 2^κ will not be considered because one could just as well run a brute-force attack.

1.3.2 Using distinguishers for Key-Recovery

All the more sophisticated attacks presented in the remainder of this section are distinguishing attacks. That is, they find a property of (parts of) the block cipher which can be used to distinguish

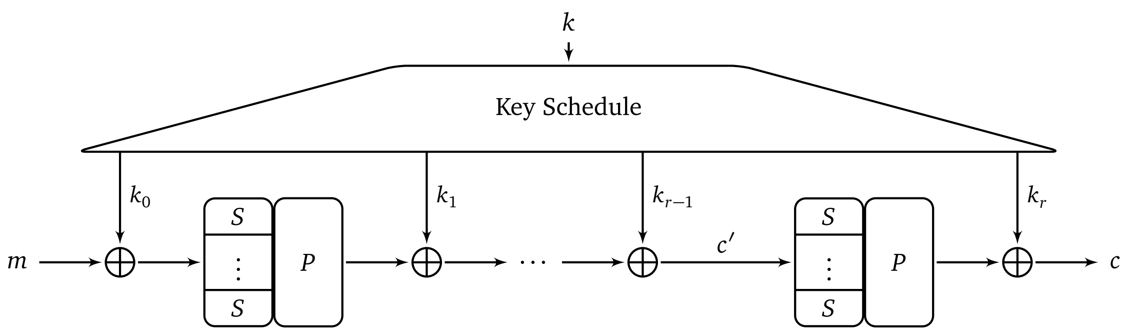


Figure 1.6: *The whole substitution-permutation network (SPN) computes $c = E_k(m)$. The result after $r-1$ rounds is $c' = E'_k$.*

it from a random permutation. Based on the example of an SPN, we will now explain how such a distinguisher can be used to recover the key.

As also shown in Fig. 1.6, let E_k be an r-rounds SPN and let E'_k be the $(r-1)$-rounds SPN obtained from E_k by removing the last round. We assume to have a distinguisher over E'_k, that is, we have observed a correlation between m and $c' = E'_k(m)$ that allows for distinguishing E'_k from a random permutation if we have enough pairs (m, c') at our disposal. Of course, as an attacker, we can only query pairs of the form (m, c), but we can then guess the last round key and derive $(m, \tilde{c})$. If the guessed round key was incorrect, the distinguisher will most probably not work and we can discard the key candidate. If the guessed round key was correct, we have $\tilde{c} = c'$ and the distinguisher will work as expected and we have successfully recovered the last round key. Given a round key, it is often possible to easily compute all the other round keys and the master key from the key schedule. If this is not possible, the next step would be to apply the same kind of attack to derive the second last round key and so on. (Note that recovery of all round keys is as bad as recovery of the master key.) However, looking at Fig. 1.6 we note that the size of a round key is equal to the block size and it is actually often also equal to the size of the master key which defines the complexity of a brute-force attack. Thus, in these cases, the above presented key-recovery attack is not better than a brute-force attack and does not harm the security of the cipher. An attacker can avoid this high complexity by finding a distinguishing property that only depends on *some* bits of c' which in turn only depend on *some* bits of k_r. Then, only the according key bits of the round key need to be guessed. The remaining bits are then recovered by repeatedly applying similar attacks or by exhaustively searching the now reduced set of possible round keys.

This attack can of course also be used to recover the first round key instead of the last one. Actually, it is not unusual to recover the first and the last round key at the same time. This might be done if the attacker wants to break a r-round cipher, but only has a $(r-2)$-round distinguisher. He then guesses parts of the first round key and parts of the last round key at once.

It depends on the specific distinguisher if known-plaintext pairs are sufficient or if chosen-plaintexts/chosen-ciphertexts need to be used. For example, the two most prominent distinguishing techniques differential cryptanalysis (Section 1.3.3) and linear cryptanalysis (Chapter 2) are CPAs/CCAs and KPAs, respectively.

In general, for an increasing number of rounds, it gets more and more difficult to find a distinguisher. Hence, the design consequence of this attack is straight forward: *use enough rounds*. In the design process of a block cipher, a thorough security analysis should be carried out. This way, one can get an idea of how many rounds can be attacked. One then adds more rounds to be sure that the cipher also withstands further attacks in the future. These additional rounds are called the *security margin*. On the one hand, the security margin should be as large as possible because it increases the security of the cipher. On the other hand, additional rounds use additional computing resources and this should be avoided, especially for lightweight applications. While it is typically not possible to easily find an attack on some block cipher right away, it might be possible to decrease the security margin. This way, cryptanalysis of block ciphers often is an incremental process in which more and more rounds can be attacked until finally no more progress is made or the cipher is broken.

1.3.3 Differential Cryptanalysis

Differential cryptanalysis was introduced in 1991 by Biham and Shamir [BS91] and is one of the most important cryptanalytic techniques. The basic idea is to look at the difference between two plaintexts and make statements about the expected difference between the according two ciphertexts. More formally, the goal of the attacker is to find a differential with high probability were a differential and its probability is defined as follows:

Definition 1.6 (Differential and its Probability)**.** Given a block cipher $E_k : \mathbb{F}_2^n \to \mathbb{F}_2^n$, a *differential* is a pair of differences (α, γ) with $\alpha, \gamma \in \mathbb{F}_2^n$. Let $\mathbf{X}$ be a random variable that takes any value in $\mathbb{F}_2^n$ with uniformly distributed probability. The probability of the differential is defined as:

$$\Pr_{\mathbf{X}}[\alpha \underset{\mathbf{X}}{\overset{E_k}{\to}} \gamma] := \Pr_{\mathbf{X}}[E_k(\mathbf{X}) + E_k(\mathbf{X} + \alpha) = \gamma].$$

α is called the input difference and γ is called the output difference. ◊

Note that the differences are computed by addition instead of subtraction because these operations are the same in $\mathbb{F}_2^n$. A differential with high probability can be used as a distinguisher, because, from a random permutation, we would expect that for any two inputs the according outputs are not related, and thus, given any non-zero input difference, the output difference should not be predictable. Therefore, it is desirable to upper bound the differential probability when designing a cipher. This would give a security guarantee against differential cryptanalysis. Unfortunately, going through all the 2^{2n} differentials is not feasible for typical block sizes like $n = 128$. Instead, attackers and designers are looking at how a given input difference propagates through a cipher. This leads to the notion of a differential characteristic for round-based ciphers.

Definition 1.7 (Differential Characteristic)**.** Given an r-round block cipher with a block size of n bits, a differential characteristic, also called a differential trail, is an $(r+1)$-tuple $(\theta_0, \theta_1, \ldots, \theta_r)$ with elements $\theta_i \in \mathbb{F}_2^n$. The first difference θ_0 is called the input difference and the last difference θ_r is called the output difference. ◊

The idea of a differential characteristic is to describe the difference not only between two plaintexts and the according ciphertexts, but also between all the intermediate results computed after every round. More precisely, let $(m_0, m_1, \dots, m_r)$ describe the intermediate results of a plaintext m that is processed by the cipher. That is, m_i denotes the plaintext processed for i rounds. In particular $m_0 = m$ and $m_r = E_k(m)$. Now, let $(m'_0, m'_1, \dots, m'_r)$ describe the intermediate results of a second plaintext m'. Then the according differential characteristic is given as $(m_0 + m'_0, m_1 + m'_1, \dots, m_r + m'_r)$. The probability of a differential characteristic for some round-based block cipher is the probability that two randomly drawn plaintexts with the given input difference yield exactly this characteristic.

Definition 1.8 (Probability of a Differential Characteristic)**.** Given an r-round block cipher $F = G_{r-1} \circ \cdots \circ G_1 \circ G_0$ with $G_i : \mathbb{F}_2^n \to \mathbb{F}_2^n$. and a differential characteristic $(\theta_0, \theta_1, \dots, \theta_r)$ with elements $\theta_i \in \mathbb{F}_2^n$. Let $\mathbf{X}$ be a random variable that takes any value in $\mathbb{F}_2^n$ with uniformly distributed probability. The probability of the differential characteristic is defined as:

$$\Pr_{\mathbf{X}}[\theta_0 \overset{G_0}{\to} \theta_1 \overset{G_1}{\to} \dots \overset{G_{r-1}}{\to} \theta_r] := \Pr_{\mathbf{X}}[\theta_0 \underset{\mathbf{X}}{\overset{G_0}{\to}} \theta_1 \wedge \theta_1 \underset{G_0(\mathbf{X})}{\overset{G_1}{\to}} \theta_2 \wedge \cdots \wedge \theta_{r-1} \underset{G_{r-2}\circ\dots\circ G_0(\mathbf{X})}{\overset{G_{r-1}}{\to}} \theta_r] \tag{1.1}$$

◊

Please note that in the above definition, we deviated from the previously introduced notation by not explicitly writing out the key dependency for the sake of a simple notation.

Looking back at the more general definition of a differential, we can now see that a differential "contains" many characteristics. Namely, a differential describes the possibility of somehow getting from an input mask to an output mask without considering any intermediate steps. The intermediate steps are only considered in the according characteristics. In particular, the probability of a differential is the sum of the probabilities of all characteristics with the respective input and output masks:

Theorem 1.1. *Given an r-round block cipher $F = G_{r-1} \circ \cdots \circ G_1 \circ G_0$ with $G_i : \mathbb{F}_2^n \to \mathbb{F}_2^n$, it holds:*

$$\Pr_{\mathbf{X}}[\alpha \underset{\mathbf{X}}{\overset{F}{\to}} \gamma] = \sum_{\substack{\theta_i \in \mathbb{F}_2^n \\ \theta_0 = \alpha, \theta_r = \gamma}} \Pr_{\mathbf{X}}[\theta_0 \overset{G_0}{\to} \theta_1 \overset{G_1}{\to} \dots \overset{G_{r-1}}{\to} \theta_r] \tag{1.2}$$

◊

Block cipher designers would like to upper bound the probability of a differential to prove that the cipher resists differential cryptanalysis. As discussed before, this fails due to the high complexity of the problem. Instead, the best we can do is to upper bound the probabilities of the differential characteristics. Of course, we might be unlucky and such small probabilities still might add up to a big probability in Eq. (1.2). However, the practice has shown that this is a sound design idea and often the high probability differentials are dominated by only one or a few high probability characteristics. Also, the attacker is in the same unpleasant situation as the designer, not being able to compute the differential probabilities and thus being forced to search for high probability characteristics. Such an attack is of course properly thwarted by upper bounding the probabilities of the characteristics.

But when we look at Eq. (1.1), we note that this equation is not suited to be used for practical cryptanalysis either. However, in a so-called *Markov cipher* [LMM91], Eq. (1.1) can be simplified to the product of the probabilities of the single round differentials, which can be practically computed for many ciphers.

Definition 1.9 (Markov cipher). An r-round block cipher $F = G_{r-1} \circ \cdots \circ G_1 \circ G_0$ is called a Markov cipher if, for every round G_i with a uniformly chosen round key, the differential probability is independent of the choice of the input text(s). ◊

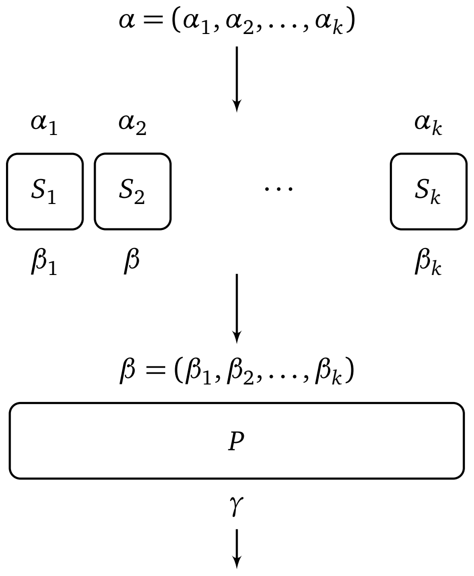
In other words, in any round of a Markov cipher with uniform and independent round keys, the round differential does only depend on the input difference. It does not matter which specific pair is used to construct this difference. Looking again at Eq. (1.1), we can now deduce the following corollary:

Corollary 1.1. *Given an r-round Markov cipher $F = G_{r-1} \circ \cdots \circ G_1 \circ G_0$ with uniformly chosen independent round keys with $G_i : \mathbb{F}_2^n \to \mathbb{F}_2^n$ and a differential characteristic $(\theta_0, \theta_1, \ldots, \theta_r)$ with elements $\theta_i \in \mathbb{F}_2^n$. The probability of the differential characteristic is:*

$$\begin{aligned}\Pr[\theta_0 \xrightarrow{G_0} \theta_1 \xrightarrow{G_1} \ldots \xrightarrow{G_{r-1}} \theta_r] &= \Pr[\theta_0 \xrightarrow{G_0} \theta_1 \wedge \theta_1 \xrightarrow{G_1} \theta_2 \wedge \ldots \wedge \theta_{r-1} \xrightarrow{G_{r-1}} \theta_r] \\ &= \prod_{i=0}^{r-1} \Pr[\theta_i \xrightarrow{G_i} \theta_{i+1}]\end{aligned}$$

◊

Above, we omitted the random variables because the according dependency from Eq. (1.1) has vanished and this leads to a simpler notation. Famous examples of Markov ciphers are AES or DES. All classical SPN ciphers as they are discussed in this thesis are Markov ciphers. However, the subkeys of such ciphers are not uniformly chosen and independent, but derived from a single master key by means of a key schedule. One therefore typically just assumes that the cipher shows a similar behavior when instantiated with the key schedule or with independent round keys. This is called the hypothesis of independent round keys. Furthermore, so far we considered probabilities of differentials and characteristics over the choice of messages and keys. However, when attacking a concrete instance of a cipher, the key is fixed. Here, the hypothesis of stochastic equivalence is used. It states that for virtually all keys, the differential trail probability can be approximated by averaging over all possible keys. In [DR02, Section 8.7.2], the authors point out two problems of this theory and the according assumptions. First, the conditions are sufficient, but not necessary, and it is often misunderstood as an incentive to avoid simple non-random key schedules. In fact, we show in Chapter 2, that also very simple linear key schedules can lead to strong designs. The second problem is that there are known practical examples of ciphers with big amounts of weak keys, contradicting the hypothesis of stochastic equivalence. Fortunately, the attacker of course does not know which key was used and typically also builds on the theory of averaging over all keys. And as long as the probability of a weak key being chosen is very small, the according attacks are not practically relevant. But still, it is highly unsatisfying from a design perspective to know that one is building on incorrect assumptions. Therefore, one should try to avoid these kinds of problems when designing a cipher. The hypothesis of independent round keys is critically analyzed for example in [Abd+12] and also in this thesis in Chapter 2. Despite these objections, in most cases, the Markov theory and the corresponding assumptions have turned out



$\alpha = (\alpha_1, \alpha_2, \ldots, \alpha_k)$
α_1
α_2
α_k
S_1
S_2
$\cdots$
S_k
β_1
β
β_k
$\beta = (\beta_1, \beta_2, \ldots, \beta_k)$
P
γ

Figure 1.7: *Propagation of a differential $\alpha \to \beta \to \gamma$ through a round of an SPN cipher.*

to be a sound approach leading to strong ciphers. Looking at Corollary 1.1, we can now see how a cipher designer finds an upper bound for the probability of the differential characteristics of the cipher. First, upper bounds for the probability of the single-round differentials are computed. Then, the upper bound for the probability of the differential characteristics is computed as the product of the single-round upper bounds. This process can be additionally simplified by using an iterated cipher where the bounds for the differential characteristics in every round are the same.

Using the example of an SPN cipher, we will now show how to compute the upper bound for the probability of a single-round differential. We recall from Section 1.2.1 that a round of an n-bit SPN cipher consists of a layer of t parallel m-bit S-boxes followed by a linear transformation. Let us assume the very common case in which all S-boxes are the same which again simplifies the analysis. Now let α be the input difference of the S-box layer, let θ be the difference between the S-box layer and the linear layer, and let γ be the output difference of this round. According to the inputs and outputs of the S-boxes, we can view $\alpha \in \mathbb{F}_2^n$ as $(\alpha_1, \ldots, \alpha_t) \in (\mathbb{F}_2^m)^t$ and β as $(\beta_1, \ldots, \beta_t) \in (\mathbb{F}_2^m)^t$ as shown in Fig. 1.7. As the S-boxes process the data independently, the probability of a differential through the S-box layer can be computed as the product of the probabilities of the according differentials through the single S-boxes.

$$\Pr[\alpha \to \beta] = \prod_{i=1}^{t} \Pr[\alpha_i \to \beta_i] \tag{1.3}$$

The trivial case of maximizing this probability is to choose the same inputs and thus having differences $\alpha_i = \beta_i = 0$ for every S-box, resulting in an overall probability of one. If we exclude this trivial case, the probability can be maximized by choosing $\alpha_i = \beta_i = 0$ for all but one single S-box difference. This S-box is then called *active*.

Definition 1.10 (Active S-boxes)**.** In differential cryptanalysis, S-boxes with an input difference unequal to zero are called *active*. Non-active S-boxes are called passive. $\Diamond$

For the active S-box, a non-trivial differential with maximum differential probability needs to be chosen. To this end, we can use the difference distribution table (DDT) of the S-box.

Definition 1.11 (Difference Distribution Table (DDT))**.** Let $f : \mathbb{F}_2^m \to \mathbb{F}_2^m$. The DDT of f is a table that contains the probabilities of all differentials through f. $\Diamond$

As discussed at the beginning of this section, with 2^{2n} steps, the complexity of computing the DDT for the whole cipher is far too high. However, for an S-box, it is feasible. The DDT of an m-bit S-box contains 2^{2m} entries. It is typically presented as a matrix in which each row corresponds to an input difference and each column corresponds to an output difference. Finally, assuming that all other S-boxes have a zero input difference, the highest entry in the DDT (excluding the trivial differential) gives us the upper bound for the one-round differential probability of the S-box layer. It can easily be checked that the subsequent linear transformation P with the input difference β produces the output difference $\gamma = P(\beta)$ with probability one. Thus, the aforementioned upper bound holds for the whole round.

Above, we showed how to find an upper bound for a one-round differential in an SPN. Using the hypothesis of independent round keys and Corollary 1.1, this can be used to derive an upper bound for the whole cipher. In the analysis, solely the quality of the S-box plays a role. The upper bound can be decreased by using a good S-box with a small maximum entry in the DDT. An even smaller upper bound can be shown by also taking into account the linear layer. Above, we assumed that only a single S-box is active in every round of the cipher. However, a good linear layer ensures that many S-boxes are activated throughout the whole cipher. The design idea of activating as many S-boxes as possible by a good choice of the linear layer is called the *wide trail design strategy* [Dae95; DR01; DR02]. If multiple S-boxes are activated within a round, then the upper bound for that round is computed by multiplying the maximum probabilities given by the DDTs of the according S-boxes. Eventually, the upper bound for the whole cipher is computed as the product of the upper bounds for the single rounds. Thus, if the same S-box is used throughout the whole cipher and if it is an iterated cipher, then the upper bound can be computed as p_S^ℓ where p_S is the maximum differential probability of the S-box and ℓ is a lower bound for the number of active S-boxes. A commonly used way to derive a lower bound for the number of active S-boxes in the whole cipher is to use the *branch number* which gives the minimal number of active S-boxes within two rounds. Given an SPN with t S-boxes, the branch number cannot be greater than $t + 1$. This can be seen by looking at the case with only one active S-box in one of the two rounds. Linear layers with branch number $t + 1$ are called Maximum Distance Separable (MDS) layers and the matrices describing the according linear transformation are called MDS matrices. Such matrices are well-known from coding theory which provides various methods of constructing them. An overview can be found in Section 5.2.2. MDS matrices are one of the most important building blocks in block ciphers, the most famous example of a cryptographic scheme that uses an MDS matrix is the AES. The problem of finding efficient implementations for linear layers in general and for MDS matrices in particular has high relevance for lightweight implementations, for example in embedded systems and IoT devices. Chapters 4 and 5 of this thesis are discussing exactly this problem.

1.3.4 Boomerang Attacks and Integral Cryptanalysis

There exist multiple advanced methods based on differential cryptanalysis that might be particularly effective in special cases. Often, it turns out that such methods are related to each other or sometimes even essentially the same. For example, the two attacks that are presented in this section can both be seen as special versions of the so-called higher-order differential cryptanalysis [Lai94] which is, in turn, an advanced version of the previously described differential cryptanalysis. In the following, we shortly introduce boomerangs attacks and integral cryptanalysis.

Boomerang Attacks

The boomerang attack [Wag99] splits the cipher under attack into two halves such that $F = F_2 \circ F_1$. It is well-suited in situations where an attacker cannot find a good differential over the full cipher, but good differentials over the two halves. This attack uses two differentials. One over the encryption of the first half $\alpha \xrightarrow{F_1} \beta$ with probability p and another one over the decryption of the second half $\gamma \xrightarrow{F_2^{-1}} \delta$ with probability q. The boomerang attack is depicted in Fig. 1.8.[10] In the first step, the chosen plaintexts m_1 and $m_2 = m_1 + \alpha$ are used to compute $c_1 = F(m_1)$ and $c_2 = F(m_2)$. In the second step, the chosen ciphertexts $c_3 = c_1 + \gamma$ and $c_4 = c_2 + \gamma$ are used to compute $m_3 = F^{-1}(c_3)$ and $m_4 = F^{-1}(c_4)$. Finally, the condition $m_3 + m_4 = \alpha$ is used as a distinguisher. In this case, the difference α was "thrown into" the cipher as $m_1 + m_2$ and "comes back" as $m_3 + m_4$, giving this attack its name. Assuming the probabilities of the single differentials to be independent, the boomerang is successful with probability p^2q^2. To show this, let us denote the intermediate values between F_1 and F_2 with $x_i = F_1(m_i) = F_2^{-1}(c_i)$. First, the difference $m_1 + m_2 = \alpha$ leads to the difference $y_1 + y_2 = \beta$ with probability p. Then, the difference $c_1 + c_3 = \gamma$ leads to the difference $y_1 + y_3 = \delta$ with q. In the same way, $c_2 + c_4 = \gamma$ leads to the difference $y_2 + y_4 = \delta$ with q. Now we have $y_3 + y_4 = (y_1 + \delta) + (y_2 + \delta) = y_1 + y_2 = \beta$ and this leads to $m_3 + m_4 = \alpha$ with probability p. All in all, four differentials must hold for the boomerang. Two of them hold with probability p and the other two with probability q, which gives us the overall probability p^2q^2. Actually, we can notice that the value of the difference δ is not important for a successful attack. The relevant condition in this step is $y_1 + y_3 = y_2 + y_4$. Thus, the value of δ can vary which leads to a higher success probability.

There also exist advanced versions of this attack called *amplified boomerangs* and *rectangle attacks*. The amplified boomerang attack does not require chosen ciphertexts any more. It encrypts many pairs with input difference α and then uses the birthday paradox to estimate the probability of $y_1 + y_3 = \delta$ which also leads to $y_2 + y_4 + \delta$. As a result, we have a pure chosen-plaintext attack at the cost of a higher complexity. A further extension of the amplified boomerang attack is the rectangle attack. Here, the observation is that actually only α and δ are fixed. As in the usual case for differentials, all intermediate values may vary, this also holds for β.

It should be mentioned that Murphy showed that the probability analysis used in the basic and advanced attacks can be highly inaccurate and that according complexity estimates must be treated with caution [Mur11]. Therefore, as always in symmetric cryptology when assumptions

[10]Fig. 1.8 is based on TikZ code from [Jea17].

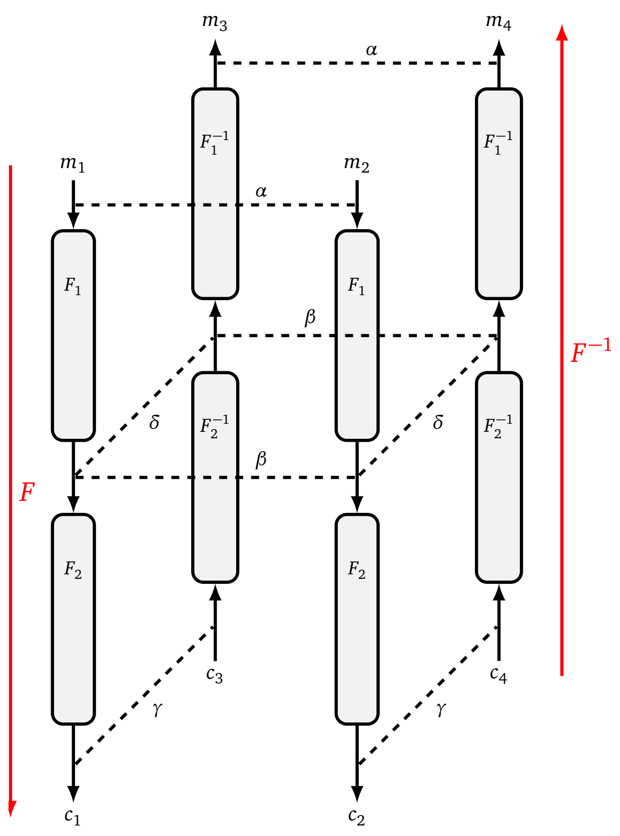


Figure 1.8: *The boomerang attack.*

are used, additional practical experiments should be used to back up the theoretical results whenever possible.

The boomerang attacks and its advanced versions are used in Chapter 3 as one of several possibilities to attack a white-box construction.

Integral Cryptanalysis

Integral cryptanalysis belongs to the so-called structural attacks. That is, attacks that make use of the structure of cipher rather than the concrete building blocks. Such attacks will still work when the cipher is instantiated with other building blocks. The attack was first described in [DKR97] and later more generally explained in [KW02]. An integral attack uses a specifically chosen set of plaintexts that sum up to zero in certain bits. By analyzing the propagation of this zero-sum property, one can build distinguishers over several rounds of modern ciphers. We will explain this in the following, by presenting an integral attack on a reduced-round version of the AES. In the literature, this attack is often called the SQUARE attack, because it was first presented as an attack on the cipher SQUARE [DKR97], a predecessor of the AES.

The AES [AES01] is a 128-bit block cipher. The 128 bits are arranged in a 4×4 matrix where each element constitutes a byte. It is a key-alternating cipher where each (but the last) round

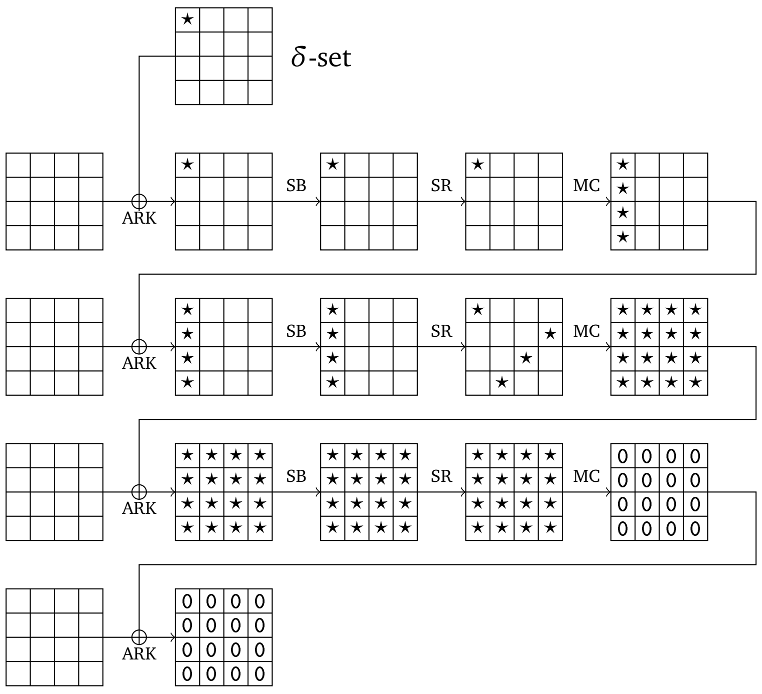


Figure 1.9: *An integral attack on 3 rounds of AES.*

consists of three steps:

1. An S-box is applied to each of the 16 bytes. This step is called *SubBytes* (SB).
2. The i-th row is rotated left by i steps. This step is called *ShiftRows* (SR).
3. Each byte is represented by an element of the finite field $\mathbb{F}_{2^8}$. Now, each column is multiplied by a fixed 4×4 matrix over $\mathbb{F}_{2^8}$. This step is called *MixColumns* (MC).

The key additions between the rounds are called *AddRoundKey* (ARK). All details including the representation of the finite field, the specification of the S-boxes, and the key schedule can be found in [AES01].

The integral distinguisher over 3 rounds of AES is shown in Fig. 1.9[11]. It uses a set of 2^{16} plaintexts. Here, one byte takes all possible 2^{16} values while the other 15 bytes are constant. The property of a byte taking all 2^{16} values is noted with a '$\star$'. This can also be used as a distinguisher. In fact, it implies the property that the sum will be zero in the according byte, which is denoted by a '0'. A blank space means that the according value is constant for all texts.

The initial state with the 2^{16} plaintexts in one byte is shown at the top. First, the ARK step does not change the properties because it is an addition with a constant which is a bijective operation. Next, the SR step also does not change anything since the first row is not affected. Writing out the MC step as a multiplication with a constant matrix, one can easily see that the

[11]Fig. 1.9 is based on TikZ code from [Jea17].

resulting first column takes every value in every row. In the same way, we can easily follow the propagation of the properties until we reach the MC step of the third round. Now, writing out the MC step for any byte we have $y = Mx$, where M is the matrix and x and y are the input and output columns, respectively. Computing the sum over the output column for all 2^{16} texts we have $\sum y = \sum Mx = M \sum x = 0$. Thus, adding the 2^{16} outputs after three rounds will result in only zeros. This distinguisher holds with probability 1 and can be used as a basis for attacks over even more rounds.

Just like boomerang attacks, the technique of integral cryptanalysis is also used in Chapter 3 to attack a white-box construction.

1.3.5 Linear Cryptanalysis

In many ways, linear cryptanalysis [Mat94] is similar to differential cryptanalysis. But instead of differences, linear functions are used. That is, rather then checking if a certain input difference leads to a certain output difference, one checks the correlations between a linear function of the input and a linear function of the output. Whereas we talk about differentials in differential cryptanalysis, we talk about *linear hulls* in linear cryptanalysis. As in the differential case, a linear hull is composed of many linear characteristics, also called linear trails. And as in the differential case, the wide trail design strategy can be applied to upper bound the probabilities of these characteristics. This is done by using linear approximation tables (LATs) instead of difference distribution tables (DDTs). However, there are also important differences. The main theorem of linear cryptanalysis (see Section 2.2) can be proven without any need for assumptions like the hypothesis of independent round keys. Furthermore, biases are used instead of probabilities. As a consequence, the effect of multiple linear trails within a linear hull might cancel each other out.

The basics of linear cryptanalysis are described within the next chapter. There, we also dive deeper into some more advanced topics like the influence of the choice of the key schedule. In the end, we show an important difference between linear and differential cryptanalysis when it comes to proving security for the aforementioned tweakable block ciphers.

Part I

Linear Cryptanalysis

2

Linear Cryptanalysis: Key Schedules and Tweakable Block Ciphers

The results from this chapter have been published in the IACR journal *Transactions on Symmetric Cryptology* [KLW17] and presented at the conference *Fast Software Encryption 2017* in Tokyo, Japan. This is joint work with Gregor Leander and Friedrich Wiemer. All authors equally contributed. The main contributions of the thesis author are in the systematization of linear cryptanalysis (Section 2.2) and in the findings about linear key schedules (Section 2.4) and tweakable block ciphers (Section 2.5).

2.1 Introduction

Linear cryptanalysis, introduced by Matsui [Mat94], is one of the major statistical attacks on block ciphers. Since its invention in the early 1990s, many extensions and variations have been considered. The most important theoretical investigation is certainly the work of Nyberg [Nyb95], where the concept of linear hulls was introduced and the assumption of round-independence needed in Matsui's original approach was clarified. Similar results can be derived by using the concept of correlation matrices, as done by Daemen and Rijmen [DR02]. A statistical model for estimating the data complexity of various linear attacks is presented in [BN17]. The concept of the linear hulls in particular shows the key-dependency of the correlation of a given linear approximation. Due to the key-dependency of the distribution, for a complete understanding of the security of a block cipher with respect to a linear attack, one has to understand not only one correlation, but rather the distribution of the correlations taken over all possible master keys. Only then one is able to estimate the fraction of weak keys, that is, keys such that the corresponding correlation is high enough to be exploitable in an attack. Following Nyberg's fundamental theorem, it is possible, at least theoretically, to compute the mean and the variance of this distribution in the case of independent round keys. But it seems hard to derive more information about the distribution. It would be especially interesting to derive bounds on the tails, i. e. the fraction of weak keys. Moreover, even for just estimating the variance in practice, the assumption of independent round keys is crucial [BN16] (while it is also possible to compute the variance without this assumption, see [BTV18], doing so in practice seems to be hard).

When it comes to the key schedule part, there is a clear lack of understanding in block cipher design. Let us quickly recall the role of the key schedule algorithm in a block cipher. The key schedule takes as input a master key (in the case of AES-128 this is a 128-bit string) and outputs so-called round keys that are used in each round to mix the current state with the key (most often by simply XORing the round key to the state). In the case of AES-128, the total length of the round keys is $11 \cdot 128 = 1408$ bits, and thus the AES key schedule, as a function, is a mapping from $\{0,1\}^{128}$ to $\{0,1\}^{1408}$.

However, as already motivated in Section 1.2.1, the influence of the key schedule on linear and differential attacks is to a large extent unknown. One usually assumes that all (round) keys are independently and uniformly chosen. While this is hardly the case for any real cipher, this assumption is on the one hand needed to make the analysis feasible and on the other hand often does not seem problematic as even with the keys not independently and uniformly chosen, most ciphers (experimentally!) do not behave differently from the expectation.

Note that the above of course extends, and in some respect becomes even more relevant, when we consider the case of a tweakable block cipher and discuss how a suitable tweak schedule should be constructed. This analogy is maybe most obvious in the TWEAKEY setup [JNP14b] where key and tweak are just parts of the same object, but is certainly important for any kind of tweak schedule.

In this chapter, we aim to systemize the theoretical notions underlying linear cryptanalysis. Furthermore, we take some steps forward to increase our understanding of the influence of key and tweak schedules on the security of (tweakable) block ciphers with respect to linear attacks.

Our Contributions

Systematization of Linear Cryptanalysis

We begin by presenting the idea of linear cryptanalysis in Section 2.2. By doing so, we try to express all terms as Fourier coefficients instead of using correlations, as we feel that this gives a more clear picture. This perspective turns out to be especially nice when it comes to the correlation of a linear trail. In many papers on linear cryptanalysis, the correlation of a linear trail is either not well-defined (when using the piling-up lemma) or not well-motivated (when given as a pure definition). However, in our setup, the correlation of the linear hull nicely corresponds to a Fourier coefficient. Note that this perspective is implicitly already contained in Nyberg's original paper on the linear hull [Nyb95], but we feel that it did not get the attention it deserves.

To support the general understanding, we develop the Fourier coefficient of the linear hull by first considering a generic key-dependent block cipher E_k, then specializing it to a round-based structure and finally to the most commonly used key-alternating case. While the corresponding proofs involve only basic techniques and may have appeared elsewhere, we nevertheless include them again in order to preserve their educational value and to aid researchers unfamiliar with the topic to gain a better understanding. Building on these fundamentals, we then turn to key schedules.

Bizarre Examples: The Tail of the Distributions

First, we start by exploring how the key schedule can influence the distribution of Fourier coefficients. This first part, in Section 2.3 builds upon the example on PRESENT from [Abd+12]. Besides the result of Abdelraheem et al., many papers cover experiments on PRESENT – to name just a few: [BPW15; BTV18; Cho10; Hua+15]. As observed in [Ohk09] the distribution of the correlation for PRESENT follows closely a normal distribution with mean zero. Moreover, the variance of this distribution fits to what can be expected for independent round keys. The observation in [Abd+12] was that, when replacing the key schedule of PRESENT by a key schedule that produces identical round keys in every round, the variance increases significantly. This, in particular, means that the cipher becomes weaker against linear cryptanalysis, as the fraction of keys that have a large correlation (in absolute terms) increases significantly (see Fig. 2.4). However, although the variance increases, the distribution still follows a normal distribution closely.

By doing extensive experiments with a large set of variants of the PRESENT cipher, we eventually observe many interesting examples of how the key schedule can influence the distribution of Fourier coefficients in a much more dramatic manner. While we show several of those distributions in the appendix, the main interesting conclusion actually follows from an example depicted in Fig. 2.7. Recall from above that one important question is if we can prove stronger statements about the number of keys with a large absolute Fourier coefficient, beyond what is given by Tchebysheff's general upper bound on any distribution. Now, the example we found leads to a negative conclusion. That is, in general, it seems that we cannot hope to prove any stronger statements. This follows from the fact that, when increasing the number of rounds, the distribution of the example corresponding to Fig. 2.7 tends to (the unique) distribution for which Tchebysheff's general upper bound is tight. In other words, there exist Fourier coefficient distributions with mean zero and variance σ^2, such that the fraction of keys with an absolute correlation greater than or equal to σk equals $\frac{1}{k^2}$.

Linear Key Schedule

The next contribution leads to a much more positive, constructive result. Here, in Section 2.4 we focus on the case of a linear key schedule. Linear key schedules are very common in block ciphers. Besides the DES, many lightweight ciphers actually use the easiest possible linear key schedule, i. e. simply use identical round keys. In order to avoid structural attacks, in particular slide attacks, and in order to break symmetries, it is common sense to add varying round constants to every round key. Now, in Section 2.4 we prove that any linear key schedule is sound, concerning linear cryptanalysis, in the following sense: For any given linear key schedule, the average variance of the distribution of the Fourier coefficients, taken over all possible round constants, is exactly the same as for independent round keys. Thus, as a designer, after fixing any linear key schedule of one's choice, one can expect that when adding a randomly chosen set of round constants, the distribution of the Fourier coefficients closely follows the one in the case of independent round keys. This adds some theoretical foundation on the hypothesis of independent round keys criticized above in the case of linear key schedules. We back up this theoretical observation by experiments on PRESENT.

Tweakable Block Ciphers

Finally, we turn our attention to tweakable block ciphers and how the additional input, i. e. the tweak, possibly helps an attacker. The main possible advantage of the tweak, when it comes to linear cryptanalysis, is that instead of approximating a linear function of the ciphertext by a linear function of the plaintext only, the attacker can now try to approximate (a linear function of) the ciphertext by a linear function of the plaintext *and* a linear function of the tweak.

To study this potentially new attack vector, we elaborate on the linear hull of a tweakable block cipher. We look at the case of a linear tweak schedule and later specialize on tweak-alternating and key-alternating block ciphers. It turns out that the linear hull, and therefore the Fourier coefficient an attacker can use, is composed of the same linear trails as in the non-tweaked case. In other words, by adding the tweak, no new linear characteristics are introduced. Thus, protecting a tweakable cipher with linear tweak schedule against linear cryptanalysis basically does not need any additional considerations, but can be done in the same way as it is done for a non-tweakable block cipher, i. e. by upper bounding the Fourier coefficient of any single linear characteristic. Note that this is in sharp contrast to the differential case, where using a difference in the tweak often leads to differential characteristics with a significantly higher probability.

We like to clearly mention that, from a technical point of view, we mainly reuse standard approaches. Still, our results shed some new light on the wide-open field of the design of a sound key schedule.

2.2 Systematization of Linear Cryptanalysis

In the course of this section, we develop in a step by step manner the setting of linear cryptanalysis in a general and consistent way. Within our systematization, we also highlight the meaning of a linear trail as this seems to be not well-known.

Recall, that we denote by $\mathbb{F}_2$ the finite field with two elements and by $\mathbb{F}_2^n$ the n-dimensional vector space over $\mathbb{F}_2$, i. e. the set of all n-bit strings together with the bitwise XOR-addition. When dealing with linear cryptanalysis, we need to define a scalar product on $\mathbb{F}_2^n$. For $x, y \in \mathbb{F}_2^n$ by $\langle x, y\rangle$ we denote the canonical scalar product, i. e. $\langle x, y\rangle := \sum x_i y_i$. We will often deal with linear mappings on $\mathbb{F}_2^n$ and, given a linear mapping $L : \mathbb{F}_2^n \to \mathbb{F}_2^n$ we denote by L^T its adjoint linear mapping, i. e. the mapping such that

$$\left\langle x, L(y)\right\rangle = \left\langle L^T(x), y\right\rangle \quad \forall x, y \in \mathbb{F}_2^n.$$

Note that, when L is given as an $n \times n$ binary matrix, then L^T is nothing else than the linear mapping corresponding to the transposed matrix.

Linear Cryptanalysis

Next, we present the basic concepts of linear cryptanalysis. For this, let

$$E_k : \mathbb{F}_2^n \to \mathbb{F}_2^n$$

be a block cipher on n-bit blocks, indexed by a key k. In classical linear cryptanalysis, we try to approximate a linear Boolean function of the output $E_k(x)$ by a linear Boolean function of the input x. More precisely, we search for a pair of *input and output masks* (α, γ), such that the bias of the linear approximation

$$\langle \gamma, E_k(x) \rangle \approx \langle \alpha, x \rangle$$

is large in absolute terms. We define the bias $\epsilon_{E_k}(\alpha, \gamma)$ by

$$\Pr_x \left[\langle \gamma, E_k(x) \rangle = \langle \alpha, x \rangle \right] = \frac{1}{2} + \epsilon_{E_k}(\alpha, \gamma),$$

and to make linear cryptanalysis successful we have to choose α and γ such that $|\epsilon_{E_k}(\alpha, \gamma)|$ is large since the linear approximation can then be used as a distinguishing property. Due to scaling issues, it is often more convenient to work with the correlation $c_{E_k}(\alpha, \gamma) := 2\epsilon_{E_k}(\alpha, \gamma)$ instead of the bias directly.

In this chapter, however, we are mainly working with the Fourier (or Walsh) transformation of E_k. The Fourier coefficient of a vectorial Boolean function $f : \mathbb{F}_2^{n_1} \to \mathbb{F}_2^{n_2}$ at position $\alpha \in \mathbb{F}_2^{n_1}$ and $\gamma \in \mathbb{F}_2^{n_2}$ is defined as

$$\widehat{f}(\alpha, \gamma) := \sum_{x \in \mathbb{F}_2^{n_1}} (-1)^{\langle \alpha, x \rangle + \langle \gamma, f(x) \rangle}.$$

In terms of linear cryptanalysis, the Fourier coefficient of E_k is nothing else than a scaled version of the bias (and therefore nothing else than a scaled version of the correlation). More precisely it holds that

$$\widehat{E_k}(\alpha, \gamma) = 2^n c_{E_k}(\alpha, \gamma) = 2^{n+1} \epsilon_{E_k}(\alpha, \gamma).$$

Next, we will present some very useful lemmas about the Fourier coefficient that will be used in the subsequent proofs and in general provide a valuable set of tools for proofs regarding linear cryptanalysis. The first one was proven by Nyberg [Nyb01, Theorem 3].

Lemma 2.1 (Consecutive Functions).

Given

$$f : \mathbb{F}_2^{n_1} \times \mathbb{F}_2^{\ell} \to \mathbb{F}_2^{n_2}, \quad g : \mathbb{F}_2^{n_1} \times \mathbb{F}_2^{\kappa} \to \mathbb{F}_2^{n_2}, \quad h : \mathbb{F}_2^{\ell} \to \mathbb{F}_2^{\kappa},$$
$$f(x, y) := g(x, h(y)).$$

Then

$$2^{\kappa} \widehat{f}((\alpha, \beta), \gamma) = \sum_{\beta' \in \mathbb{F}_2^{\kappa}} \widehat{g}\big((\alpha, \beta'), \gamma\big) \cdot \widehat{h}\big(\beta, \beta'\big).$$

◊

Proof. We only need the well-known fact that for the dot product it holds:

$$\sum_{\beta \in \mathbb{F}_2^n} (-1)^{\langle \beta, x \rangle} = \begin{cases} 2^n & \text{, if } x = 0 \\ 0 & \text{, else} \end{cases} .$$

Hence:

$$\begin{aligned} \sum_{\beta' \in \mathbb{F}_2^\kappa} \widehat{g}\big((\alpha, \beta'), \gamma\big) \cdot \widehat{h}\big(\beta, \beta'\big) &= \sum_{\beta'} \sum_{\substack{x \in \mathbb{F}_2^{n_1} \\ y \in \mathbb{F}_2^\kappa}} (-1)^{\langle \alpha, x \rangle + \langle \beta', y \rangle + \langle \gamma, g(x,y) \rangle} \sum_{z \in \mathbb{F}_2^\ell} (-1)^{\langle \beta, z \rangle + \langle \beta', h(z) \rangle} \\ &= \sum_{x,y,z} (-1)^{\langle \alpha, x \rangle + \langle \beta, z \rangle + \langle \gamma, g(x,y) \rangle} \sum_{\beta'} (-1)^{\langle \beta', y + h(z) \rangle} \\ &= 2^\kappa \sum_{x,z} (-1)^{\langle \alpha, x \rangle + \langle \beta, z \rangle + \langle \gamma, g(x, h(z)) \rangle} \\ &= 2^\kappa \widehat{f}((\alpha, \beta), \gamma) \end{aligned}$$

□

The next lemma was discussed by Daemen *et al.* [DGV95, Eq. (15)] and describes the Fourier coefficient of a composite function.

Lemma 2.2 (Function Composition)**.**

Given

$$f : \mathbb{F}_2^{n_1} \to \mathbb{F}_2^{n_3}, \quad g : \mathbb{F}_2^{n_1} \to \mathbb{F}_2^{n_2}, \quad h : \mathbb{F}_2^{n_2} \to \mathbb{F}_2^{n_3}, \\ f := h \circ g$$

Then

$$2^{n_2} \widehat{f}(\alpha, \gamma) = \sum_{\beta \in \mathbb{F}_2^{n_2}} \widehat{g}(\alpha, \beta) \cdot \widehat{h}(\beta, \gamma).$$

◊

Proof.

$$\begin{aligned} \sum_{\beta \in \mathbb{F}_2^{n_2}} \widehat{g}(\alpha, \beta) \cdot \widehat{h}(\beta, \gamma) &= \sum_{\beta} \sum_{x \in \mathbb{F}_2^{n_1}} (-1)^{\langle \alpha, x \rangle + \langle \beta, g(x) \rangle} \sum_{y \in \mathbb{F}_2^{n_2}} (-1)^{\langle \beta, y \rangle + \langle \gamma, h(y) \rangle} \\ &= \sum_{x,y} (-1)^{\langle \alpha, x \rangle + \langle \gamma, h(y) \rangle} \sum_{\beta} (-1)^{\langle \beta, y + g(x) \rangle} \\ &= 2^{n_2} \sum_{x} (-1)^{\langle \alpha, x \rangle + \langle \gamma, h(g(x)) \rangle} \\ &= 2^{n_2} \widehat{f}(\alpha, \gamma) \end{aligned}$$

□

We can easily prove a variant of this lemma for functions with another, independent input.

Lemma 2.3.

Given

$$f : \mathbb{F}_2^{n_1} \times \left(\mathbb{F}_2^{\kappa_1} \times \mathbb{F}_2^{\kappa_2}\right) \to \mathbb{F}_2^{n_3}, \quad g : \mathbb{F}_2^{n_1} \times \mathbb{F}_2^{\kappa_1} \to \mathbb{F}_2^{n_2}, \quad h : \mathbb{F}_2^{n_2} \times \mathbb{F}_2^{\kappa_2} \to \mathbb{F}_2^{n_3},$$
$$f(x,(y,z)) := h(g(x,y),z).$$

Then, for $\beta = (\beta_0, \beta_1)$

$$2^{n_2}\widehat{f}((\alpha,\beta),\gamma) = \sum_{\theta \in \mathbb{F}_2^{n_2}} \widehat{g}((\alpha,\beta_0),\theta) \cdot \widehat{h}((\theta,\beta_1),\gamma).$$

◊

Proof.

$$\begin{aligned}
\sum_{\theta \in \mathbb{F}_2^{n_2}} \widehat{g}((\alpha,\beta_0),\theta) \cdot \widehat{h}((\theta,\beta_1),\gamma) &= \sum_{\theta} \sum_{\substack{x \in \mathbb{F}_2^{n_1} \\ z \in \mathbb{F}_2^{\kappa_1}}} (-1)^{\langle \alpha,x \rangle + \langle \beta_0,z \rangle + \langle \theta,g(x,z) \rangle} \sum_{\substack{y \in \mathbb{F}_2^{n_2} \\ z' \in \mathbb{F}_2^{\kappa_2}}} (-1)^{\langle \theta,y \rangle + \langle \beta_1,z' \rangle + \langle \gamma,h(y,z') \rangle} \\
&= \sum_{\substack{x,y \\ z,z'}} (-1)^{\langle \alpha,x \rangle + \langle \beta_0,z \rangle + \langle \beta_1,z' \rangle + \langle \gamma,h(y,z') \rangle} \sum_{\theta} (-1)^{\langle \theta,y+g(x,z) \rangle} \\
&= 2^{n_2} \sum_{x,z,z'} (-1)^{\langle \alpha,x \rangle + \langle \beta_0,z \rangle + \langle \beta_1,z' \rangle + \langle \gamma,h(g(x,z),z') \rangle} \\
&= 2^{n_2}\widehat{f}((\alpha,\beta),\gamma)
\end{aligned}$$

□

Bogdanov and Rijmen [BR14, Lemma 1] studied how the XOR operation influences linear cryptanalysis.

Lemma 2.4 (XOR at input).

Given

$$g : \mathbb{F}_2^{n_1} \times \mathbb{F}_2^{n_1} \to \mathbb{F}_2^{n_2}, \textit{ and } f : \mathbb{F}_2^{n_1} \to \mathbb{F}_2^{n_2},$$
$$g(x,y) := f(x+y).$$

Then

$$\widehat{g}((\alpha,\beta),\gamma) = \begin{cases} 2^{n_1}\widehat{f}(\alpha,\gamma) & , \textit{if } \alpha = \beta \\ 0 & , \textit{else} \end{cases}.$$

◊

Proof.

$$\begin{aligned}
\widehat{g}((\alpha,\beta),\gamma) &= \sum_{x,y\in\mathbb{F}_2^{n_1}} (-1)^{\langle\alpha,x\rangle+\langle\beta,y\rangle+\langle\gamma,f(x+y)\rangle} \\
&= \sum_{x',y} (-1)^{\langle\alpha,x'+y\rangle+\langle\beta,y\rangle+\langle\gamma,f(x')\rangle} \\
&= \sum_{x',y} (-1)^{\langle\alpha,x'\rangle+\langle\alpha,y\rangle+\langle\beta,y\rangle+\langle\gamma,f(x')\rangle} \\
&= \sum_{x'} (-1)^{\langle\alpha,x'\rangle+\langle\gamma,f(x')\rangle} \sum_{y} (-1)^{\langle\alpha+\beta,y\rangle} \\
&= \widehat{f}(\alpha,\gamma)\cdot\sum_{y}(-1)^{\langle\alpha+\beta,y\rangle} \\
&= \begin{cases} 2^{n_1}\widehat{f}(\alpha,\gamma) & \text{, if } \alpha=\beta \\ 0 & \text{, else} \end{cases}
\end{aligned}$$

□

Lemma 2.5 (XOR at output).

Given

$$g:\mathbb{F}_2^{n_2}\times\mathbb{F}_2^{n_1}\to\mathbb{F}_2^{n_1},\ \textit{and}\ f:\mathbb{F}_2^{n_2}\to\mathbb{F}_2^{n_1},$$
$$g(x,y):=f(x)+y.$$

Then

$$\widehat{g}((\alpha,\beta),\gamma)=\begin{cases} 2^{n_1}\widehat{f}(\alpha,\gamma) & \textit{, if } \beta=\gamma \\ 0 & \textit{, else} \end{cases}.$$

◊

Proof. The proof works analogously to the proof of Lemma 2.4. □

As it is usually computationally infeasible to compute the (exact) Fourier coefficient of any reasonable block cipher E_k, we make use of the fact that almost all block ciphers are round-based. That is, E_k is then the composition of several (comparably simple) round functions $G_i:\mathbb{F}_2^n\to\mathbb{F}_2^n$. Those round functions are actually key-dependent, but in order to simplify notation, we ignore this key-dependency for now (and come back later to this topic extensively). So instead of computing the exact Fourier coefficient, or correlation, of a linear approximation, one usually focuses on what is called linear trail (synonymously often called linear path or linear characteristic). For an r-round cipher

$$E_k(x)=G_{r-1}\circ\cdots\circ G_1\circ G_0(x)$$

a linear trail θ is a collection of $r+1$ masks

$$\theta=(\theta_0,\theta_1,\ldots,\theta_r)$$

and the correlation of a trail is defined as

$$C_\theta := \prod_{i=0}^{r-1} c_{G_i}(\theta_i, \theta_{i+1}). \tag{2.1}$$

Initially, in his seminal work [Mat94], Matsui derived the correlation of a trail by the so-called piling-up lemma, assuming that the approximations of different rounds behave as independent Boolean random variables. Later, Nyberg [Nyb95] showed how this assumption can be avoided by introducing the concept of the linear hull. She also showed that Matsui's famous Algorithm 2, which he used to break DES, was actually making use of the linear hull and not of a single linear trail. This has also nicely been shown for iterated block ciphers by using the technique of correlation matrices [Dae95; DGV95; DR02]. We recall Nyberg's results in terms of the Fourier coefficients of E_k. The first and crucial idea is to consider E_k as a function in two variables, one being the plaintext and the second being the key. For a κ-bit key k, we consider

$$F : \mathbb{F}_2^n \times \mathbb{F}_2^\kappa \to \mathbb{F}_2^n$$

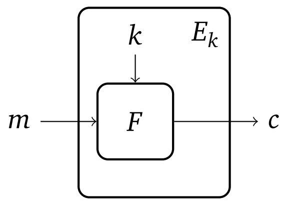
with

$$E_k(x) := F(x, k),$$

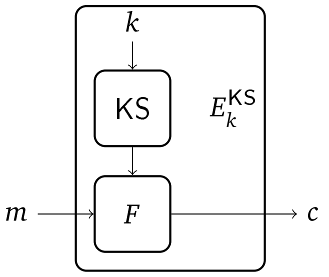
see also Fig. 2.1a. Nyberg basically showed that

$$2^\kappa \widehat{E_k}(\alpha, \gamma) = \sum_{\beta \in \mathbb{F}_2^\kappa} (-1)^{\langle \beta, k \rangle} \widehat{F}((\alpha, \beta), \gamma), \tag{2.2}$$

i. e. the Fourier coefficient of E_k corresponds to the (signed) sum of Fourier coefficients of F over all possible masks for the key-input. This is what is referred to as the *linear hull*. We recall Eq. (2.2) and its key scheduled variant in Proposition 2.1. In addition to the already mentioned results, Nyberg [Nyb01, Theorem 3] also covered this generic influence of a key schedule by the notation of one function having as an input the output of another function.



(a) *Generic key-dependent function E_k*



(b) *and its key scheduled variant E_k^{KS}*

Figure 2.1: *Most generic function.*

Proposition 2.1. *Let E_k and E_k^{KS} be the functions (see Fig. 2.1a and Fig. 2.1b)*

$$\begin{aligned} E_k &: \mathbb{F}_2^n \to \mathbb{F}_2^n & E_k^{\mathsf{KS}} &: \mathbb{F}_2^n \to \mathbb{F}_2^n \\ E_k(x) &:= F(x,k) & E_k^{\mathsf{KS}}(x) &:= E_{\mathsf{KS}(k)}(x) = F(x, \mathsf{KS}(k)) \end{aligned}$$

with $F : \mathbb{F}_2^n \times \mathbb{F}_2^\kappa \to \mathbb{F}_2^n$ and key schedule $\mathsf{KS} : \mathbb{F}_2^\ell \to \mathbb{F}_2^\kappa$. Then

$$2^\kappa \widehat{E_k}(\alpha,\gamma) = \sum_{\beta \in \mathbb{F}_2^\kappa} (-1)^{\langle \beta, k\rangle} \widehat{F}((\alpha,\beta),\gamma),$$

$$2^{\ell+\kappa} \widehat{E_k^{\mathsf{KS}}}(\alpha,\gamma) = \sum_{\substack{\beta \in \mathbb{F}_2^\ell \\ \beta' \in \mathbb{F}_2^\kappa}} (-1)^{\langle \beta, k\rangle} \widehat{\mathsf{KS}}(\beta,\beta') \widehat{F}\big((\alpha,\beta'),\gamma\big).$$

◊

Proof. We compute directly

$$\begin{aligned} \sum_\beta (-1)^{\langle \beta,k\rangle} \widehat{F}((\alpha,\beta),\gamma) &= \sum_\beta (-1)^{\langle \beta,k\rangle} \sum_{x,k'} (-1)^{\langle \alpha,x\rangle + \langle \beta,k'\rangle + \langle \gamma, F(x,k')\rangle} \\ &= \sum_{\beta,x,k'} (-1)^{\langle \alpha,x\rangle + \langle \beta,k+k'\rangle + \langle \gamma, E_{k'}(x)\rangle} \\ &= \sum_{x,k'} (-1)^{\langle \alpha,x\rangle + \langle \gamma, E_{k'}(x)\rangle} \sum_\beta (-1)^{\langle \beta,k+k'\rangle} \\ &= 2^\kappa \sum_x (-1)^{\langle \alpha,x\rangle + \langle \gamma, E_k(x)\rangle} \\ &= 2^\kappa \widehat{E_k}(\alpha,\gamma). \end{aligned}$$

For the key scheduled variant: $E_k^{\mathsf{KS}}(x) = F^{\mathsf{KS}}(x,k) = F(x,\mathsf{KS}(k))$. With the first part of Proposition 2.1 for the non-key-scheduled variant we have

$$2^\ell \widehat{E_k^{\mathsf{KS}}}(\alpha,\gamma) = \sum_\beta (-1)^{\langle \beta,k\rangle} \widehat{F^{\mathsf{KS}}}((\alpha,\beta),\gamma),$$

applying Lemma 2.1 results in

$$2^{\ell+\kappa} \widehat{E_k^{\mathsf{KS}}}(\alpha,\gamma) = \sum_{\beta,\beta'} (-1)^{\langle \beta,k\rangle} \widehat{\mathsf{KS}}(\beta,\beta') \widehat{F}\big((\alpha,\beta'),\gamma\big),$$

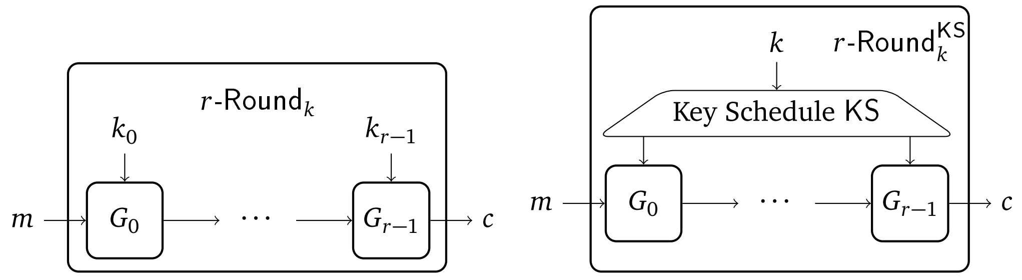
which concludes the proof. □

From Eq. (2.2) we can easily deduce the following equation by a simple application of the well-known fact [Car07, Corollary 2] that the Fourier transform is its own inverse, up to a constant factor.

$$\widehat{F}((\alpha,\beta),\gamma) = \sum_{k \in \mathbb{F}_2^\kappa} (-1)^{\langle \beta,k\rangle} \widehat{E_k}(\alpha,\gamma) \tag{2.3}$$

Eq. (2.3) might not seem helpful at first sight because it would mean a known-key attack. However, it turns out to be very meaningful in multiple ways. First of all, it will enable us to assert a clear meaning to the definition of a linear trail later in this section. Second, this is already the most basic form of the linear hull theorem for tweakable block ciphers which will be discussed in Section 2.5 extensively.

Next, we consider the already mentioned case of round-based block ciphers. The linear hull theorem can then be simplified such that the right-hand side of the equation only contains Fourier coefficients of the round functions. This specialization of the first proposition has applications for block ciphers that introduce the key material in other ways than simply XORing it onto the state.



(a) *Round-based key-dependent function* $r\text{-Round}_k$

(b) *and its key scheduled variant* $r\text{-Round}_k^{\mathsf{KS}}$

Figure 2.2: *Round-based functions.*

Proposition 2.2. *Let* $r\text{-Round}_k$ *and* $r\text{-Round}_k^{\mathsf{KS}}$ *be the functions (see Fig. 2.2a and Fig. 2.2b)*

$$
\begin{aligned}
r\text{-Round}_k &: \mathbb{F}_2^n \to \mathbb{F}_2^n & r\text{-Round}_k^{\mathsf{KS}} &: \mathbb{F}_2^n \to \mathbb{F}_2^n \\
r\text{-Round}_k(x) &:= G_{r-1}(\ldots(G_0(x,k_0),\ldots),k_{r-1}) & r\text{-Round}_k^{\mathsf{KS}}(x) &:= r\text{-Round}_{\mathsf{KS}(k)}(x)
\end{aligned}
$$

with $G_i : \mathbb{F}_2^n \times \mathbb{F}_2^\kappa \to \mathbb{F}_2^n$ *and key schedule* $\mathsf{KS} : \mathbb{F}_2^\ell \to \left(\mathbb{F}_2^\kappa\right)^r$. *Then*

$$
2^{r\kappa+(r-1)n}\,\widehat{r\text{-Round}_k}(\alpha,\gamma) = \sum_{\beta\in\left(\mathbb{F}_2^\kappa\right)^r} (-1)^{\langle\beta,k\rangle} \sum_{\substack{\theta\in\left(\mathbb{F}_2^n\right)^{r+1}\\ \theta_0=\alpha,\theta_r=\gamma}} \prod_{i=0}^{r-1} \widehat{G_i}((\theta_i,\beta_i),\theta_{i+1}),
$$

$$
2^{\ell+r\kappa+(r-1)n}\,\widehat{r\text{-Round}_k^{\mathsf{KS}}}(\alpha,\gamma) = \sum_{\substack{\beta\in\mathbb{F}_2^\ell\\ \beta'\in\left(\mathbb{F}_2^\kappa\right)^r}} (-1)^{\langle\beta,k\rangle}\,\widehat{\mathsf{KS}}(\beta,\beta') \sum_{\substack{\theta\in\left(\mathbb{F}_2^n\right)^{r+1}\\ \theta_0=\alpha,\theta_r=\gamma}} \prod_{i=0}^{r-1} \widehat{G_i}\left((\theta_i,\beta_i'),\theta_{i+1}\right).
$$

$\Diamond$

Proof.

Recall $r\text{-Round}_k(x) = G_{r-1}(\ldots(G_0(x,k_0),\ldots),k_{r-1})$. With the first part of Proposition 2.1 for the non-key-scheduled variant it holds

$$2^{r\kappa}\widehat{r\text{-Round}_k}(\alpha,\gamma) = \sum_{\beta}(-1)^{\langle\beta,k\rangle}\widehat{F}((\alpha,\beta),\gamma).$$

Applying Lemma 2.3 iteratively $r-1$ times we then get

$$2^{r\kappa+(r-1)n}\widehat{r\text{-Round}_k}(\alpha,\gamma) = \sum_{\beta}(-1)^{\langle\beta,k\rangle}\sum_{\substack{\theta\\ \theta_0=\alpha,\theta_r=\gamma}}\prod_{i=0}^{r-1}\widehat{G_i}((\theta_i,\beta_i),\theta_{i+1}).$$

The key scheduled variant follows from the second part of Proposition 2.1 and again applying Lemma 2.3 iteratively $r-1$ times. □

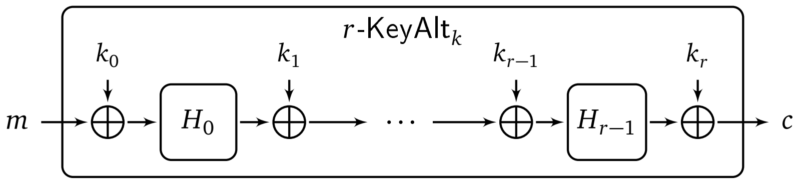
As this proposition looks a bit puzzling, let us elaborate a bit on it. We only need to pay attention to the rightmost part, the sum over θ and the product over the round functions' Fourier coefficients, as we know the other part already from Proposition 2.1. So basically $\widehat{r\text{-Round}_k}$ is the product of the round functions' G_i Fourier coefficients. But instead of having only one possible trail through all round functions, we can choose, after each round, which intermediate mask to use. Eventually, we end up with the sum over all possible θ, beginning with α and ending in γ, and thus having a linear hull over the round functions.

Finally, we focus on the case where E_k is round-based and the key-dependency is introduced by XORing a key onto the current state in each round, i. e. if E_k is a *key-alternating cipher* as depicted in Fig. 2.3a. This special case of the linear hull theorem is the most famous one. It is usually cited using the correlation of linear trails as defined in Eq. (2.1).

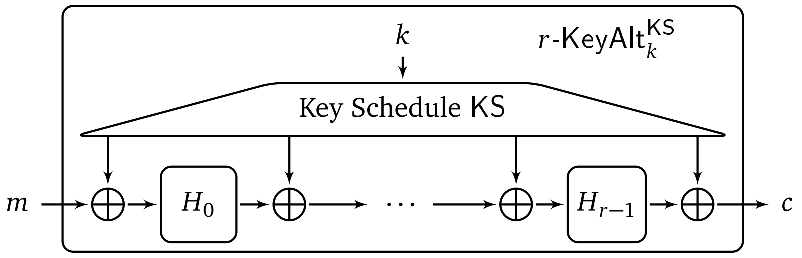
Another point that is nicely highlighted by this stepwise development via the round based function is the only small difference between Propositions 2.2 and 2.3. While we sum over both the key mask β and the round functions input mask θ in the former, the second sum collapses in the latter, as we will see in the next paragraph. This is due to the fact that we cannot say anything about the introduction of key material in a generic round function. But instead, if the key is simply XORed onto the input of the round function, this fixes the corresponding masks $\theta_i = \beta_i$, see [Bih95] or [BR14, Lemma 1].

Proposition 2.3. *Let $r\text{-KeyAlt}_k$ and $r\text{-KeyAlt}_k^{\mathsf{KS}}$ be the functions (see Fig. 2.3a and Fig. 2.3b)*

$$\begin{aligned} r\text{-KeyAlt}_k &: \mathbb{F}_2^n \to \mathbb{F}_2^n & r\text{-KeyAlt}_k^{\mathsf{KS}} &: \mathbb{F}_2^n \to \mathbb{F}_2^n \\ r\text{-KeyAlt}_k(x) &:= H_{r-1}(\ldots H_0(x+k_0)+\ldots)+k_r & r\text{-KeyAlt}_k^{\mathsf{KS}}(x) &:= r\text{-KeyAlt}_{\mathsf{KS}(k)} \end{aligned}$$



(a) *Key-Alternating function r-KeyAlt$_k$ over r rounds with $k = (k_0, \dots, k_r)$*



(b) *and its key scheduled variant r-KeyAlt$_k^{\mathsf{KS}}$*

Figure 2.3: *Key-Alternating functions.*

with $H_i : \mathbb{F}_2^n \to \mathbb{F}_2^n$ and key schedule $\mathsf{KS} : \mathbb{F}_2^\ell \to \left(\mathbb{F}_2^n\right)^{r+1}$. Then

$$\begin{aligned}
2^{(r-1)n}\widehat{r\text{-}\mathsf{KeyAlt}_k}(\alpha,\gamma) &= \sum_{\substack{\beta\in\left(\mathbb{F}_2^n\right)^{r+1}\\ \beta_0=\alpha,\beta_r=\gamma}} (-1)^{\langle\beta,k\rangle} \prod_{i=0}^{r-1} \widehat{H_i}(\beta_i,\beta_{i+1}) \\
&= 2^{rn} \sum_{\substack{\beta\\ \beta_0=\alpha,\beta_r=\gamma}} (-1)^{\langle\beta,k\rangle} C_\beta, \\
2^{\ell+(r-1)n}\widehat{r\text{-}\mathsf{KeyAlt}_k^{\mathsf{KS}}}(\alpha,\gamma) &= \sum_{\substack{\beta\in\mathbb{F}_2^\ell\\ \beta'\in\left(\mathbb{F}_2^n\right)^{r+1}\\ \beta'_0=\alpha,\beta'_r=\gamma}} (-1)^{\langle\beta,k\rangle} \widehat{\mathsf{KS}}(\beta,\beta') \prod_{i=0}^{r-1} \widehat{H_i}(\beta'_i,\beta'_{i+1}) \\
&= 2^{rn} \sum_{\substack{\beta,\beta'\\ \beta'_0=\alpha,\beta'_r=\gamma}} (-1)^{\langle\beta,k\rangle} \widehat{\mathsf{KS}}(\beta,\beta') C_{\beta'}.
\end{aligned}$$

$\Diamond$

Proof.

Using Proposition Proposition 2.1, Lemma Lemma 2.5, and Lemma 2.3 results in

$$2^{(2r-1)n}\widehat{r\text{-}\mathsf{KeyAlt}_k}(\alpha,\gamma) = \sum_{\substack{\beta\\ \beta_r=\gamma}} (-1)^{\langle\beta,k\rangle} \sum_{\substack{\theta\\ \theta_0=\alpha,\theta_r=\gamma}} \prod_{i=0}^{r-1} \widehat{G_i}((\theta_i,\beta_i),\theta_{i+1}),$$

and applying Lemma 2.4 for each round yields

$$2^{(r-1)n}\widehat{r\text{-}\mathsf{KeyAlt}_k}(\alpha,\gamma)=\sum_{\substack{\beta\\ \beta_0=\alpha,\beta_r=\gamma}}(-1)^{\langle\beta,k\rangle}\prod_{i=0}^{r-1}\widehat{H_i}(\beta_i,\beta_{i+1}).$$

The key scheduled variant follows analogously from the second part of Proposition 2.1. □

Furthermore, in the case of a key-alternating cipher with independent round keys, i. e. the case without a key schedule, the following lemma holds:

Lemma 2.6. *Let $E_k = r\text{-}\mathsf{KeyAlt}_k$ be a* key-alternating cipher *and F as defined in Proposition 2.1. Then*

$$2^{-(r+2)n}\widehat{F}((\alpha,\beta),\gamma)=\begin{cases}\displaystyle\prod_{i=0}^{r-1}c_{H_i}(\beta_i,\beta_{i+1})=C_\beta & \text{, if } (\alpha,\gamma)=(\beta_0,\beta_r)\\ 0 & \text{, else}\end{cases}.$$

◊

Proof. We first use Eq. (2.3) and then Proposition 2.3:

$$\begin{aligned}\widehat{F}((\alpha,\beta),\gamma)&=\sum_k(-1)^{\langle\beta,k\rangle}\widehat{E_k}(\alpha,\gamma)\\&=\sum_k(-1)^{\langle\beta,k\rangle}\left(2^n\sum_{\substack{\beta'\\ \beta'_0=\alpha,\beta'_r=\gamma}}(-1)^{\langle\beta',k\rangle}\prod_{i=0}^{r-1}c_{H_i}\big(\beta'_i,\beta'_{i+1}\big)\right)\\&=2^n\sum_{\beta',k}(-1)^{\langle\beta+\beta',k\rangle}\prod_{i=0}^{r-1}c_{H_i}\big(\beta'_i,\beta'_{i+1}\big)\\&=\begin{cases}2^{(r+2)n}\prod_{i=0}^{r-1}c_{H_i}\big(\beta'_i,\beta'_{i+1}\big) & \text{, for } (\alpha,\gamma)=(\beta_0,\beta_r)\\ 0 & \text{, else}\end{cases}.\end{aligned}$$

□

We like to highlight this fact as we feel it is not well-known, although it is of course implicitly contained in Nyberg's work, see e. g. [Nyb15, Theorem p. 12]: *The correlation of a linear trail is nothing but the Fourier coefficient of F* where $F:\mathbb{F}_2^n\times\left(\mathbb{F}_2^n\right)^{(r+1)}\to\mathbb{F}_2^n$ is the key-alternating cipher and the first and last key masks correspond to the message input and message output mask, respectively. Hence, alternatively to Eq. (2.1) we can write

$$C_\theta=2^{-(r+2)n}\widehat{F}((\theta_0,\theta),\theta_r).$$

This is important to keep in mind as it asserts a clear meaning to the correlation of a trail. And this is in contrast to many papers in the literature where either the trail is derived by the piling-up lemma or the correlation of a trail is given directly by using Eq. (2.1) as a definition, as done above to link the usual notation to what we feel is a cleaner way of presenting those connections. Given Lemma 2.6, one can also easily see the connection between Propositions 2.1 and 2.3. Here, all masks that do not start and end in α and γ vanish in the linear hull sum.

Distributions

When applying linear cryptanalysis in practice, we have to compute Fourier coefficients $\widehat{E_k}$ for some fixed key k. But as the Fourier coefficient exhibits a key-dependent behavior, see Eq. (2.2), we need to take into account how $\widehat{E_k}$ is distributed over the key space, i. e. what is the probability $\Pr_k\left[\widehat{E_k}(\alpha,\gamma) = X\right]$. In the case of key-alternating block ciphers *with independent round keys*, r-KeyAlt in our notation, the Fourier coefficient follows a normal distribution $\mathcal{N}$ and there are already results about the expected value and the expected squared value, e. g. see [DR02, pp. 103–108]. Namely,

$$\mathbb{E}\Big(\widehat{r\text{-KeyAlt}_k}(\alpha,\gamma)\Big) = \frac{1}{2^{(r+1)n}} \sum_{k\in\mathbb{F}_2^{(r+1)n}} \widehat{r\text{-KeyAlt}_k}(\alpha,\gamma) = 0,$$

and

$$\mathbb{E}\Big(\widehat{r\text{-KeyAlt}_k}(\alpha,\gamma)^2\Big) = \frac{1}{2^{(r+1)n}} \sum_{k\in\mathbb{F}_2^{(r+1)n}} \widehat{r\text{-KeyAlt}_k}(\alpha,\gamma)^2 = 2^{2n} \sum_{\substack{\beta\in\left(\mathbb{F}_2^n\right)^{r+1}\\ \beta_0=\alpha,\beta_r=\gamma}} C_\beta^2.$$

Thus, the mean μ is 0 and the variance $\sigma^2 = 2^{2n}\sum C_\beta^2$.

Daemen and Rijmen [DR07] did also extensively study the probability distributions for block ciphers with independent round keys in both, the setting for differential and linear cryptanalysis. However, they did not regard possible influences of the key schedule. But usually, real block ciphers have a (often linear) key schedule to generate round keys. In particular, we are interested in exactly this case, where the key schedule is linear. Such a key schedule can have unexpected influences on our standard assumptions for block cipher designs. In the following section, we investigate the special case of identical round keys.

2.3 Bizarre Examples

When we design a new cipher, we typically assume independent round keys and analyze the behavior of linear trails in the hope that the behavior when using an actual key schedule does not differ too much in practice. Note that, mainly due to Nyberg [Nyb95], the behavior of independent round keys is well-scrutinized. Here, one understands theoretically the basic parameters of the distribution of possible Fourier coefficients for varying keys. In particular, the average Fourier coefficients, that is the mean of the distribution, and the average squared Fourier coefficients can be formalized, as shown in Section 2.2. Moreover, we often expect the Fourier coefficients of linear trails to follow a normal distribution in the case of independent round keys.

Aside from this general result, only few research was conducted for round keys derived by a key schedule. One rather recent contribution by Abdelraheem *et al.* [Abd+12] exhibited Fourier

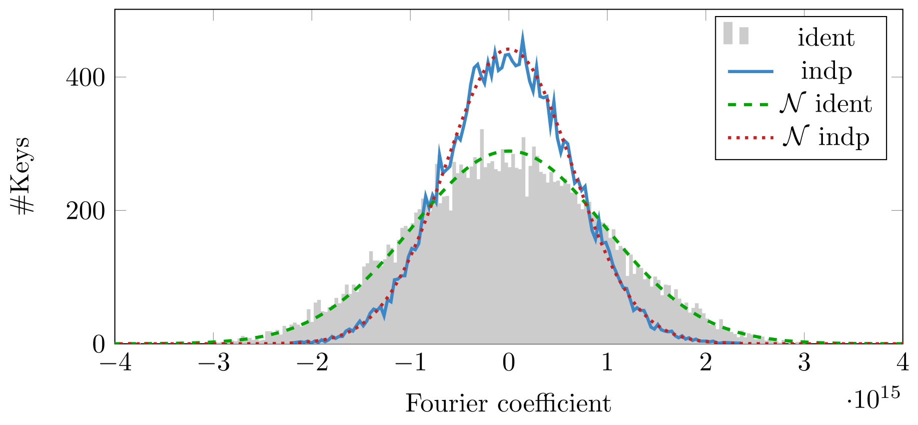


Figure 2.4: *Distribution of Fourier coefficients for standard* PRESENT *reduced to 10 rounds. Possible Fourier coefficients of the mask* (e_{21}, e_{21}) *are plotted on the abscissa, while the number of keys that lead to this Fourier coefficient is plotted on the ordinate.*

coefficient distributions as in Fig. 2.4. Here, the distribution for identical round keys has a significantly bigger variance than independent round keys, but still follows a normal distribution. Continuing this analysis, we conduct extensive experiments with PRESENT variants and report the observed distributions.

The main motivation behind those experiments was to explore if one can bound the fraction of weak keys, that is keys with a large absolute bias, tighter than by using the very general result by Tchebysheff's bound. In other words, we are interested in studying what can be said about the tails of the Fourier coefficient distribution over the keys.

Recall that, for any probability distribution *Tchebysheff's inequality* gives a result about deviations from the distribution's mean. Let D be a distribution with mean μ and variance σ^2. Then for any random variable $x \sim D$,

$$\Pr_x \left[|x - \mu| \geq k \cdot \sigma \right] \leq \frac{1}{k^2}.$$

While this is a general result for *any* probability distribution, we know much stronger results for some common distributions. In particular for the normal distribution that often seems a good approximation of the distribution of Fourier coefficients, much stronger bounds can be proven. More precisely, when considering a normal distribution, the cumulative distribution function (CDF) results in the well-known *68–95–99.7 rule* (or *three-sigma rule of thumb* [Gra06]) that says

- 68 % of the probability mass lies within one,
- 95 % lies within two, and
- 99.7 % lies within three standard deviations away from the mean.

The remainder of the section discusses our results for some selected S-boxes. Additionally, all results are given in Appendix A.

Experimental setting

PRESENT is a classical Substitution-Permutation-Network with a 64-bit block size and uses a substitution layer based on a 4-bit S-box with optimal properties regarding differential and linear cryptanalysis, together with a bit-permutation-based linear layer. In [LP07], the authors classified all 4-bit S-boxes and found 16 so-called *optimal* equivalence classes and 20 *Serpent-type* equivalence classes.[1] While optimal 4-bit S-boxes exhibit the best uniformity and linearity possible, the notion of Serpent-type S-boxes also include desired attributes to ensure a higher number of active S-boxes for differential cryptanalysis. The PRESENT S-box was chosen from one of these Serpent-type equivalence classes.

In order to better understand the behavior of identical round keys, we conducted extensive experiments with modified PRESENT versions. Our modifications are of the following form. We substituted the used S-box within the encryption with each of the optimal representatives O_0 to O_{15} and Serpent-type representatives R_0 to R_{19} given in [LP07]. Additionally, we reduced the encryption to 10 rounds. For each experimental distribution, we then computed the Fourier coefficients of 1-bit trails for 20 000 independent and 20 000 identical round keys.

Before discussing our results, let us recall some observations of PRESENT. In [Ohk09] Ohkuma has shown that 1-bit trails dominate the linear hull in the case of PRESENT, at least for a limited number of rounds. Later, Abdelraheem [Abd13] showed that with an increasing number of rounds, one has to take into account more trails in order to get good estimates of the total Fourier coefficient. A 1*-bit trail* $\theta = (\theta_0, \dots, \theta_r)$ is a trail, for which all intermediate masks θ_i have Hamming weight 1, i. e. $\mathrm{hw}(\theta_i) = 1$.

We build on these findings and run our experiments under the following assumption:

Assumption 2.1. 1*-bit trails dominate the linear hull of* PRESENT. ◊

We discuss the validity of this assumption in the next subsection, see Figs. 2.7 and 2.8.

As we consider a small number of rounds, we can thus approximate the Fourier coefficient of PRESENT by

$$\widehat{E_k}(\alpha, \gamma) = \sum_{\substack{\theta \in \left(\mathbb{F}_2^n\right)^{r+1} \\ \theta_0 = \alpha, \theta_r = \gamma}} (-1)^{\langle \theta, k \rangle} C_\theta \approx \sum_{\substack{\theta \\ \theta_0 = \alpha, \theta_r = \gamma \\ \mathrm{hw}(\theta_i) = 1}} (-1)^{\langle \theta, k \rangle} C_\theta .$$

We can exploit this observation in our experiments in two ways. First, as we have to consider only 1-bit trails, computing the Fourier coefficient becomes very efficient compared to computing Fourier coefficients of all trails. The reason for the reduced complexity is the following. Normally we utilize correlation matrices [DGV95] to compute the trail's Fourier coefficient. But as we restrict the trails to 1-bit masks only, we also greatly reduce the size of the corresponding correlation matrix.

Additionally, we can use the resulting matrix as an intuitive illustration of the Fourier coefficient-influencing parts of the cipher. For that purpose, we interpret the correlation matrix restricted to 1-bit trails as an adjacency matrix of a graph $\mathscr{G}$. We call $\mathscr{G}$ *the induced graph*. Standard PRESENT induces the graph depicted in Fig. 2.5. A vertex in $\mathscr{G}$ corresponds to a bit in

[1] Actually, a more general classification was already published in [Can07]

the cipher's state, an edge from α to γ to a trail over one round with non-zero Fourier coefficient. That is, α is connected to γ by an edge if

$$\widehat{H}(e_\alpha, e_\gamma) \neq 0,$$

where H denotes the PRESENT round function, and e_j the jth unit vector.

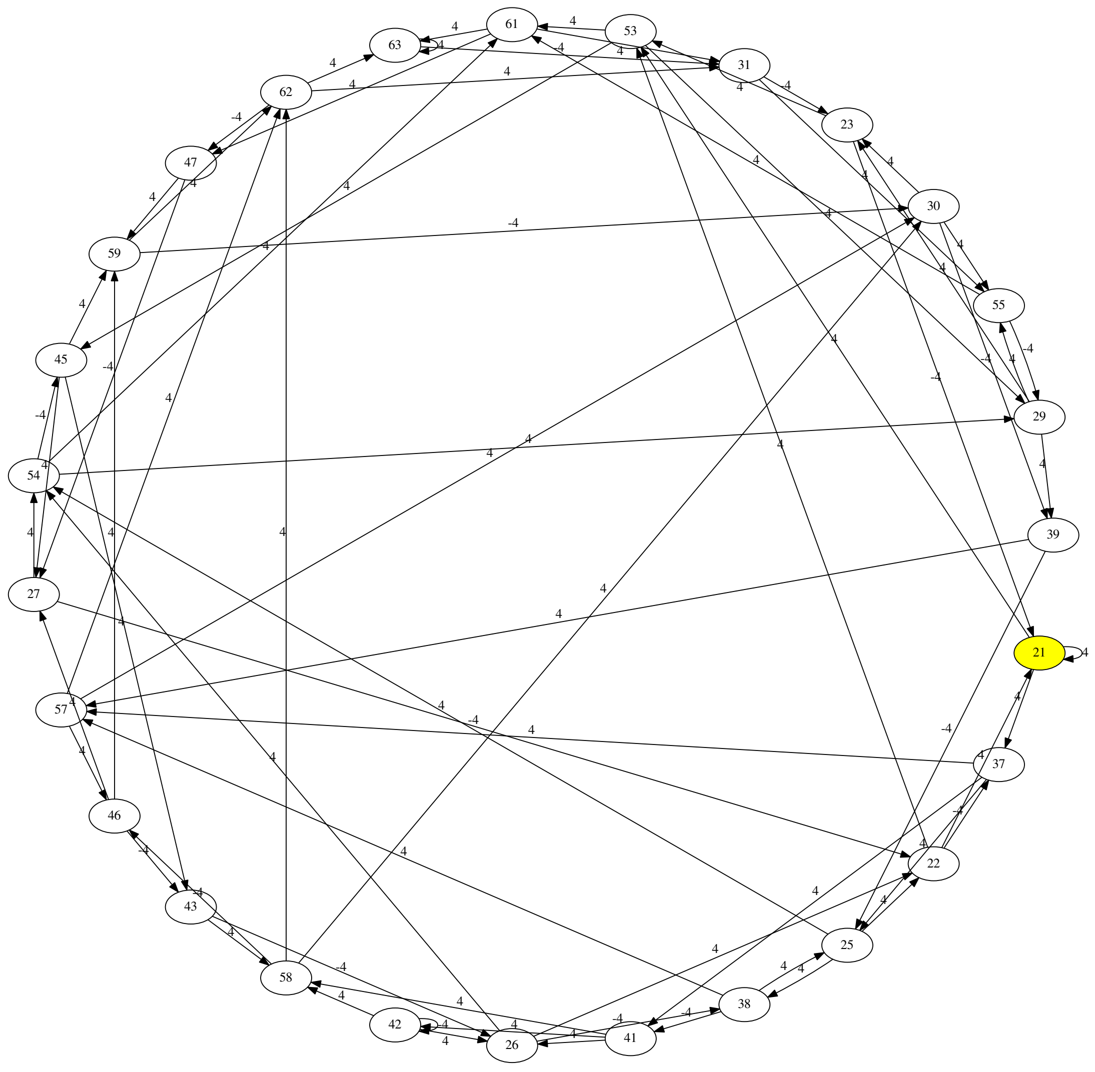


Figure 2.5: *Graph induced by* PRESENT. *Vertices α, γ correspond to possible* 1*-bit masks* (e_α, e_γ) *and are thus connected by an edge, if the Fourier coefficient at* (e_α, e_γ) *is non-zero. The highest number of trails is achieved by starting and ending in the marked vertex,* $e_\alpha = e_\gamma = e_{21}$.

Note that finding 1-bit trails over r rounds now reduces to finding paths in $\mathcal{G}$ of length r. $\mathcal{G}$ can be reduced in size, if we discard vertices not covered by paths of length r. Counting the

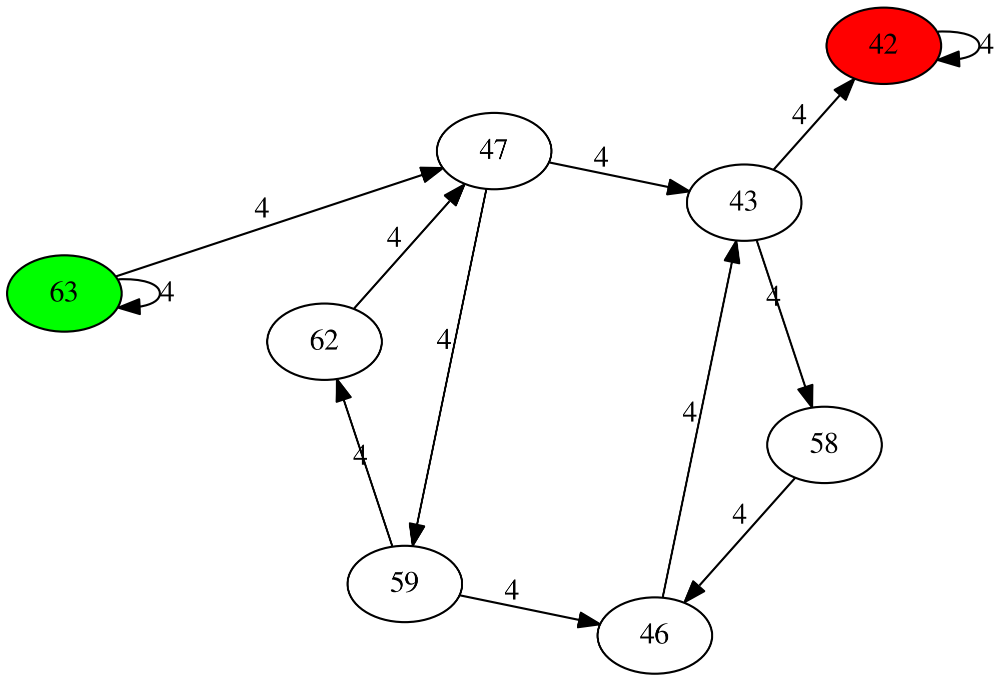


Figure 2.6: *Graph induced by* PRESENT *and* R_1. *Vertices* α, γ *correspond to possible* 1*-bit masks* (e_α, e_γ) *and are thus connected by an edge, if the Fourier coefficient at* $\left(e_\alpha, e_\gamma\right)$ *is non-zero. The highest number of trails is achieved for* $(e_\alpha = e_{63}, e_\gamma = e_{42})$.

number of 1-bit trails from α to γ over r rounds can now simply be done, by raising the adjacency matrix to the rth power. The resulting element at position (α, γ) is the number looked for.

Returning to Ohkuma's observations, the second advantage of this phenomenon is, it limits the number of keys that result in different behavior. Consider Eq. (2.2) and only 1-bit trails. The key-dependent sign of the Fourier coefficient now depends only on the few key bits masked by 1-bit trails. In the case of PRESENT, there are 27 out of the possible 64 bits of each round key.

Thus, significantly fewer key bits influence the Fourier coefficient, and further, all keys which are equal in these masked bits behave identically. For some S-boxes, we can then compute the distribution of Fourier coefficients of 1-bit trails over all keys.

The induced graph can differ significantly in size for different S-boxes. Fig. 2.6 shows the graph induced by PRESENT and R_1. Compared to standard PRESENT, only 8 of the originally 27 key bits influence the Fourier coefficient.

Table 2.1: S-box representative for the equivalence class R_1 that is used in our experiments.

x	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$R_1(x)$	0	3	5	8	6	9	10	7	11	12	14	2	1	15	13	4

Resulting distributions and behavior over several rounds

In our experiments, various distributions occur. For some S-boxes, we observe the same behavior as for standard PRESENT. Several other S-boxes exhibit unexpected distributions. We do not want to cover every individual distribution in detail here, but plots for each can be found in the appendix. Instead, we concentrate on R_1 (see Table 2.1), which is the most interesting example with respect to our initial question, i. e. to study the tails of the distribution. Fig. 2.7 shows the resulting distribution of 1-bit Fourier coefficients for R_1, see the bar plot. In Fig. 2.8 we plot the CDF, which has the advantage that the scaling issue of Fig. 2.7 vanishes.

Clearly, the resulting distribution does not follow a normal distribution.

When observing such a different distribution to the expected normal distribution, the question arises if Ohkuma's initial observation on PRESENT's behavior still is correct. That is, do the 1-bit trails still dominate the distribution of the Fourier coefficient?

In order to investigate this, we computed the distribution for all 2-bit trails on top of the 1-bit trails. As can be seen in Fig. 2.7 the 1-bit trails still dominate the general shape of the distribution. The 2-bit trails alone roughly follow a normal distribution with a relatively small variance. In total, this has the effect that the 2-bit trails together with the 1-bit trails differ from the 1-bit trails by changing the isolated discrete distribution into roughly bell-shaped parts. Thus, we can still see a clear dominance of the 1-bit trails in the 2-bit trail distribution, which supports the underlying assumption. In particular, the tail of the distribution is still far from following the normal distribution.

Most importantly in our context of studying the tails of the distributions, Fig. 2.7 exhibits two deviates "quite far" from the distribution's mean. Indeed, those outliers are more than three standard deviations away from the mean and have a joint probability of roughly 3 %. For the standardly assumed normal distribution the corresponding probability to lie outside of $3 \cdot \sigma$ is roughly a factor of 10 smaller, i. e. approximately 0.3 %, see the 68–95–99.7 rule. Moreover, when assuming independent round keys (which implies a significantly smaller variance) and a normal distribution, the fraction of keys with an absolute bias larger than $3 \cdot \sigma$ would be roughly 2^{-25}, which is an underestimation by a factor of roughly 2^{20}.

Table 2.2 summarizes the probabilities of these outliers for ten and twelve rounds.

When increasing the number of rounds further, it can be expected that at some point the dominance of the 1-bit trails vanishes, especially when the correlation of the 1-bit trails drops below $2^{-n/2}$. However, for an increasing number of rounds, the 1-bit trails show a fascinating behavior that we like to shortly elaborate below.

We normalize the Fourier coefficient by the distribution's standard deviation. For an increasing number of rounds, the above-mentioned outliers then converge to four standard deviations. Recall that for R_1, only $2^8 = 256$ keys exhibit distinct Fourier coefficients, due to the fact that we only

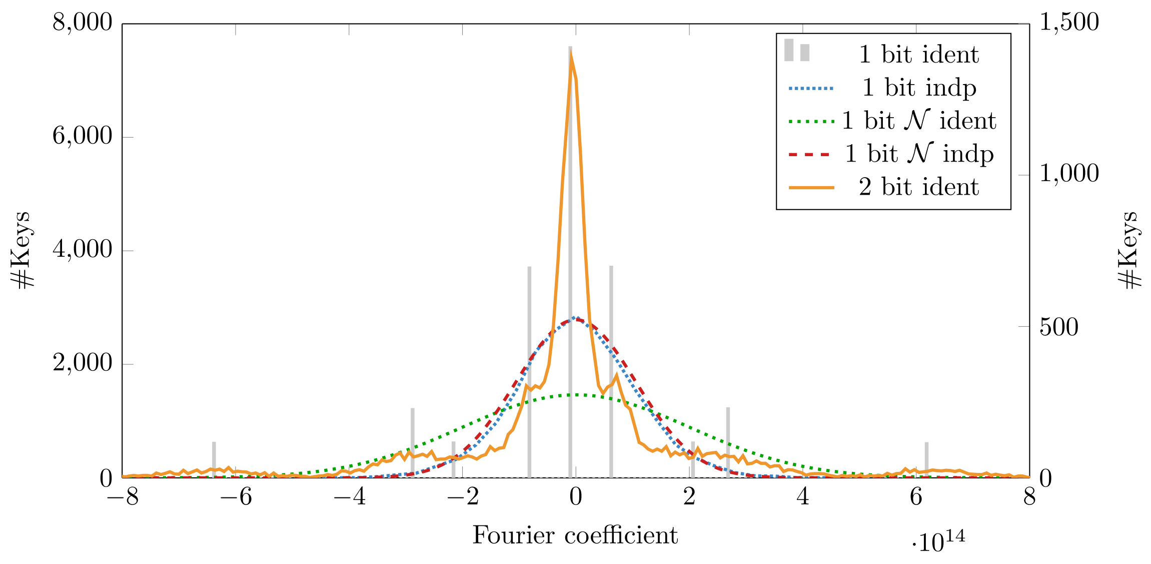


Figure 2.7: *Distribution of Fourier coefficients for* PRESENT *with R_1 S-box, reduced to 10 rounds. Again, Fourier coefficients for the mask (e_{63}, e_{42}) are plotted on the x-axis, the corresponding number of keys on the y-axis. Note that the plot for 2-bit trails is plotted against the right y axis.*

Table 2.2: Probability of outliers deviating more than $3 \cdot \sigma$, or $\Pr[|X| > 3 \cdot \sigma]$, for 1-bit and 2-bit distributions. For $X \sim \mathcal{N}(0, \sigma)$, $\Pr[|X| > 3 \cdot \sigma] = 0.0027$.

Rounds	$\log_2(\sigma)$	$\log_2(\sigma_{\mathcal{N}})$	$\Pr_{1\text{bit}}$	$\Pr_{2\text{bit}}$	$\log_2(\Pr_{\mathcal{N}})$
10	−16.50	−17.41	0.03130	0.0343	−25.59
12	−19.76	−21.01	0.03205	0.0342	−40.14

consider 1-bit trails. The outliers cover 16 out of the 256 possible keys, converging to the following distribution D_{lim}, see Fig. 2.9:

$$\widehat{E_k}(\alpha,\gamma) \sim D_{\text{lim}} \begin{cases} -4\sigma & \text{with probability } \frac{1}{32} \\ 0 & \text{with probability } \frac{15}{16} \\ 4\sigma & \text{with probability } \frac{1}{32} \end{cases} .$$

Thus, this distribution fulfills Tchebysheff's bound with equality:

$$256 \cdot \Pr\left[\left|\widehat{E_k}(\alpha,\gamma)\right| \geq 4 \cdot \sigma\right] = 256 \cdot \left(\frac{1}{32} + \frac{1}{32}\right) = 256 \cdot \frac{1}{4^2} = 16.$$

From our perspective of cipher designers, this is a worst-case behavior, as such a distribution not only exhibits a wider variance, but also shows a maximal fraction of weak keys possible for a given variance.

Although this resulting distribution is quite contrary to what we typically expect, we have to keep in mind that identical round keys can per se be insecure due to slide [BW99], invariant

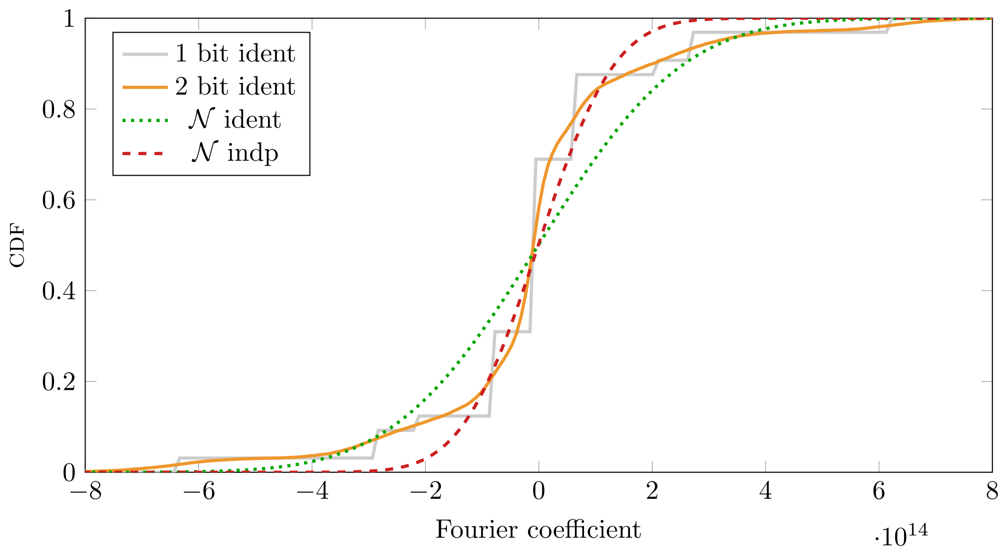


Figure 2.8: *Distribution's CDF of* 1*-bit and* 2*-bit Fourier coefficients, and the corresponding normal distribution for identical and independent round keys.*

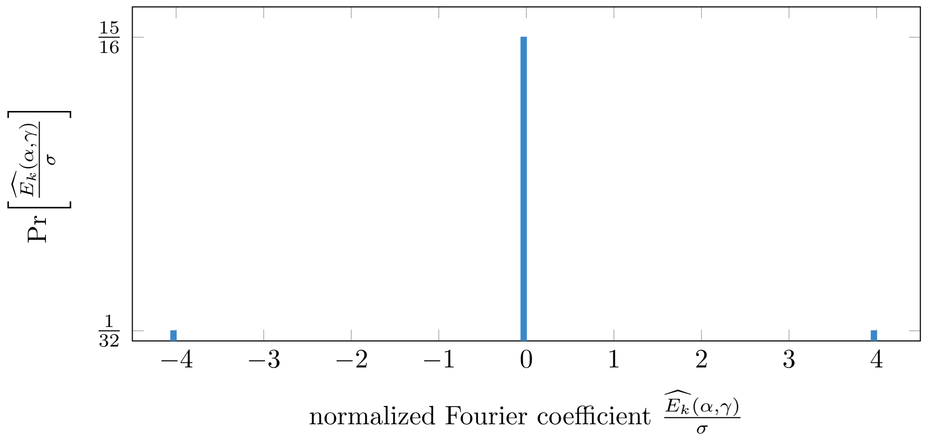


Figure 2.9: *Convergence distribution for* PRESENT *with* R_1 *S-box and many rounds. Here, the Fourier coefficient of* (e_{63}, e_{42}) *is normalized by the standard deviation* σ *(x-axis), while the corresponding probability to obtain such a Fourier coefficient is denoted on the ordinate.*

subspace [Lea+11], or non-linear invariant attacks [TLS16]. Therefore designs usually involve round constants. The next section takes their influence into account.

2.4 Linear Key Schedules

As mentioned above, in cipher design one typically makes use of the *hypothesis of independent round keys*, which states that the analyzed cipher shows a similar behavior when instantiated with the key schedule or with independent round keys. However, as discussed in the previous section, this assumption might actually be wrong.

Here we show that for any linear key schedule together with randomly chosen round constants, those distributions where the variance is significantly larger than for independent round keys are rare exceptions. That is, we theoretically back-up the use of linear key schedules as a sound design approach with respect to linear cryptanalysis. Interestingly, from a technical point of view, this observation is almost trivial.

We consider a key-alternating cipher and analyze the effect of a key schedule that consists of a linear function followed by the addition of a constant. Thus, the key schedule $\mathsf{KS} : \mathbb{F}_2^\ell \times \left(\mathbb{F}_2^n\right)^{r+1} \to \left(\mathbb{F}_2^n\right)^{r+1}$ is given as

$$\mathsf{KS}(k,c) = \mathsf{KS}_c(k) = L(k) + c,$$

where $L : \mathbb{F}_2^\ell \to \left(\mathbb{F}_2^n\right)^{r+1}$ is a linear function, and $c \in \left(\mathbb{F}_2^n\right)^{(r+1)}$. The constant has the form $c = (c_0, \ldots, c_r)$, where the $c_i \in \mathbb{F}_2^n$ are called the round constants.

Let us look at the key-alternating cipher r-KeyAlt using the key schedule KS_c, that is r-$\mathsf{KeyAlt}^{\mathsf{KS}_c}$. First, we note that all constants from the same coset of the linear subspace $U = L(\mathbb{F}_2^\ell)$ result in the same key schedule up to a permutation. Namely, given two constants $c_1 = L(k_1) + d$ and $c_2 = L(k_2) + d$, it holds that $\mathsf{KS}_{c_1}(k) = \mathsf{KS}_{c_2}(k + k_1 + k_2)$. Accordingly, when analyzing the squared Fourier coefficient over the keys, the choice of the constant c can be reduced to the choice of a coset $U + d$.

Applying the linear hull theorem (see Proposition 2.3), we can compute the average squared Fourier coefficient over the keys, that is the variance of the distribution for fixed input and output masks (α, γ) as

$$\begin{aligned}
\mathrm{Var}(c) &:= 2^{-\ell} \sum_{k \in \mathbb{F}_2^\ell} \widehat{r\text{-}\mathsf{KeyAlt}_k^{\mathsf{KS}_c}}(\alpha,\gamma)^2 \\
&= 2^{2n-\ell} \sum_{\substack{\theta,\theta' \in \left(\mathbb{F}_2^n\right)^{r+1} \\ \theta_0=\theta'_0=\alpha \\ \theta_r=\theta'_r=\gamma}} (-1)^{\langle \theta+\theta', c\rangle} C_\theta C_{\theta'} \sum_k (-1)^{\langle \theta, L(k)\rangle + \langle \theta', L(k)\rangle} \\
&= 2^{2n-\ell} \sum_{\substack{\theta,\theta' \in \left(\mathbb{F}_2^n\right)^{r+1} \\ \theta_0=\theta'_0=\alpha \\ \theta_r=\theta'_r=\gamma}} (-1)^{\langle \theta+\theta', c\rangle} C_\theta C_{\theta'} \sum_k (-1)^{\langle k, L^T(\theta) + L^T(\theta')\rangle} \\
&= 2^{2n} \sum_{\substack{\theta,\theta' \\ \theta_0=\theta'_0=\alpha \\ \theta_r=\theta'_r=\gamma \\ L^T(\theta)=L^T(\theta')}} (-1)^{\langle \theta+\theta', c\rangle} C_\theta C_{\theta'}.
\end{aligned}$$

Next, we look at the average variance over all possible constants c. As discussed above, except for a factor, this is actually the same as summing over one representative for each coset. We have

$$\begin{aligned}
\mathbb{E}_c\left(\mathrm{Var}(c)\right) &= 2^{-(r+1)n} \sum_{c \in \left(\mathbb{F}_2^n\right)^{r+1}} \mathrm{Var}(c) \\
&= 2^{2n-(r+1)n} \sum_{c} \sum_{\substack{\theta,\theta' \\ \theta_0=\theta'_0=\alpha \\ \theta_r=\theta'_r=\gamma \\ L^T(\theta)=L^T(\theta')}} (-1)^{\langle \theta+\theta', c\rangle} C_\theta C_{\theta'} \\
&= 2^{2n-(r+1)n} \sum_{\substack{\theta,\theta' \\ \theta_0=\theta'_0=\alpha \\ \theta_r=\theta'_r=\gamma \\ L^T(\theta)=L^T(\theta')}} C_\theta C_{\theta'} \sum_{c} (-1)^{\langle \theta+\theta', c\rangle} \\
&= 2^{2n} \sum_{\substack{\theta \\ \theta_0=\alpha \\ \theta_r=\gamma}} C_\theta^2.
\end{aligned}$$

Thus, *the average variance over all constants is the same as for independent round keys.*

While this is actually quite clear, as in both cases we eventually sum over all possible $2^{(r+1)n}$-bit round keys, this observation has an important implication for cipher design.

Having a key-alternating cipher, *any* linear key scheduling can be turned into a key schedule which is on average as good as having independent round keys (in terms of the variance of the distribution, and thus in terms of the fraction of weak keys): Simply choose random round constants.

Known ciphers that actually deploy this approach (for different reasons) include the low-latency cipher PRINCE [Bor+12] and the cipher LOWMC [Alb+15].

We conducted experiments on how the distributions vary for different choices of random round constants. As we will see in the following, in this case not only the variance behaves as in the independent round key setup, but the whole distribution does.

Experiments

We experimentally verified our results in the same setting as discussed in Section 2.3. Fig. 2.10 plots the resulting Fourier coefficient distribution.

The gray histogram in the background represents the distribution for independent random round keys. It smoothly follows a normal distribution as expected. The dashed line in the foreground depicts the distribution for identical round keys with an all-zero round constant. This distribution is similar to the independent round key case, but exhibits a wider variance, as already observed in [Abd+12] and discussed in Section 2.3. According to our results from above, this behavior must be a clear outlier.

Indeed, all other lines correspond to identical round keys with a random round constant, and all exhibit the same behavior following a normal distribution. While the plot only shows 256 different round constants, we conducted the same experiment for several thousand random round constants, each resulting in the same behavior.

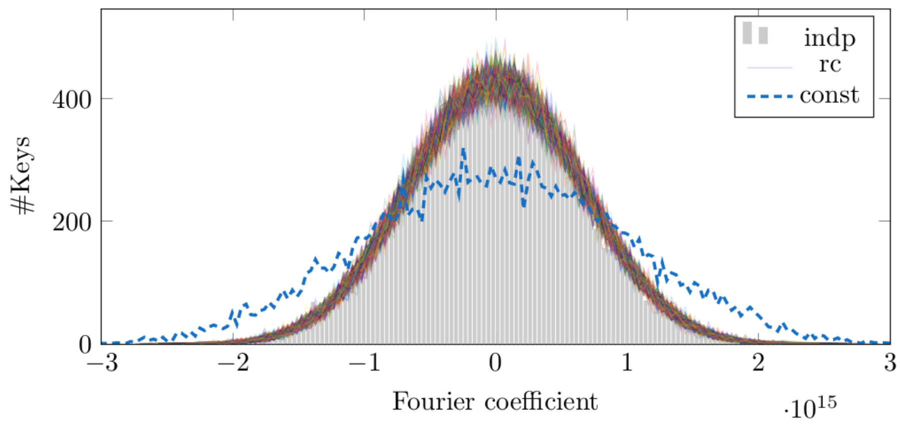


Figure 2.10: *Experimental distributions for* PRESENT *with different key schedules. The gray histogram is for independent random round keys. The dashed line is for identical round keys and an all-zero round constant. All other lines are for identical round keys and independent random round constants.*

2.5 Linear Approximations of Tweakable Block Ciphers

Tweakable block ciphers, introduced by Liskov et al. [LRW02], are an important cryptographic primitive. Recall to the mind from Section 1.2.3 that the basic idea of a tweakable block cipher is to use a public tweak as an additional input such that each tweak selects a different block cipher. The adversary is allowed to query the tweakable block cipher under a message and tweak of his choice. As motivated in Section 1.2.3, tweakable block ciphers have many important applications.

For a tweakable block cipher, the attacker is no longer restricted to linear approximations from the plaintext to the ciphertext, but can also make use of the tweak. In this section, we develop a formula for the linear hull of a tweakable block cipher and discuss its implications. We again develop our formulas in a top-down manner starting with a generic tweakable block cipher and then looking at the more special cases step by step.

A tweakable block cipher takes as input a key k, a tweak t, and a message x and computes the ciphertext c. It can then be written as a function

$$F : \mathbb{F}_2^n \times \mathbb{F}_2^\tau \to \mathbb{F}_2^n,$$

where m denotes the tweak size and, for simplicity, we hide the key-dependency in the function F itself. This means that we do not explicitly mention the key-dependency of F in our notation.

As in the case of keys, real block ciphers usually do not have independent tweaks, but rather have a tweak schedule generating the round tweaks. For the tweak schedule

$$\mathsf{TS} : \mathbb{F}_2^\ell \to \mathbb{F}_2^\tau$$

we define

$$F^{\mathsf{TS}} : \mathbb{F}_2^n \times \mathbb{F}_2^\ell \to \mathbb{F}_2^n$$

as

$$F^{\mathsf{TS}}(x,t) := F(x, \mathsf{TS}(t)).$$

Analogous to Section 2.2, we define $E_t^{\mathsf{TS}}(x) := F^{\mathsf{TS}}(x, t)$.

With the plaintext and the tweak, there are now two public inputs. Accordingly, an input mask for a linear approximation now consists of two parts, (α, β), the plaintext mask α and the tweak mask β. The main question is now how to express the Fourier coefficient of this linear approximation, that is, how to compute $\widehat{F^{\mathsf{TS}}}((\alpha, \beta), \gamma)$. While the most basic case of this linear hull for tweakable block ciphers was already discussed in Eq. (2.3), one can observe the following relation for a linear tweak schedule:

Proposition 2.4. *With the notation from above, for a linear tweak schedule L, it holds that*

$$\widehat{F^L}((\alpha, \beta), \gamma) = 2^{\ell-\tau} \sum_{\substack{\theta \in \mathbb{F}_2^\tau \\ L^T(\theta)=\beta}} \widehat{F}((\alpha, \theta), \gamma).$$

◊

Proof. As we have used the notation of a block cipher in two variables already intensively in Section 2.2, we can now reuse the results for tweakable ciphers. Accordingly, the basic ingredients for the proof are already known from that section. We first apply Eq. (2.3), then Eq. (2.2) and eventually use some basic summation techniques.

$$\begin{aligned}
\widehat{F^L}((\alpha, \beta), \gamma) &= \sum_{t \in \mathbb{F}_2^\ell} (-1)^{\langle \beta, t \rangle} \widehat{E_t^L}(\alpha, \gamma) \\
&= 2^{-\tau} \sum_{t \in \mathbb{F}_2^\ell} (-1)^{\langle \beta, t \rangle} \sum_{\theta \in \mathbb{F}_2^\tau} (-1)^{\langle \theta, L(t) \rangle} \widehat{F}((\alpha, \theta), \gamma) \\
&= 2^{-\tau} \sum_{\theta \in \mathbb{F}_2^\tau} \widehat{F}((\alpha, \theta), \gamma) \sum_{t \in \mathbb{F}_2^\ell} (-1)^{\langle \beta, t \rangle + \langle \theta, L(t) \rangle} \\
&= 2^{-\tau} \sum_{\theta \in \mathbb{F}_2^\tau} \widehat{F}((\alpha, \theta), \gamma) \sum_{t \in \mathbb{F}_2^\ell} (-1)^{\langle \beta + L^T(\theta), t \rangle} \\
&= 2^{\ell-\tau} \sum_{\substack{\theta \in \mathbb{F}_2^\tau \\ L^T(\theta)=\beta}} \widehat{F}((\alpha, \theta), \gamma)
\end{aligned}$$

□

In the following, we will consider what we call *tweak-alternating ciphers* analogous to key-alternating ciphers. Actually, all tweakable block ciphers we are aware of, including secondary constructions, are of this form. A tweak-alternating cipher is defined as

$$\begin{aligned}
r\text{-}\mathsf{TweakAlt}(x, t) &: \mathbb{F}_2^n \times \left(\mathbb{F}_2^n\right)^{r+1} \to \mathbb{F}_2^n \\
r\text{-}\mathsf{TweakAlt}(x, t) &:= H_{r-1}(\ldots H_0(x + t_0) + \ldots) + t_r
\end{aligned}$$

with $H_i : \mathbb{F}_2^n \to \mathbb{F}_2^n$. Analogous to above, the key-dependency is hidden in the round functions H_i.

Substituting this definition in Proposition 2.4 and using Lemma 2.6 directly gives the following corollary:

Corollary 2.1.

$$\widehat{r\text{-TweakAlt}^L}((\alpha,\beta),\gamma) = 2^{\ell-(r+1)n} \sum_{\substack{\theta\in\left(\mathbb{F}_2^n\right)^{r+1}\\ L^T(\theta)=\beta}} \widehat{r\text{-TweakAlt}}((\alpha,\theta),\gamma) = 2^{\ell+n} \sum_{\substack{\theta\in\left(\mathbb{F}_2^n\right)^{r+1}\\ L^T(\theta)=\beta\\ \theta_0=\alpha,\theta_r=\gamma}} \prod_{i=0}^{r-1} c_{H_i}(\theta_i,\theta_{i+1})$$

$\Diamond$

Note that we cannot yet write the last product as a trail correlation C_θ at this point because the influence of the key is still hidden in the round functions H_i. However, looking at a cipher that is not only tweak-alternating but also key-alternating, we can finally express the linear hull in terms of the trail correlations.

Corollary 2.2. *Let r-TweakAlt^L be a tweak-alternating cipher where the round keys $k = (k_0,\ldots,k_r)$ are added in a key-alternating way. It holds that*

$$\widehat{r\text{-TweakAlt}^L}((\alpha,\beta),\gamma) = 2^{\ell+n} \sum_{\substack{\theta\\ L^T(\theta)=\beta\\ \theta_0=\alpha,\theta_r=\gamma}} (-1)^{\langle\theta,k\rangle} C_\theta .$$

$\Diamond$

The crucial observation of Corollary 2.2 is that tweaking a block cipher with a linear tweak schedule does not introduce any new linear trails. In other words, the tweakable block cipher's linear hulls consist of linear trails that already existed in the linear hulls for the non-tweakable cipher. As explained in the introduction, this stands in contrast to differential trails, where it might well be that adding a difference in the tweak leads to new differential characteristics with a significantly higher probability than any differential characteristic for the non-tweaked version of the cipher.

In particular, from a designer's point of view, protecting a tweakable block cipher with a linear tweak schedule against linear cryptanalysis is not more difficult than for non-tweaked ciphers. In almost all settings, the best one can do as a designer, is to bound the correlation of single trails. As those trails are valid both for the tweaked as for the non-tweaked version, no special attention has to be paid to the additional freedom of the attacker. However, and this is important to note, in a tweakable block cipher, the attacker is potentially able to collect more data than in a traditional cipher, where the data complexity is clearly bounded by the block size. Thus, while the bounds are valid for both scenarios, a tweakable block cipher might require stronger bounds on the correlation of trails in order to argue its security. Again, the method of obtaining this bound stays the same when moving from a non-tweaked to a tweakable block cipher. It is an interesting question how the new degrees of freedom influence the linear hull in concrete examples.

2.6 Conclusion and Further Research

In this chapter, we introduced and systemized the basic concepts of linear cryptanalysis. We also provided new insights into the areas of key schedule design and tweakable block ciphers.

While section Section 2.3 points out interesting examples and strange behavior of the resulting distributions, we clearly lack insights on what causes those peculiarities exactly. We think that it is an interesting and challenging task for future research to theoretically explain our observations. In particular one might ask, why the S-box R_1 shows such a peculiar behavior and if there is a connection between the linear approximation table and the resulting distributions.

The theoretical and experimental results from Section 2.4 suggest that choosing random round constants in a linear key schedule is a good design approach. Shortly after our results were presented, Thomas Fuhr informed us that he found out that this observation actually holds for all key schedules, not only for linear ones. This significantly increases the impact of this result. For the proof, let the key schedule be $\mathsf{KS}_c(k) = f(k) + c$ for any function $f(k) : \mathbb{F}_2^\ell \to \left(\mathbb{F}_2^n\right)^{r+1}$. Then we get

$$\begin{aligned}
\mathbb{E}_c\left(\mathrm{Var}(c)\right) &= 2^{-(r+1)n-\ell} \sum_{c\in\left(\mathbb{F}_2^n\right)^{r+1}} \sum_{k\in\mathbb{F}_2^\ell} \widehat{r\text{-}\mathsf{KeyAlt}_k^{\mathsf{KS}_c}}(\alpha,\gamma)^2 \\
&= 2^{2n-(r+1)n-\ell} \sum_{c,k} \sum_{\substack{\theta,\theta'\in\left(\mathbb{F}_2^n\right)^{r+1} \\ \theta_0=\theta'_0=\alpha \\ \theta_r=\theta'_r=\gamma}} (-1)^{\langle\theta+\theta', f(k)+c\rangle} C_\theta C_{\theta'} \\
&= 2^{2n-(r+1)n-\ell} \sum_{k,\theta,\theta'} C_\theta C_{\theta'} \sum_{c'=c+f(k)} (-1)^{\langle\theta+\theta', c\rangle}.
\end{aligned}$$

When c takes all possible constant values, so does c'=c+f(k), therefore the last sum in the expression above equals 0 unless θ, θ' and we get:

$$\begin{aligned}
\mathbb{E}_c\left(\mathrm{Var}(c)\right) &= 2^{2n-(r+1)n-\ell} \sum_{k,\theta} C_\theta^2 \sum_{c'=c+f(k)} (-1)^0 \\
&= 2^{2n-\ell} \sum_{k,\theta} C_\theta^2 \\
&= 2^{2n} \sum_{\theta} C_\theta^2.
\end{aligned}$$

Also, the results from Section 2.5 are valuable for the practical design of cryptographic schemes. The fact that the tweak of a block cipher does not introduce new trails simplifies the security analysis when designing a tweakable block cipher. For example, our results have been directly applied in the design and analysis of the block ciphers SKINNY [Bei+16] and CRAFT [Bei+19]. Nevertheless, the linear hull is composed differently than before. It might therefore in some cases still enable the attacker to run a better linear attack than originally, although the underlying linear trails have not changed. To this end, future work could consist in experimentally analyzing and comparing the success of these attacks for concrete examples. Another obvious possibility of future work is the analysis of tweakable block ciphers with other than the basic linear and differential attacks. Here, very interesting results have been obtained by Ankele et al. [Ank+19]. They showed that our results for linear cryptanalysis do not hold for zero-correlation attacks [BR14] for block ciphers with a linear tweak schedule. That is, the introduction of a tweak often enabled them to attack additional rounds of the cipher.

Part II

Structural Cryptanalysis of ASASA

3 Decomposing the ASASA Block Cipher Construction

This chapter is based on the publication [Din+15], which is joint work with Itai Dinur, Orr Dunkelmann, and Gregor Leander. All authors equally contributed. The main contributions of the thesis author are in the integral attack (Section 3.3) and in the differential attack (Section 3.5).

3.1 Introduction

In this chapter we consider generic substitution permutation networks consisting of linear layers interleaved with S-box layers. The linear layers considered are arbitrary linear bijections on the full state, whereas the S-box layers partition the state into several fixed-size parts and apply a non-linear permutation to each part in parallel. Special cases of this general setup include a large variety of well-known block ciphers, most prominently the AES.

Such constructions are far from being new. The first construction, due to Patarin and Goubin, is the ASAS construction, i.e., two rounds of a linear layer (A) followed by an S-box layer (S) [PG97]. The scheme was later broken by Biham, exploiting the fact that the S-boxes were chosen to be non-bijective [Bih00] (which allowed for a simple differential attack).

Later, Biryukov and Shamir explored the general construction, denoted by SASAS (three S-box layers and two linear layers) [BS01; BS10]. In the context of this system, the cryptanalytic task is to recover the actual specification of the secret S-boxes and linear layers, given only black-box access to the SP-network and its inverse. The attack is a generalization of the integral attack [KW02] (a.k.a. SQUARE attack) which was already shortly introduced in Section 1.3.4 and originally proposed by Knudsen as an attack on the block cipher SQUARE [DKR97]. It is thus a structural attack in nature, exploiting the fact that the SASAS construction is composed of only three permutation layers over a smaller alphabet interleaved with linear layers, but ignoring the actual specification of these layers.

Besides the very general results of [BS01], several papers have focused on special cases of the above scenario where more information about the internal structure is known. Most prominently, a case that has received considerable attention is where only the S-boxes are unknown but the specification of the linear layers is completely revealed to the attacker (see e.g. [Bor+13; Tie+15]).

Here we focus not on a special case of the most general setting but rather on the natural complement of the SASAS structure, namely ASASA. That is, we consider an n-bit SP-network consisting of three linear (or affine) layers, interleaved with two S-box layers, where each S-box is a permutation over b bits. Our main motivation for analyzing this particular setup was given by a proposal from ASIACRYPT 2014 [BBK14] which constructed block ciphers with the additional property of having memory-hard white-box implementations (see [BBK14] for details). Such a block cipher is represented as a black-box (or a few black-boxes) describing how each input is mapped to an output. The goal of the adversary is to find a succinct representation of the given black-box mapping, which in our case implies recovering (or decomposing) the actual specifications of the affine and S-box layers.

Since the security of generic ASASA constructions was not previously analyzed, the authors of [BBK14] supported their designs by claiming that they resist standard attacks (such as differential and integral cryptanalysis, and boomerang attacks), essentially due to the external affine layers that hide the internal structure of the S-boxes. The best generic attack on the ASASA construction proposed in [BBK14] is a variant of the attack by Biryukov and Shamir on the SASAS structure, and is extremely inefficient, having a time complexity of $2^{m(n-m)}$. For example, [BBK14] claims that an ASASA scheme with a block size of $n = 16$ and an S-box size of $m = 8$ provides security of $2^{8(16-8)} = 2^{64}$.

It should be noted that [BBK14] also uses a special case of the ASASA structure with expanding S-boxes to construct strong white-box cryptography (again we refer to the paper for details) and that this part has been later broken [GPT15] by algebraic cryptanalysis. However, those attacks do not seem to carry over to the general ASASA case that we consider here. Furthermore, other attacks on the ASASA constructions have been found independently from our work by Minaud et al. [Min+15; Min+18]. This also includes an attack on the white-box scheme from [BBK14], that we are analyzing in this chapter.

Our Contribution

In this chapter, we describe several attacks that decompose the ASASA structure significantly faster than the $2^{m(n-m)}$ complexity of [BBK14]. Our most efficient attacks have time complexity of roughly $n \cdot 2^{3n/2}$, and when applied to the specific instances proposed in [BBK14], they have practical complexities. For example, the ASASA scheme with $n = 16$ and $m = 8$, which was claimed to provide security of 2^{64}, can in fact be broken in $16 \cdot 2^{(3 \cdot 16)/2} = 2^{28}$ time.

Table 3.1 shows the effectiveness of our attacks when applied to the instances given in [BBK14] in detail.

More precisely, we present and analyze three different attack strategies. The first attack follows the general idea of an integral attack and is thus a structural attack. The second and third

Table 3.1: Attack complexities for the ASASA instances suggested in [BBK14].

ASASA instance	Security claim [BBK14]	Our attack complexity (Section 3.3.3)
2×8-bit	64 bits	28 bits
2×10-bit	100 bits	35 bits
3×8-bit	128 bits	41 bits

attacks, on the other hand, are statistical attacks: the second attack is based on a boomerang attack, making use of boomerang distinguishers in the known-plaintext setting. The authors of [BBK14] were aware of the existence of such boomerang distinguishers, but left the possibility of exploiting them efficiently in an attack as an open problem, which we address. Finally, the third attack is a variant of a differential attack making use of non-random behavior of the differential distribution table. The running time of the integral attack is roughly $n \cdot 2^{3n/2}$, the boomerang has an attack complexity of roughly $2^{3n/2+3m/2}$ and the differential attack a complexity of 2^{2n}.

The integral attack is generally the most efficient attack and should be used whenever possible. However, this attack is applicable with the claimed complexity only to ASASA constructions in which $m > \sqrt{n}$. In cases where $m \leq \sqrt{n}$, the boomerang attack is generally the most efficient attack, as its time complexity is at most $2^{3n/2+3\sqrt{n}/2}$. However, the success and efficiency of the integral and boomerang attacks are based on some assumptions on ASASA instances in which the linear and S-box layers are chosen at random. These assumptions seem rather natural and were verified whenever possible on random instances of ASASA with small state sizes. Nevertheless, it is clear that our assumptions do not hold for all ASASA instances, and in such cases (which seem rather sparse), one can try to apply the differential attack which has a higher time complexity of 2^{2n}, but is based on different assumptions. Finally, we note that we focus on ASASA constructions in which all S-boxes have the same size, but our algorithms easily generalize to schemes built with different S-box sizes.

The chapter is organized as follows. After fixing the notation and some preliminary results outlining the general attack vector in Section 3.2, we present our integral attack in Section 3.3. The boomerang attack is given in Section 3.4. Finally, the differential attack is described in Section 3.5.

3.2 Preliminaries

Let
$$F : \mathbb{F}_2^n \to \mathbb{F}_2^n$$
be the ASASA construction with t S-boxes on m bits each ($n = tm$). More precisely, F is constructed as
$$F = L_2 \circ S_1 \circ L_1 \circ S_0 \circ L_0$$

where L_i are linear bijections on $\mathbb{F}_2^n$ and S_i consist of parallel applications of t m-bit S-boxes. That is,

$$S_i(x_1,\ldots,x_t) = \left(S_i^{(1)}(x_1),\ldots,S_i^{(t)}(x_t)\right)$$

with $x_i \in \mathbb{F}_2^m$

If we can recover the first linear layer, we are left with the SASA construction that can be decomposed very efficiently as shown in [BS10]. Thus, the aim of our attacks is to recover L_0 (or L_2 by considering the inverse of F). For this, first, note that it is impossible – and unnecessary – to recover L_0 exactly. This is due to the fact that a given ASASA instance is not uniquely determined by F. More precisely, replacing L_0 by

$$L_0' = \begin{pmatrix} T_1 \\ T_2 \\ \vdots \\ T_t \end{pmatrix} \circ L_0$$

and $S_0^{(i)}$ by

$$S'^{(i)}_0 = S_0^{(i)} \circ T_i^{-1},$$

where $T_i : \mathbb{F}_2^m \to \mathbb{F}_2^m$ are linear bijections, results in the same ASASA instance F. This observation motivates the following definition.

Definition 3.1. Two linear bijections L and L' on $\mathbb{F}_2^n$ are *ASASA-equivalent* if there exist linear bijections T_i on $\mathbb{F}_2^m$ such that

$$L' = \begin{pmatrix} T_1 \\ T_2 \\ \vdots \\ T_t \end{pmatrix} \circ L$$

◊

For recovering L_0 (up to equivalence) we are actually going to recover the subspaces

$$V_i = L_0^{-1}(U_i)$$

where

$$U_i = \{(x_1, x_2, \ldots, x_k) \mid x_j \in \mathbb{F}_2^m \text{ and } x_j = 0 \text{ if } j \neq i\}.$$

As formalized in the proposition below, given the set of V_i's basically determines L_0 up to (unavoidable) equivalences.

Proposition 3.1. *Given t subspaces V_i of dimension m it holds that any linear bijection $L : \mathbb{F}_2^n \to \mathbb{F}_2^n$ such that $L(V_i) = U_i$ with U_i as above is ASASA equivalent to L_0.* ◊

Proof. We consider $T = L \circ L_0^{-1}$ and observe that T maps inputs from U_i to U_i since

$$T(U_i) = L(L_0^{-1}(U_i)) = L(V_i) = U_i.$$

As a result, T represents t parallel linear bijections T_i over $\mathbb{F}_2^m$ and L is ASASA equivalent to L_0:

$$L \circ L_0^{-1} = T \Rightarrow L = T \circ L_0$$

□

Thus, after recovering the spaces V_i we can set up linear equations for L_0 and any bijective linear mapping fulfilling all those linear equations will be a valid solution for L_0.

More precisely, L_0 can be written as

$$L_0 = \begin{pmatrix} L_0^{(1,1)} & L_0^{(1,2)} & \dots & L_0^{(1,t)} \\ L_0^{(2,1)} & L_0^{(2,2)} & \dots & L_0^{(2,t)} \\ \vdots & \vdots & \ddots & \vdots \\ L_0^{(t,1)} & L_0^{(t,2)} & \dots & L_0^{(t,t)} \end{pmatrix}$$

where $L_0^{(i,j)}$ are $m \times m$ submatrices over $\mathbb{F}_2$. Then, for any element $(x_1, \dots, x_t) \in V_i \subseteq (\mathbb{F}_2^m)^t$ we obtain m equations

$$L_0^{(j,1)} x_1 + L_0^{(j,2)} x_2 + \ldots + L_0^{(j,t)} x_t = 0 \quad \forall j \in \{1, \dots, t\},\ j \neq i.$$

Choosing m linearly independent vectors from every V_i leads to a system of

$$t(t-1)m^2 = t^2m^2 - tm^2$$

linearly independent equations over $\mathbb{F}_2$ with t^2m^2 unknowns for L_0. This system of equations has 2^{tm^2} solutions and we know that the invertible solutions are exactly the linear bijections that are ASASA equivalent to L_0. The number of such linear bijections for a given L_0 is N_m^t where

$$N_m = \prod_{i=0}^{m-1}(2^m - 2^i) = 2^{m^2}\prod_{i=1}^{m}(1-2^{-i}) = 2^{m^2} p_m$$

is the number of invertible $m \times m$ matrices and p_m is the probability that a randomly chosen $m \times m$ matrix is invertible. Accordingly, the fraction of invertible solutions for the system of linear equations is

$$\frac{2^{tm^2} p_m^t}{2^{tm^2}} = p_m^t.$$

Thus, if we know the set of V_i's for an ASASA construction with t m-bit S-boxes, we try out p_m^t solutions on average. Since $p_m > 2^{-1.792}$, it holds that

$$p_m^t > 2^{-1.792t}$$

when considering constructions with up to tS-boxes. This step is completely separated from the attacks that find the set of V_i's. Accordingly, the complexities do *not* multiply and as we will see this step does not dominate the overall attack complexity.

Note that it is sufficient to know $V_{\pi(i)}$ for an unknown permutation π. The solution will then be a matrix which is ASASA equivalent to L_0 with its rows permuted according to π. This is also a valid solution as additionally permuting the S-boxes in S_1 and the columns in L_1 results in the same ASASA instance.

3.3 Integral Attack

In this section, we describe our integral attack on the ASASA construction. For integral attacks, also see Section 1.3.4. We start by describing a basic integral attack that recovers the subspaces V_i for $i \in \{1, \ldots, t\}$ in time complexity of about $n \cdot 2^{2n}$. Finally, we optimize the attack and reduce its time complexity to about $n \cdot 2^{3n/2}$.

The analysis of the attack assumes that the parameters of the ASASA construction satisfy $m > t$, or equivalently $m > \sqrt{n}$ (and $t < \sqrt{n}$). Moreover, it assumes that the algebraic degree of the S-box layers S_0 and S_1 and their inverses is the maximal possible value of $m - 1$.[1] We note that if the S-boxes are selected at random, then this will be the case with high probability. Additionally, the analysis is based on more subtle and heuristic (but natural) assumptions that we specify later in this section.

The starting point of the attack is the following integral property, which was used to devise a related attack in the original ASASA paper [BBK14]. We begin with a short definition, followed by the property and its proof.

Definition 3.2. Given a set R and $x \in \mathbb{F}_2^n$, define $R + x := \{x + y \mid y \in R\}$ ◊

Proposition 3.2. *Let $x \in \mathbb{F}_2^n$ and $i \in \{1, \ldots, t\}$, then* $\sum\limits_{y \in V_i} F(x + y) = 0.$ ◊

Similar properties have been used before in integral and related attacks, most notably in the cryptanalysis of the SASAS structure [BS10]. It is possible to prove this property combinatorially (as done in [BS10]), but here we give an algebraic proof, as it is more relevant to the rest of this section. The proof is based on the use of high-order derivatives (see [Knu95; Lai94] for details about high-order differential cryptanalysis).

Proof. Due to the linearity of L_0, we have $L_0(x + V_i) = L_0(x) + L_0(V_i) = L_0(x) + U_i$. Namely, S-box i is active in S_0 (its input attains all 2^m possible values), while all other S-boxes are inactive, as their inputs are fixed to the corresponding m bits of $L_0(x)$. Since $S_0^{(i)}$ is a permutation, then $S_0(L_0(x) + U_i) = S_0(L_0(x)) + U_i$, which is an affine subspace of dimension m. We are interested in computing the sum of outputs (over $\mathbb{F}_2$) of this subspace through the remaining layers $L_2 \circ S_1 \circ L_1$, which is equivalent to evaluating an m-order derivative of these layers on their n output bits. Therefore, in order to show that $\sum_{y \in V_i} F(x + y) = 0$, it is sufficient to show that the algebraic degree of every output bit of $L_2 \circ S_1 \circ L_1$ is less than m, which implies that all of the m-order derivatives are zero.

All the m-bit S-boxes in the scheme (and in S_1) are bijective, and therefore, their algebraic degree is at most $m - 1$, implying that the degree of S_1 is upper bounded by $m - 1$. Since L_1 and L_2 are affine, the degree of $L_2 \circ S_1 \circ L_1$ is less than m, proving the claim. □

In the attack, we sum over the encryption values of larger affine subspaces and use the following (more general) property.

Proposition 3.3. *Let $x \in \mathbb{F}_2^n$, $i \in \{1, \ldots, t\}$, and let R be a linear subspace such that $V_i \subseteq R$, then* $\sum\limits_{y \in R} F(x + y) = 0.$ ◊

[1]This condition can be somewhat relaxed, but we do not go into details for the sake of simplicity.

Proof. Write R as a direct sum of orthogonal subspaces $R = V_i \oplus R'$. Then

$$\sum_{y\in R} F(x \oplus y) = \sum_{y_1\in R', y_2\in V_i} F(x + y_1 + y_2) = \sum_{y_1\in R'} (\sum_{y_2\in V_i} F((x + y_1) + y_2)) = 0,$$

using the previous property. □

When we select at random a linear subspace R of a fixed dimension d such that $m \leq d < n$, Proposition 3.3 implies that we can potentially distinguish between the case $V_i \subseteq R$ (where $\sum_{y\in R} F(y) = 0$), and the case $V_i \nsubseteq R$ (assuming $\sum_{y\in R} F(y) \neq 0$). Distinguishing between these two cases is very useful, as it narrows down the search for V_i from the full n-dimensional space to the smaller space R of dimension $d < n$. Assuming that we find two subspaces $R_1 \neq R_2$ such that $V_i \subseteq R_1$ and $V_i \subseteq R_2$, then $V_i \subseteq R_1 \bigcap R_2$, which further narrows down the search for V_i. We then continue the search until we find sufficiently many subspaces such that their intersection gives the m-dimensional subspace V_i. This gives rise to the attack given in Algorithm 1.

The success and complexity of the attack are determined by the two quantities defined below.

Definition 3.3. Let R be a subspace chosen at random by Algorithm 1 and fix $i \in \{1, \ldots, t\}$. The covering probability is defined as

$$p_c := Pr[V_i \subseteq R].$$

◊

Definition 3.4. Let R be a subspace chosen at random by Algorithm 1. The false alarm probability is defined as

$$p_f := Pr[\sum_{y\in R} F(y) = 0 \mid V_i \nsubseteq R \text{ for each } i \in \{1, \ldots, t\}].$$

◊

Our analysis is based on the two assumptions below. The validity of these assumptions depends on the concrete ASASA scheme and we cannot prove them in general. The assumptions are discussed at the end of Section 3.3.2.

Assumption 3.1. *There are no false alarms in Line 17 of Algorithm 1. Namely, if $s' = 0$, then there exists V_i such that $V_i \subseteq R'$.* ◊

This assumption guarantees that if the algorithm halts, it returns the correct answer. This assumption can be somewhat relaxed, as noted at the end of Section 3.3.2.

Assumption 3.2. *For $d = n - t$, the false alarm probability in Line 7 of Algorithm 1 satisfies $p_f < 2^{-n/2}$. Namely,*

$$Pr[\sum_{y\in R} F(y) = 0 \mid V_i \nsubseteq R \textit{ for each } i \in \{1, \ldots, t\}] < 2^{-n/2}.$$

◊

Note that if $\sum_{y\in R} F(y)$ is uniformly distributed when $V_i \nsubseteq R$ for each $i \in \{1, \ldots, t\}$, then $p_f = 2^{-n}$. Assumption 3.2 is weaker and only requires $p_f < 2^{-n/2}$.

The analysis of the algorithm is rather involved, and we start with a high-level description.

Algorithm 1: Integral Attack

```
   Input  : An integer $d$
   Output : Subspaces $V_i$ for $i \in 1, \dots, t$
 1 begin
       // The set of candidate subspaces
 2     $Candidates \longleftarrow \emptyset$;
 3     $Found \longleftarrow 0$;
 4     while $Found < t$ do
 5         Pick a subspace $R$ of dimension $d$ at random;
 6         $s \longleftarrow \sum_{y \in R} F(y)$;
 7         if $s = 0$ then
               // Iterate over found subspaces
 8             for $j \leftarrow 1$ to $size(Candidates)$ do
 9                 $R' \longleftarrow R \bigcap Candidates[j]$;
10                 if $R' = Candidates[j]$ then
                       // $R$ contains a previously found subspace
11                     go to 4;
12                 end
13                 if $dim(R') < m$ then
                       // The dimension of $R'$ is too small
14                     go to 8;
15                 end
16                 $s' \longleftarrow \sum_{y \in R'} F(y)$;
17                 if $s' = 0$ then
                       // Narrow down subspace
18                     $Candidates[j] \leftarrow R'$;
19                     if $dim(R') = m$ then
                           // Found a subspace of dimension $m$
20                         $V_{Found} \leftarrow R'$;
21                         $Found \leftarrow Found + 1$;
22                     end
23                     go to 4;
24                 end
25             end
               // Store $R$, as it does not narrow down any previously
                  found subspace
26             $NewIndex \leftarrow size(Candidates) + 1$;
27             $Candidates[NewIndex] \leftarrow R$;
28         end
29     end
30 end
```

3.3.1 Analysis Overview

We estimate the time complexity of the algorithm based on several propositions which are stated below and proved in Section 3.3.2.

Proposition 3.4. *Let R be an affine subspace of dimension $d > m$ chosen at random by Algorithm 1, then*

$$p_c \geq 2^{m(d-n)} \cdot (1 - 2^{m-d-1}) \approx 2^{m(d-n)}.$$

◊

Proposition 3.5. *Let e be the number of subspaces R that Algorithm 1 considers before halting, then*

$$e \approx n \cdot p_c^{-1}.$$

◊

Proposition 3.6. *The expected total time complexity of Algorithm 1 is about*

$$(2^d \cdot e) \cdot (1 + e \cdot p_f^2).$$

◊

Plugging the value of p_c obtained from Proposition 3.4 into the value of e, obtained from Proposition 3.5, we get

$$e \approx n \cdot p_c^{-1} \approx n \cdot 2^{m(n-d)}.$$

Plugging this value of e into the total time complexity of the attack obtained from Proposition 3.6, we can conclude that it is preferable to select the largest dimension d which does not significantly affect p_f. However, according to the upper-bound of Proposition 3.7, we cannot take d too large, as this will result in $p_f = 1$, causing the algorithm to fail (or to be extremely inefficient). We note that unlike the previous propositions, the upper bound on d is not directly used in the complexity analysis of the attack, but rather restricts the possible choice of parameters.

Proposition 3.7. *Assume that the parameters of the ASASA construction satisfy $m > t$ and $d > n-t$, then $p_f = 1$.* ◊

Since we require $p_f < 1$, we select the maximal possible value for which this may occur, namely $d = n - t$. According to Proposition 3.4,

$$p_c \approx 2^{m(d-n)} = 2^{-mt} = 2^{-n}.$$

Plugging the value $p_c = 2^{-n}$ into the expression obtained in Proposition 3.5, we get

$$e \approx n \cdot p_c^{-1} \approx n \cdot 2^n.$$

The total time complexity of the attack is estimated in Proposition 3.6 as $(2^d \cdot e) \cdot (1 + e \cdot p_f^2)$, where $e \approx n \cdot 2^n$. According to Assumption 3.2, $p_f < 2^{-n/2}$, implying that $1 + e \cdot p_f^2 \approx 1$ and the time complexity of the algorithm is about

$$2^d \cdot e \approx n \cdot 2^{2n},$$

since $d = n - t \approx n$ when $t < \sqrt{n}$.

3.3.2 Detailed Analysis

In this section, we prove Propositions 3.4 to 3.7 and consider Assumptions 3.1 and 3.2.

Proofs of Propositions 3.4 to 3.7

Proof of Proposition 3.4. Let $(a_1, \ldots, a_m)$ be an arbitrary basis of V_i. We have $V_i \subseteq R$ if and only if $a_j \in R$ for $j \in \{1, \ldots, m\}$. Since R is a random d-dimensional subspace, then $Pr[a_1 \in R] = 2^{d-n}$. Next, since a_1 and a_2 are linearly independent $Pr[a_2 \in R \mid a_1 \in R] = 2^{d-n} - 2^{-n}$, and $Pr[\{a_1, a_2\} \subseteq R] = Pr[a_1 \in R] \cdot Pr[a_2 \in R \mid a_1 \in R] = 2^{d-n} \cdot (2^{d-n} - 2^{-n})$.

In general, for $j > 1$

$$Pr[\{a_1, \ldots, a_j\} \subseteq R \mid \{a_1, \ldots, a_{j-1}\} \subseteq R] = 2^{d-n} - 2^{j-2-n}$$

and therefore

$$\begin{gathered}
p_c = Pr[\{a_1, \ldots, a_m\} \subseteq R] = \\
Pr[\{a_1, \ldots, a_m\} \subseteq R \mid \{a_1, \ldots, a_{m-1}\} \subseteq R] \cdot Pr[\{a_1, \ldots, a_{m-1}\} \subseteq R] = \\
Pr[a_1 \in R] \cdot \prod_{j=2}^{m} Pr[\{a_1, \ldots, a_j\} \subseteq R \mid \{a_1, \ldots, a_{j-1}\} \subseteq R] > \\
\prod_{j=1}^{m} (2^{d-n} - 2^{j-2-n}) = \\
(2^{d-n})^m \cdot \prod_{j=1}^{m} (1 - 2^{j-2-d}) \geq \\
2^{m(d-n)} \cdot (1 - 2^{m-d-1}),
\end{gathered}$$

where the last inequality can be easily proved by induction. □

Proof of Proposition 3.5. We estimate e as follows. Fix $i \in \{1, \ldots, t\}$ and let $R_1, \ldots, R_\ell$ be ℓ random d-dimensional subspaces under the restriction that $V_i \subseteq R_j$ for each $j \in \{1, \ldots, \ell\}$. Since each subspace R_j is of dimension $n - t$, it is easy to see that for $\ell = n/t = m$, with good probability, $\bigcap_{j=1}^{\ell} R_j = V_i$ (as every subspace in the sequence is expected to reduce the dimension of the intersection by about t, until the intersection is equal to V_i). Therefore, after about $m \cdot p_c^{-1}$ choices of random subsets, we expect that V_i will be recovered in Line 20. Note that this assumes that there are no false alarms in Line 17, as such false alarms will disrupt the sequence $\bigcap_{j=1}^{\ell} R_j$, stored in memory.[2] In order to recover all V_i for $i \in \{1, \ldots, t\}$, we need to try about

$$e \approx \log(t) \cdot m \cdot p_c^{-1} < n \cdot p_c^{-1}$$

random subsets of dimension $d = n - t$. □

[2]We also assume that we did not select R such that $V_i \subseteq R$ and $V_j \subseteq R$ for $i \neq j$. This is event is very unlikely and can be ignored.

Proof of Proposition 3.6. The time complexity of summing over all the d-dimensional subspaces in Line 6 of the attack is $e \cdot 2^d$. Moreover, there are additional operations on the set of candidates that we need to take into account. At the end of the attack, the size of the stored subspaces in $Candidates$ depends on p_f and is about $e \cdot p_f$, in addition to the t subspaces which are not false alarms. Therefore, at the end of the attack $size(Candidates) \approx t + e \cdot p_f \approx e \cdot p_f$. Every candidate subspace R is intersected with all previous subspaces in $Candidates$, and the sum over the intersection R' is computed. In total, the amount of additional work for the candidates is dominated by Line 16 and is about $(e \cdot p_f)^2 \cdot 2^d$. The total time complexity of the attack is about

$$e \cdot 2^d + (e \cdot p_f)^2 \cdot 2^d = (2^d \cdot e) \cdot (1 + e \cdot p_f^2).$$

□

Proof of Proposition 3.7. Assume that $d > n - t$, and we want to show that $p_f = 1$. Since R is a linear subspace, the expression $\sum_{y \in R} F(y)$ evaluates a derivative of F of order higher than $n-t$ for each of the n output bits. Therefore, it is sufficient to show that $deg(F) < n-t+1$, which implies that the outcome of the derivation is zero regardless of R. In other words, $Pr[\sum_{y \in R} F(y) = 0] = 1$, and this holds in particular if $V_i \nsubseteq R$ for each $i \in \{1, \ldots, t\}$, implying $p_f = 1$.

The fact that $deg(F) < n - t + 1$ is derived from a theorem due to Boura and Canteaut in [BC13] (stated below in a slightly modified form).

Theorem 3.1 ([BC13]). *Let H be a permutation of $\mathbb{F}_2^n$ and let G be a function from $\mathbb{F}_2^n$ to $\mathbb{F}_2^n$. Then we have*

$$deg(G \circ H) < n - \lfloor \frac{n - 1 - deg(G)}{deg(H^{-1})} \rfloor.$$

◊

In our case, let $H = L_1 \circ S_0 \circ L_0$ and $G = L_2 \circ S_1$, then $deg(G) = deg(H^{-1}) = m - 1$ (as assumed at the beginning of the section). Therefore,

$$\begin{aligned} deg(F) < n - \lfloor \frac{n-1-m-1}{m-1} \rfloor = n - \lfloor \frac{mt-m}{m-1} \rfloor = \\ n - \lfloor \frac{(m-1)(t-1)+(t-1)}{m-1} \rfloor = \\ n-(t-1) + \lfloor \frac{t-1}{m-1} \rfloor = n - t + 1, \end{aligned}$$

as $t < m$. □

Assumptions on False Alarms

Assumption 3.1 states that false alarms do not occur in Line 17 of Algorithm 1. A false alarm in Line 17 occurs in case there are false alarms for both R and the smaller subspace $R' \subseteq R$. This event is significantly less likely than p_f, but may occur nevertheless. Therefore, we relax the assumption and slightly modify the algorithm to deal with such false alarms: in case $s' = 0$ in Line 17, before updating the candidate list we add an additional filtering to test the condition $V_i \subseteq R'$. This is done by selecting at random a vector x which is not in R', and testing whether

$\sum_{y \in span(\{x\} \bigcup R')} F(y) = 0$. The additional filtering will not result in deterioration in performance (again, assuming false alarms in Line 17 do not occur often).

Recall that in the attack we evaluate arbitrary d-order derivatives of F, and therefore the value of p_f in Assumption 3.2 depends on the density of monomials of degree (at least) d in its algebraic normal form. We select the maximal possible value $d = n - t$ for which it is theoretically possible that $deg(F) \leq d$, or $p_f < 1$ (and in particular, we assume that $p_f < 2^{-n/2}$). Indeed, as shown above, the bound due to [BC13] implies that $deg(F) < n - t + 1$, but does not rule out the possibility that $deg(F) = n - t$. Moreover, the trivial bound on the algebraic degree of F gives $\deg(S_0) \cdot deg(S_1) = (m-1) \cdot (m-1) \geq (\sqrt{n})^2 = n > n - t$, and does not contradict the possibility that $deg(F) = n - t$.

Our experiments of toy ASASA variants confirm our assumption about p_f. In fact, for ASASA schemes with 2 or 3 S-boxes, $\sum_{y \in R} F(y)$ was almost uniformly distributed when $V_i \nsubseteq R$ for each $i \in \{1, \ldots, t\}$, namely $p_f \approx 2^{-n}$.

3.3.3 Optimized Integral Attack

The basic integral attack sums the outputs of $e \approx n \cdot 2^n$ subspaces, each containing $2^d = 2^{n-t} \approx 2^n$ elements. Therefore, its total time complexity is about $n \cdot 2^{2n}$. The complexity of summing over the subspaces can be significantly reduced if we choose correlated subspaces instead of picking them at random. More specifically, as we show next, it is possible to sum over the outputs of $2^{n/2}$ carefully chosen subspaces in about 2^n time. This reduces the complexity of the attack to about $n \cdot 2^{3n/2}$ under the assumption that $p_f < 2^{-3n/4}$, which is stronger than the assumption made in the basic attack.

One way to optimize the summation process is to divide the n-bit block into two equal halves (assuming n is even). First, for each value of the $n/2$ most significant bits (MSBs), compute the sums over the outputs of all possible $2^{n/2}$ values of the $n/2$ least significant bits (LSBs). This gives an array of partial sums of size $2^{n/2}$ which is computed in time 2^n. Next, choose a subspace of dimension $d - n/2$ from the $n/2$ MSBs, and sum over the outputs of the bigger subspace of dimension d that includes the $n/2$ LSBs. Using the recomputed array, this can be done in time $2^{d-n/2}$. Repeating the process for $2^{n/2}$ subspaces of dimension $2^{d-n/2}$, the sums over the outputs of all of them can be computed in about 2^d time using the precomputed array. In total, we sum over the outputs of $2^{n/2}$ subspaces in about 2^n time, as claimed.

In general, instead of dividing the bits of the block into two halves, we can work with two orthogonal subspaces of dimension $n/2$. The pseudocode of the general procedure is given in Algorithm 2.

We consider slightly modify Definitions 3.3 and 3.4 which refer to subspaces selected according to the procedure of Algorithm 2. The analysis at the end of this section shows that the covering probability of the algorithm p_c remains roughly 2^{-n}, and thus the attack still requires evaluating $e \approx n \cdot 2^n$ subspaces. The sums over these subspaces are computed by running the procedure of Algorithm 2 about $n \cdot 2^{n/2}$ times.[3]

[3]Note that, in order to recover the final subspaces V_i, the base $n/2$-dimensional subspace T in the full attack must be randomized frequently enough (as done in our algorithm), since the intersection of d-dimensional subspaces produced with the same choice of T will always contain T.

Algorithm 2: Efficient Sum over Subspaces

```
   Input  : An integer $d > n/2$
   Output : Sums on outputs of $2^{n/2}$ $d$-dimensional subspaces $s_j$
1  begin
2      Pick a subspace $T$ of dimension $n/2$ at random;
       // Array of $2^{n/2}$ auxiliary sums
3      $A \longleftarrow A[1,\ldots,2^{n/2}]$;
       // Iterate over the $n/2$-dimensional subspace orthogonal to $T$
4      for each $x \in T_\perp$ do
           // A is accessed using an $n/2$-bit representation of $x$
5          $A[x] \leftarrow \sum_{y \in T} F(x+y)$;
6      end
7      for $j \leftarrow 1$ to $2^{n/2}$ do
8          Pick a subspace $T' \subset T_\perp$ of dimension $d - n/2$ at random;
9          $s_j \leftarrow \sum_{x \in T'} A[x]$;
10     end
11 end
```

Recall from Proposition 3.6 that the work spent on false alarms is about $2^d \cdot (e \cdot p_f)^2 \approx 2^{3n} \cdot (p_f)^2$. Therefore, in order to run in the claimed time complexity of about $n \cdot 2^{3n/2}$, we need to strengthen Assumption 3.2 and assume that the false alarm probability satisfies $p_f < 2^{-3n/4}$ (and this assumption was supported by our experiments, as noted above).

Analysis of Covering Probability

Fix a subspace T of dimension $n/2$ and write $V_i = W_i \oplus W'_i$, where $W_i \subseteq T$ and $W'_i \subseteq T_\perp$. For a random subspace $T' \subseteq T_\perp$ of dimension $d - n/2$, $V_i \subseteq T \oplus T'$ if and only if $W'_i \subseteq T'$. In order to estimate p_c, we distinguish between two cases according to the S-box size m: either $m = n/2$, or $m \leq n/3$. If $m \leq n/3$, the worst (and likely) case for our attack is $dim(W'_i) = dim(V_i) = m$. In this case, (reusing the analysis of Proposition 3.4) p_c remains about 2^{-n} for our choice of T' of dimension $d' = n/2 - t$ (up to the small multiplicative factor $1 - 2^{m-d'-1} = 1 - 2^{m+t-n/2-1} \approx 1$.

For $m = n/2$ the previous worst-case analysis assumes $dim(W'_i) = dim(V_i) = n/2$ and gives $p_c = 0$, as we need to cover the $n/2$-dimensional subspace W'_i with a smaller subspace T' of dimension $n/2 - 2$. It is possible to show that in this case p_c remains roughly 2^{-n} using the randomness in the choice of T. However, it is somewhat simpler to slightly tweak the algorithm, and use subspaces T of dimension $n/2 - 3$, implying $dim(T_\perp) = n/2 + 3$ and $d' = dim(T') = d - (n/2 - 3) = n/2 + 1$. Therefore, even if $dim(W'_i) = dim(V_i) = n/2$, then $p_c \geq 2^{-n} \cdot (1 - 2^{m-d'-1}) = 2^{-n} \cdot (1 - 2^{-2}) \approx 2^{-n}$. The tweaked algorithm sums over $2^{n/2}$ subspaces of dimension 2^{n-2} in time $2^{n/2} \cdot 2^{d'} = 2^{n+1}$ which is slower compared to what is claimed, but only by a small multiplicative factor of 2.

3.4 Boomerang Attack

3.4.1 Distinguishing Boomerang Attack on the Core $S_1 \circ L_1 \circ S_0$

The boomerang attack [Wag99], introduced by Wagner in 1999, allows for breaking a cipher with short high-probability differentials (rather than one long low-probability differential), also see Section 1.3.4. When we consider the core of the ASASA construction, the $S_1 \circ L_1 \circ S_0$ layer, it can be easily identified that very good short differentials exist for the S_0 and the $S_1 \circ L_1$ layers.[4] For each such layer, it is easy to see that many single active S-box differentials exist.

Consider the first layer S_0, it is easy to see that one can pick any of the $t \cdot (2^m - 1)$ possible input differences composed of a single active S-box, and obtain differentials (which in this degenerate case correspond to differential characteristics), each with probability of at least $2 \cdot 2^{-m}$. Similarly, one can find $t \cdot (2^m - 1)$ possible output differences composed of a single active S-box in the second layer. Again, each such differential has a probability of at least $2 \cdot 2^{-m}$.

One advantage of the boomerang attack is that once the input difference to the first layer is fixed, or once the output difference of the second layer is fixed, one can use multiple differentials (see [BDK01] for the full analysis). So a simple boomerang distinguisher for the core of $S_1 \circ L_1 \circ S_0$ can be easily constructed as follows:

- Pick an input difference α with one active S-box,
- Pick an output difference δ with one active S-box,
- Generate $c \cdot 2^{2m}$ boomerang quartets (pick a pair with input difference α, encrypt, XOR each ciphertext with δ, and decrypt the newly obtained ciphertexts), and expect a boomerang quartet.

We note that for a random permutation, in $c \cdot 2^{2m}$ boomerang quartets, we expect $cd \cdot 2^{2m} \cdot 2^{-mt}$ quartets such that the newly decrypted plaintexts satisfy that their difference is α (as this is an mt-bit condition). For the core we discuss the probability of a quartet to become a right one is:

$$\begin{aligned}
p_{boomerang} &= \sum_{\beta,\gamma} \Pr{}^2[\alpha \xrightarrow{S_0} \beta] \cdot \Pr{}^2[\gamma \xrightarrow{S_1 \circ L_1} \delta] \\
&= \underbrace{\left(\sum_{\beta} \Pr{}^2[\alpha \xrightarrow{S_0} \beta]\right)}_{(*)} \cdot \underbrace{\left(\sum_{\gamma} \Pr{}^2[\gamma \xrightarrow{S_1 \circ L_1} \delta]\right)}_{(**)} \\
&\geq 2^{-m+1} \cdot 2^{-m+1} = 2^{-2m+2}
\end{aligned}$$

The last transition follows the fact that the sums $(*)$ and $(**)$, reach minimal value for S-boxes which are 2-differentially uniform, i. e. for any given non-zero input (or output) difference to (from) the S-box, there are exactly 2^{m-1} possible output (input) differences, each with probability $2 \cdot 2^{-m}$.

[4]We note that we can decompose the core into $L_1 \circ S_0$ and S_1, and obtain essentially the same results.

Hence, for $c = 1/4$, we expect one right boomerang quartet for the core $S_1 \circ L_1 \circ S_0$. Moreover, as the actual number of right quartets follows a Poisson distribution, for $c = 1/4$, we indeed expect to get (at least) one quartet with probability of 63%. Obviously, increasing c allows a better success rate.

To conclude, there are several (actually, $(t \cdot (2^m - 1))^2$) boomerang distinguishers for the core. Each such distinguisher can be applied using $1/4 \cdot 2^{2m}$ quartets, i. e. 2^{2m} adaptive chosen-plaintexts and ciphertexts. The identification of the right quartets is immediate, and the memory complexity is restricted to storing a single quartet each time.

3.4.2 Extending the Attack to $L_2 \circ S_1 \circ L_1 \circ S_0$

The main problem that prevents a simple adaptation of the above attack to two full layers is the fact that due to the L_2 layer, we cannot identify by which δ we need to XOR the ciphertexts to obtain the new ciphertexts. This prevents an adaptive chosen-plaintext and ciphertext attack, and forces us to use a chosen-plaintext attack (as we can still control the α difference). To this end, we just transform the above boomerang attack into an amplified boomerang attack [KKS01] (or the rectangle attack [BDK01]).

The amplified boomerang attack starts with pairs of plaintexts $(m_i, m'_i = m_i + \alpha)$, encrypted into (c_i, c'_i). If the amplified boomerang condition holds for some quartet $((m_i, m'_i), (m_j, m'_j))$, then we know that the partial encryption of m_i and m_j through $S_1 \circ L_1 \circ S_0$ have difference δ (of one active S-box), and that the same holds for m'_i, m'_j. Unlike the case of the boomerang attack on the core, we do not have the differences $m_i + m_j$ and $m'_i + m'_j$ exposed to us as δ. However, we know that L_2 is a linear transformation, namely,

$$\begin{aligned}
S_1(L_1(S_0(m_i))) + S_1(L_1(S_0(m_j))) = \delta &\Rightarrow L_2(S_1(L_1(S_0(m_i)))) + L_2(S_1(L_1(S_0(m_j)))) = L_2(\delta) \\
&\Rightarrow c_i + c_j = L_2(\delta) \\
S_1(L_1(S_0(m'_i))) + S_1(L_1(S_0(m'_j))) = \delta &\Rightarrow L_2(S_1(L_1(S_0(m'_i)))) + L_2(S_1(L_1(S_0(m'_j)))) = L_2(\delta) \\
&\Rightarrow c'_i + c'_j = L_2(\delta) \\
&\Rightarrow c_i + c_j = c'_i + c'_j \Rightarrow c_i + c'_i = c_j + c'_j
\end{aligned}$$

The result is an amplified boomerang attack which takes $c \cdot 2^{n/2+m}$ pairs with input difference α, and searches for the right quartets, which can be identified by the fact that for right quartets, the above condition (which can be checked by analyzing a pair $(m_i, m'_i = m_i + \alpha)$ and storing in a hash table the value $c_i + c'_i$ of the corresponding ciphertexts).

The analysis of the number of right amplified quartets is relatively straightforward, and we obtain that of the $c^2 \cdot 2^{n+2m}$ possible quartets[5] about $4c^2$ are right quartets. Hence, for the correct value of $L_2(\delta)$ we expect to encounter $4c^2$ quartets (identified as collision in the hash table).

Given the large number of quartets, we expect wrong quartets to offer collisions in the hash table. There are going to be (about) additional $c^2 \cdot 2^{2m}$ such collisions in the table, but as they happen randomly, they are going to be scattered over the 2^n possible $L_2(\delta)$ values. Hence, as long as $4c^2 \gg c^2 \cdot 2^{2m-n}$, we expect the right value to be identified.

[5]We refer the interested reader to [BDK01] for the full analysis, e. g. why there are $c^2 \cdot 2^{n+2m}$ rather than $c^2/2 \cdot 2^{n+2m}$ quartets.

The result is an attack that takes $c \cdot 2^{n/2+m}$ chosen plaintexts, time, and memory, and identifies $L_2(\delta)$. We later show how to transform this knowledge into (partial) key recover attack on L_2.

A Small (and somewhat insignificant) Technicality

We note that the probability estimation that we have used assumes that the S-box in use is differentially 2-uniform. Obviously, if the S-boxes are chosen at random, it is highly unlikely that this condition holds. In these circumstances, the probability of the boomerang $p_{boomerang}$ is actually slightly higher.

We can use [OCo94; OCo95] to evaluate the way the difference distribution table behaves for an S-box chosen at random. Namely, an entry in the difference distribution table of an m-bit S-box (besides those involved with input/output zero), has a value distributed according to $2 \cdot Poi(1/2)$. As a result, the sums $(*)$ and $(**)$ are expected to obtain the value:

$$\begin{aligned}(*) &= \sum_{\beta \neq 0} Pr^2[\alpha \xrightarrow{S_0} \beta] = (2^m - 1) \cdot 2^{-2m} \cdot \sum_{i=0}^{2^{m-1}} (2 * i)^2 \cdot e^{-1/2} \cdot (1/2)^i / i! \\ &\approx 3 \cdot 2^{-m}\end{aligned}$$

(compared with the value $2 \cdot 2^{-m}$ which is a lower bound).

A Small (but Important) Technicality

We note that the amplified boomerang attack is being run in parallel for all $t \cdot (2^m - 1)$ possible values of δ, each resulting in about $4c^2$ quartets. Hence, we can obtain almost an exhaustive list of all the output differences of L_2 that originate from a single active S-box.

Due to the nature of the amplified attack, we *do not* need additional data to find all these values. They are just suggested by the different boomerangs. The only thing we need to take into account is that we want all $t \cdot (2^m - 1)$ boomerangs to "succeed" (i.e., have enough quartets). In Table 3.2 we give an example of parameters that follows AES ones ($t = 16, m = 8, n = 128$).

One additional technicality is the fact that there is some (non-zero) chance that differential characteristics with *two* active S-boxes in the second-layer, may also be encountered. A simple analysis reveals that the expected number of quartets for a given two-active-S-boxes differences is $4c^2 \cdot 2^{-m}$. Moreover, there are $\binom{t}{2} \cdot (2^m - 1)^2$ possible such differences. Hence, to avoid too much noise from these cases we must demand that about $t^2/2 \cdot 2^{2m}$ random variables following a Poisson distribution with a mean value of about $4c^2 \cdot 2^{-m+1}$ will not interfere with the correct differences (whose number of right quartets follows a Poisson distribution with a mean value of $4c^2$).[6] Luckily, it is easy to see that for reasonable values of c, the problem is far from causing trouble. Obviously, the problem generalizes to more active S-boxes in the second layer, but it

[6]We alert the reader that it is easy to overcome a "small" number of two-active-S-boxes differences in the other steps of the attack, and as we see later, they may even be useful for the attack. Moreover, one can start the exploration of the differences with those that are suggested by as many quartets as possible. These values are very likely to be with a single active S-box in S_2. Using the fact that if all non-zero differences of a single S-box form a linear subspace, one can check whether differences with a smaller amount of quartets is actually of single active S-box, and even complete the space for values that received no quartet.

Table 3.2: Number of differences discovered by the (amplified boomerang) attack as a function c for AES parameters ($t = 16, m = 8, n = 128$).

c	Expected Number of Quartets	Threshold (for ident.)	Discovered differences Single S-box (out of 4,080)	Discovered differences Two S-boxes (out of 7,803,000)
1	4	1	4,005	240,073
		2	3,706	3,732
		3	3,109	39
		4	2,311	0.3
2	16	3	4,079	2,313
		4	4,079	72
		5	4,078	2
		6	4,074	0.03

is (again) of no effect on the main result. We list in Table 3.2 also the expected number of two-active S-boxes differences that might pass for a cipher with AES parameters.

This leads to the following problem, which we address next: once we have all these $t\cdot(2^m-1)$, can we recover all (or part) of L_2?

Partial Key-Recovery

At this point, we obtain $t \cdot (2^m - 1)$ output differences $L_2(\delta_{i,j})$ where $\delta_{i,j}$ is a difference of j in the i'th S-box (and zero in all other S-boxes). We now show how to divide this difference into the t different S-boxes (as the order between them can be fixed arbitrarily, given that any order can be "adjusted" by using L_1).

The simplest method to do so is to actually increase a bit the data/time complexity and look for two active S-boxes in the second layer. It is easy to see that once we identify all $\delta_{i,j}$'s values, then a difference in two active S-boxes can be written as $\Delta = \delta_{i,j} + \delta_{i',j'}$, i. e. $L_2(\Delta) = L_2(\delta_{i,j} + \delta_{i',j'}) = L_2(\delta_{i,j}) + (\delta_{i',j'})$. This holds only when $i \neq i'$, and thus, we can identify when two of the one active S-box differences do not share an S-box (as there will be a corresponding two active S-box difference for them). Hence, the separation into different S-boxes can be done quite immediately, resulting in an attack that requires $c \cdot 2^{n/2+3m/2}$ chosen plaintexts, and about the same amount of time.

3.4.3 Attacking the Full Structure

We now turn our attention to the full ASASA construction. The main problem with directly applying the amplified boomerang attack to the full construction is that we cannot have a difference in the first S-box layer in only a single S-box.

Luckily, the solution to the problem is to perform another birthday paradox argument and use a known-plaintext boomerang. The known-plaintext boomerang was first mentioned in [Wag99], and it is a natural extension of the boomerang (and the amplified boomerang attack). A random set of $c \cdot 2^{3n/4+m/2}$ known plaintexts contains $c^2/2 \cdot 2^{3n/2+m}$ pairs at the entrance to S_0, i. e. offers $c^2/2 \cdot 2^{n/2+m}$ pairs with any given input difference, including those with a single active S-box. As shown earlier, such amount of pairs is sufficient to generate right amplified boomerang quartets.

The only remaining task is the identification of the quartets $((m_1, m_2), (m_3, m_4))$ themselves. If both (m_1, m_2) and (m_3, m_4) are pairs which are part of the quartet, in other words, have the same difference in a single byte after L_0 (i. e. $L_0(m_1 + m_2) = L_0(m_3 + m_4)$ with a single active S-box), then $m_1 + m_2 = m_3 + m_4$. As before, we can also detect that $L_2(c_1) + L_2(c_2) = L_2(c_3) + L_2(c_4)$. This suggests that finding the quartets $((m_1, m_2), (m_3, m_4))$ can be easily done by:

- For all pairs of plaintext (m_i, m_j) store in a hash table $m_i + m_j || c_i + c_j$ (along with m_i and m_j).
- Collect all collisions in the table as candidate quartets, and analyze them as before.

Given $c \cdot 2^{3n/4+m/2}$ known plaintexts, we expect $c^2/2 \cdot 2^{3n/2+m}$ pairs, and in total $c^4/2 \cdot 2^{3n+2m}$ quartets.[7] Hence, we expect $c^4/2 \cdot 2^{n+2m}$ quartets to be suggested by the collisions in the table, out of which $c^4/2$ are right quartets that suggest the correct differences in the plaintexts and in the ciphertexts. These differences are, of course, differences that become an active single S-box (either through L_0 or the inverse of L_2). Hence, once again, it is possible to identify all the differences that are transformed into a single active S-box (on both sides of the scheme). Again, increasing the data complexity a bit (to $c \cdot 2^{3n/4+3m/4}$ known plaintexts), allows finding all the differences that go to a single active S-box, by working with differences of two active S-boxes.

To conclude, one can easily identify the $t \cdot (2^m - 1)$ differences that lead to a single active S-box thorough L_0 or L_2, using a known-plaintext boomerang. The attack takes $c \cdot 2^{3n/4+3m/4}$ plaintexts and has memory and time complexity of $c^2/2 \cdot 2^{3n/2+3m/2}$.

3.5 Differential Attack: Using the DDT

We denote by
$$d(\alpha) := |\{F(x) + F(x + \alpha) \mid x \in \mathbb{F}_2^n\}|$$
the number of possible output differences for a given input difference.

The basic idea is that the number of possible output differences depends on the number of active S-boxes in the first layer of S-boxes. More precisely, we rely on the following assumption.

Assumption 3.3. *The number of possible output differences is (expected to be) smaller if only one S-box is active in the first layer compared to the case when more than one S-box is active in the first layer.* ◊

So the attack starts by computing $d(\alpha)$ for all non-zero α values. This takes time 2^{2n} and will be the bottleneck of the attack. Afterward, the list of all $d(\alpha)$ is sorted in increasing order. We denote the sorted list by $\mathscr{T}$ with $\mathscr{T}_0$ corresponding to the α with the smallest $d(\alpha)$ value.

[7]We note that the quartet $((m_1, m_2), (m_3, m_4))$ differs from the quartet $((m_1, m_2), (m_4, m_3))$.

The assumption is now that the top of the list contains mostly input differences α such that

$$L(\alpha) = (\beta_1, \dots, \beta_t)$$

with only one non-zero $\beta_i \in \mathbb{F}_2^m$. In other words, the top of the list contains mainly elements α such that $\alpha \in V_i$ for some (unknown) $1 \le i \le t$.

For recovering V_i we next have to sort those α values into different bins such that two α values are in the same bin if and only if they are both in the same subspace V_i.

We explain how to recover the first V_i, without loss of generality V_0 , in detail. Recovering the remaining ones is very similar and will only be briefly sketched.

We start by considering the first ℓ entries in the sorted table $\mathcal{T}$. Choosing ℓ such that $\binom{\ell}{m}$ is smaller than 2^{2n} makes sure that this step is not the bottleneck of the attack. In most cases (as long as $m \le 2^{2t}$) it would be sufficient to choose $\ell \le 2^{2t}$, as

$$\binom{2^{2t}}{m} \le \left(2^{2t}\right)^m = 2^{2n}.$$

However, our experiments show that the success probability of the attack is reduced for high values of ℓ. The best results have been obtained when ℓ was set to a small multiple of m.

The idea now is to identify a basis of V_0. For any choice of m (linearly independent) elements $\alpha_1, \dots, \alpha_m$ among the first ℓ elements of $\mathcal{T}$ we compute a penalty value

$$P(\{\alpha_1, \dots, \alpha_m\}) := \sum_{u \in \mathrm{span}(\{\alpha_1, \dots, \alpha_m\}), u \neq 0} d(u)\ ,$$

According to Assumption 3.3, there are now two cases to be considered. First, if all α_i values actually belong to V_0 the penalty $P(\{\alpha_1, \dots, \alpha_m\})$ is simply the sum over all $d(u)$ values for $u \in V_i$ and thus expected to be small. Second, if at least one α_j does not belong to V_0, at least $(2^{m-1} - 1)$, that is roughly half, of the elements in $\mathrm{span}(\{\alpha_1, \dots, \alpha_m\})$ do not belong to any V_i and thus the penalty value is expected to be significantly higher.

Thus, this procedure allows to identify a basis of V_0. As will be shown below in the experimental results, this is successful with very good probability.

After recovering V_0 we simply remove all elements in V_0 from the list $\mathcal{T}$ and repeat the same procedure again. Experimentally, the overall success probability, i. e. the probability that we recover all V_i correctly is again rather high, as shown in Table 3.3 below.

It can be seen that the success probability increases for increasing t and m, with the exception of $m = 3$. The reason for this exception might be linear components that occur in the randomly chosen 3-bit S-boxes.

Table 3.3: Result of 100 runs of the attack for various parameters. In each run, a new random ASASA instance has been generated. The numbers correspond to the success probability to recover all V_i correctly. The dashes correspond to experiments we did not run due to limited resources.

m \ t	2	3	4	5
3	0.16	0.06	0.02	0
4	0.78	0.92	1	—
5	1	1	—	—
6	1	—	—	—
7	1	—	—	—
8	1	—	—	—

3.6 Conclusion and Further Research

In this chapter, we analyzed efficient decomposition algorithms for the ASASA structure. In particular, we described three different attack vectors, two based on differential properties and one on an integral – i. e. a structural – property. In all the attacks, the difficulty lies not in exploiting the properties, but rather in locating them as they are hidden by the external linear layers. Reciprocally, as soon as such a property has been detected, information on the linear layers can be easily deduced, and combining the data gathered from several properties allows to peel off the external linear layers efficiently. While our integral attack is the most efficient, it is not always applicable, and in such cases the other attacks should be considered.

As our most efficient attacks have time complexity of roughly $n \cdot 2^{3n/2}$, it seems that ASASA schemes must have a large block size n in order to be considered as secure candidates for white-box cryptography. However, this requires their black-box representations to be extremely large and inappropriate for practical use (e. g. in order to guarantee a minimal security level of 64 bits, an ASASA scheme requires storage of about 2^{43} words, which is more than 10 terabytes).

A natural future work item is to study the security of general SP-networks with more layers and to investigate if any of the attacks presented here can be generalized or improved on these constructions. This was picked up by Biryukov and Khovratovich, who extended the existing attacks to structures with up to 9 layers [BK15]. Afterward, this was generalized to a closed formula describing the number of attackable rounds r of the $A(SA)^r$ construction depending on the size and number of S-boxes [BKP16].

Subsequent research on affine equivalence algorithms by Dinur [Din18] led to an improvement of the integral attack presented in Section 3.3. In particular, the efficient computation of the sum over the subspaces from Section 3.3.3 can be improved by using a symbolic algorithm, leading to an overall complexity of about $n \cdot 2^n$.

Despite all these attacks, the ASASA construction was actually used in a recent design for a family of stream ciphers called RASTA [Dob+18]. The authors used it in a setting such that the presented attacks are not feasible.

A possible direct application of cryptanalytic attacks on ASASA is the research problem of reverse engineering secret S-boxes [BPU16]. However, secret building blocks of ciphers are of course not restricted to SPN structures. Results in this area have also been published for Feistel structures [BLP16; PU16].

Finally, another obvious research problem motivated by our work is to continue the vivid research on white-box cryptography and to design improved, secure, and practically usable schemes. There has been quite some work after the presentation of our findings in 2015. A white-box design that completely avoids the attacks from above was presented by Bogdanov and Isobe [BI15]. Afterward, Fouque et al. made a first proposal for provable and efficient white-box cryptography [Fou+16]. Further research on efficient and secure white-box schemes was conducted for example in [Cho+16] and [BIT16].

Part III

Lightweight Linear Layers

4 Lightweight Multiplication in $GF(2^m)$ with Applications to MDS Matrices

The results from this paper have been published in the proceedings of the IACR conference *CRYPTO 2016* [BKL16] and presented at that conference in Santa Barbara, USA. This is joint work with Christof Beierle and Gregor Leander. All authors equally contributed. The main contributions of the thesis author are in the construction of lightweight MDS matrices (Section 4.4).

4.1 Introduction

Many cryptographic schemes build on finite fields as their underlying mathematic structure. In almost all cases, the schemes can be designed without having to specify a concrete representation of the finite field in advance. However, when finally being implemented in practice, one necessarily has to choose a particular representation of the finite field, basically as bit strings. In general, this choice does not influence the security of the scheme, but might well influence the performance of the resulting implementation. In this chapter, we focus on this choice of field representations and derive theoretical results on how to choose an optimal field representation with respect to multiplication with fixed field elements. Before going into details, we elaborate on this setup in the special case of symmetric cryptography.

Today, we are in the comfortable situation of having at hand a choice of strong block ciphers and hash functions that seem secure against even the strongest adversaries with practically unlimited computational resources. As discussed in Section 1.2.1, those primitives are based on rather well-understood design principles that allow constructing efficient, simple and easy to analyze ciphers. Especially in the case of substitution-permutation (SP) networks, following the seminal ideas of AES [DR98] and its predecessor SQUARE [DKR97], arguing for the security of ciphers against the two most powerful generic attacks, that is, differential and linear attacks [BS91; Mat94], became significantly easier (see Section 1.3.3 and Chapter 2). Recall from Section 1.2.1 that the cipher (or the cryptographic permutation) in an SP-network consists of a number of almost identical rounds, each of which consists of a layer of S-boxes and an $\mathbb{F}_2$-linear layer to mix those parts.

One of the most important design strategies for those primitives is the so-called wide trail strategy (see Section 1.2), initiated in [Dae95]. Here, the main observation is that it is actually the linear layer that is to a large extent responsible for the security of the primitive against linear and differential attacks. Linear layers are often designed via MDS matrices defined over $\mathbb{F}_{2^m}$. As discussed in Section 1.2.2, one then has to choose a mapping on $\mathbb{F}_{2^m}^t$ and an $\mathbb{F}_2$-basis of $\mathbb{F}_{2^m}$ to concretely specify the primitive. This is actually a very natural separation of the design of the cipher and its specification (and thus implementation) on the bit-level. As nicely explained in [DR11] by introducing RIJNDAEL-GF this separation is probably most obvious for AES itself, but in principle possible for any cipher. Following [DR11], the choice of basis is to a large extent independent of the design and the security of the cipher. However, the choice of basis might have a significant impact on the efficiency of the cipher on certain platforms. This is actually a very natural separation of the design of the cipher and its specification (and thus implementation) on the bit-level. As nicely explained in [DR11] by introducing RIJNDAEL-GF this separation is probably most obvious for AES itself, but in principle possible for any cipher. Following [DR11], the choice of basis is to a large extent independent of the design and the security of the cipher. However, the choice of basis might have a significant impact on the efficiency of the cipher on certain platforms.

For software implementations, depending on the details, the choice of basis is either irrelevant (e. g. in a table-based implementation) or hard to capture (e. g. in a bit-sliced implementation) as the efficiency might depend on the exact instructions offered by a given platform. For hardware implementations, one has to distinguish between a serial implementation or a round-based implementation. As the round-based implementation seems most relevant in practice (see [Sim+15]), we mainly focus on this use-case here. Surprisingly, compared to a serial hardware implementation, the case of a round-based hardware implementation has attracted less attention so far.

For a round-based hardware implementation, the impact of the choice of basis already becomes apparent when focusing on how to implement the multiplication with one given element α in $\mathbb{F}_{2^m}$. For different choices of bases, the efficiency of implementations of the resulting $\mathbb{F}_2$-linear mappings differs significantly. Thus, the very fundamental task we study in the first part of this chapter is:

For a given element $\alpha \in \mathbb{F}_{2^m}$ find a basis such that multiplication by α can be implemented most efficiently.[1]

It is worth pointing out that the related question of how to efficiently multiply *two* arbitrary field elements has been studied extensively in the past.

[1]Note that the choice of basis is of course not restricted to choosing different irreducible polynomials to represent the finite field.

While the above question is of independent interest, with potentially very different applications, we use our results for designing efficient linear layers. Thus, in the second part, we will give several constructions of MDS matrices. Echoing the above, the construction of our MDS matrices are independent of the choice of the basis – actually to a large extent independent of the field size as well.

The combination of the first part, i. e. how to choose a basis that allows for an optimal implementation, and the second part, i. e. the construction of MDS matrices, finally results in implementations of MDS matrices that are more efficient for a large variety of parameters than the best matrices discussed so far in literature.

Thus, this application serves as a nice example were an improved understanding of how to choose the field representation immediately leads to improved results. This is even more interesting as the construction of efficient MDS matrices has been an active field of research recently.

4.1.1 Previous Research

In particular, the construction of efficient serial MDS matrices is a well-studied subject. Considering serial implementations of MDS matrices is based on the initial idea of Guo et al. used in the design of PHOTON [GPP11] and later in the block cipher LED [Guo+11]. In a nutshell, the idea is not to implement an MDS matrix directly, but rather implement a matrix A such that A^d is MDS for some small d. When considering a hardware implementation, it reduces the chip area if implementing A is significantly cheaper than A^d. The circuit implementing A is then iterated d times, which does not increase its size significantly. This basic idea has been further generalized and improved in a series of subsequent papers. In [Saj+12b] and [WWW13] the authors focus on even more efficient choices for A by considering additive, i. e. $\mathbb{F}_2$-linear MDS codes. Their approach uses symbolic computations in order to derive general conditions on how to choose the matrix entries independent of the dimension.

In [XZL14] Xu et al. furthermore took into account the cost of implementing the inverse matrix. At FSE 2014, Augot and Finiasz [AF15] improved significantly upon the efficiency of the search algorithm of [Saj+12b], allowing them to search for MDS matrices of a much larger dimension than previously possible.

For a round-based implementation, less work has been done so far. The authors of [Sim+15] focus on MDS matrices that have an efficient implementation (in terms of the XOR count) and put special emphasis on involutory MDS matrices, i. e. MDS matrices that are their own inverse. They derive several constructions and rather efficient search methods for MDS matrices meeting their goals. Liu and Sim [LS16] improved upon some of those results by characterizing equivalences in circulant (and circulant-like) MDS matrices and thus further reducing the search space. In both works, in order to improve the efficiency for a given MDS matrix defined over a finite field, the authors considered different representations of the underlying finite fields by running through all possible irreducible polynomials of the given degree. However, in view of the question of how to choose an optimal basis, this corresponds to investigating only a small subset of all possible bases. Work on investigating the XOR count distribution for other than the polynomial bases has been done in [SS16a].

Li and Wang constructed circulant involutory $\mathbb{F}_2$-linear MDS matrices [LW16]. While it was already known that circulant MDS matrices over a finite field cannot be involutory [GR14], they have shown their existence in the additive case. Independently, Liu and Sim [LS16] have shown the existence of cyclic involutory MDS matrices over finite fields, where cyclic matrices are closely related to circulant matrices.

4.1.2 Our Contribution

After fixing our notation and recalling basic facts in Section 4.2, in the first part of the paper we focus on the question on how to find an optimal implementation of the multiplication by a given field element α (see Section 4.3). Here efficiency is measured in terms of the number of XOR operations needed to implement the corresponding binary matrix. Note that this metric differs from the XOR count used in [Sim+15]. In [Sim+15] the XOR count of an $m \times m$ matrix M was defined as the number of ones in M minus m. However, the number of (additional) ones in a matrix does not necessarily correspond to the number of XOR operations needed for implementation. Thus, while the number of ones in M is certainly an easier to handle metric, in our opinion it is more appropriate to consider the actual number of XOR operations as the efficiency metric. Note that this improved notion was first introduced in [Jea+17b]. For technical reasons, we focus on the number of XOR operations without temporary registers, i. e. in-place XOR operations. One of our main results in this first part of the paper is, that for a non-trivial element α one can find a basis such that the resulting matrix can be implemented with one XOR operation if and only if the characteristic polynomial of α is an irreducible trinomial. Note that an XOR count equal to one in our notion coincides with the definition of the XOR count in [Sim+15]. The interesting part here is that the condition on the characteristic polynomial is not only sufficient but also necessary. As an immediate consequence, one cannot hope to implement the multiplication by any element $\alpha \neq 1$ in $\mathbb{F}_{2^8}^*$ with one XOR only. This follows by the above and the well-known fact that there are no irreducible trinomials of degree 8 [Swa62].

We furthermore show that, for any given basis, there are at most two (non-trivial) elements α and β such that the multiplication with those elements can be implemented with one XOR operation. In fact, β is necessarily the multiplicative inverse of α.

While the weight of the (irreducible) characteristic polynomial of an element α clearly gives an upper bound of the number of XOR operations needed to implement the corresponding multiplication, we show that this bound is in general not tight in the case where the characteristic polynomial is of weight larger than three.

In particular, for all elements $\alpha \in \mathbb{F}_{2^m}^*$ with $m \leq 8$ we present an optimal representation such that the multiplication with α can be implemented with a minimal number of XOR operations. For all those elements α, that are not contained in a proper subfield of $\mathbb{F}_{2^m}$, the multiplication can be implemented with at most 3 XOR operations (and often with two only). Those results are given in Tables 4.1 and 4.5 and cover the cases which are most relevant for symmetric cryptography. Interestingly, and maybe counter-intuitive, multiplication with non-trivial elements in a proper subfield turns out to be among the most expensive in all the cases explored here.

Moreover, for all $m \leq 2048$ for which no irreducible trinomial of degree m exists, we present one element $\alpha \in \mathbb{F}_{2^m}$ such that multiplication by α requires two XOR operations, see Table B.1 in the appendix. Those results are proven optimal by the above mentioned necessary and sufficient condition.

In the second part of this chapter (see Section 4.4) we present several (circulant) matrices. Entries in those matrices are represented as powers of a generic field element α. By symbolically computing all minors, i. e. the determinants of all square submatrices, we derive a list of polynomials in $\mathbb{F}_2[\alpha]$. Now, whenever α is chosen such that it is not a root of any of those polynomials, the matrix is MDS. One nice consequence of this approach is that, as the degree of those polynomials is limited, our matrices are MDS for almost all elements in $\mathbb{F}_{2^m}$ as soon as m is large enough, i. e. larger than the maximal degree of those polynomials.

Finally, the first and second part are combined in Section 4.4.2 to result in efficient MDS matrices. When these results were first presented in 2016, those were the most efficient MDS matrices in terms of the XOR count. A summary of our results and comparison with previous work is given in Table 4.6 and Table 4.7, respectively. The main observation here is that if multiplication by α can be implemented with d XOR operations, then multiplication by $\alpha^{\pm i}$ for $i \geq 0$ can be implemented with at most $d \cdot i$ XOR operations.[2] Thus, by simply minimizing the sum of the (absolute) exponents for our circulant MDS matrices, we immediately reduce the XOR count.

As an interesting side result, we like to point out that the *XOR count per bit actually decreases with increasing field size*.[3] For example, our 4×4 MDS matrices have a per bit XOR count of $3 + \frac{3}{m}$, or $3 + \frac{6}{m}$ in the case that no irreducible trinomial of degree m exists.

Thus, even so reducing the number of XOR operations has already received considerable attention, this part nicely shows that our improved understanding of how to choose an optimal basis allows us to easily improve upon previous constructions. Note that such improvements are possible independent from which XOR count definition is used, that is, we were able to improve existing results also in the old XOR count definition by changing the basis. For example, we found an element in $\mathbb{F}_{2^8}$ with only 2 additional non-zero entries which directly improves the results of [Sim+15].

Finally, in Section 4.5 we give a perspective on non-linear, additive MDS matrices. In particular, we point out that while there exists no $\alpha \in \mathbb{F}_{2^8}$ (resp. $\mathbb{F}_{2^{13}}$, $\mathbb{F}_{2^{16}}$) which can be implemented with only one XOR operation, there does exist an 8×8 (resp. 13×13, 16×16) binary matrix, that can be used in place for the multiplication by α in the above mentioned 4×4 matrix to result in an additive MDS matrix with reduced cost.[4] Again, the idea of considering the entries of the matrix as powers of a single field element is beneficial as the conditions for the matrix to be MDS remain basically unchanged.

We conclude the paper by pointing to some interesting questions for future investigations.

[2]It is exactly this part where considering only in-place XOR operations becomes very helpful, as otherwise multiplication by α and by α^{-1} might differ in their XOR count.

[3]This is also true for the constructions given in [WWW13], but does not hold for the subfield (or code-interleaving) construction.

[4]Note that the authors of [LW16] recently constructed a similar 32×32 $\mathbb{F}_2$-linear MDS matrix.

4.2 Preliminaries

The basic concepts and notations of finite field representations were introduced in Section 1.2.2. The multiplicative group of some field K is denoted by K^*. The ring of $m \times m$ matrices over a field K will be denoted by $\mathrm{Mat}_m(K)$ in this chapter. The symbol $\mathbf{0}_m$ will denote the *zero matrix* and I_m will be the *identity matrix*. As a third important type of matrix in $\mathrm{Mat}_m(\mathbb{F}_2)$, we introduce $E_{i,j}$ which consist of all zeros except in the i-th row of the j-th column for $i, j \in \{1, \dots, m\}$. We denote a block diagonal matrix consisting of d matrix blocks A_k as $\bigoplus_{k=1}^{d} A_k$. By $\mathrm{hw}(A)$, we denote the number of non-zero entries of a matrix A. Analogously, $\mathrm{hw}(q)$ denotes the number of non-zero coefficients of a polynomial q.

4.2.1 Some Basic Facts about Linear Transformations

Recall from Section 1.2.2 that the multiplication by an element $\alpha \in \mathbb{F}_{2^m}$ can be described by a matrix $T_{\alpha,B}$. This representation depends on the basis B for $\mathbb{F}_{2^m}$ and changing the basis from B to B' results in a different matrix representation. This transformation is called the *change of basis* transformation, which is simply a conjugation of $T_{\alpha,B}$. Thus, $T_{\alpha,B'} = ST_{\alpha,B}S^{-1}$ using an invertible matrix S. In this case, $T_{\alpha,B}$ and $T_{\alpha,B'}$ are called *similar* (resp. *permutation-similar* if S is a permutation matrix). We denote similarity of matrices with the relation symbol $\sim$, (resp. $\sim_\pi$ for permutation-similarity). The *characteristic polynomial* of a matrix A is defined as $\chi_A := \det(\lambda I_m - A) \in \mathbb{F}_2[\lambda]$ and the *minimal polynomial* is denoted by m_A. Recall that the minimal polynomial is the (monic) polynomial p of least degree, such that $p(A) = \mathbf{0}_m$. It is a well-known fact that the minimal polynomial divides the characteristic polynomial, thus $\chi_A(A) = \mathbf{0}_m$. As the minimal polynomial and the characteristic polynomial are actually properties of the underlying linear mapping, similar matrices have the same characteristic and the same minimal polynomial.

A special type of matrix, that will play an important role in the following is the companion matrix of a polynomial. For a polynomial

$$q = x^m + q_{m-1}x^{m-1} + \dots + q_1 x + q_0 \in \mathbb{F}_2[x]$$

of degree m, the *companion matrix* of q is defined as

$$C_q = \begin{pmatrix} 0 & & & & q_0 \\ 1 & 0 & & & q_1 \\ & \ddots & \ddots & & \vdots \\ & & 1 & 0 & q_{m-2} \\ & & & 1 & q_{m-1} \end{pmatrix}.$$

It is known from linear algebra that the characteristic polynomial and the minimal polynomial of C_q are equal to q itself, i. e. $\chi_{C_q} = m_{C_q} = q$. In addition, any matrix A is similar to a companion matrix if and only if its characteristic polynomial coincides with its minimal polynomial. In particular, C_q is exactly the *rational canonical form* [DF04, Section 12.2] of A in this case.

4.2.2 The XOR Count and the Cycle Normal Form

The *XOR count* of a field element was already studied in [Kho+14] and [Sim+15]. In the formal definition in [Sim+15], an invertible m-dimensional matrix A has an *XOR count* of d if and only if A can be written as a permutation matrix with d additional non-zero entries. Formally, $A = P + \sum_{k=1}^{d} E_{i_k,j_k}$ and $\mathrm{hw}(A) = m + d$. Although all matrices of that structure can be implemented with at most d XOR operations (not necessarily without temporary registers), the construction does not contain all possible matrices which are realizable with at most d XOR operations. For instance, there are matrices with three additional non-zero entries such that the result of their defining linear function can be computed with just two additions. As an example, consider

$$\begin{pmatrix} 1 & 0 & 1 \\ 1 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} v_1 \\ v_2 \\ v_3 \end{pmatrix} = \begin{pmatrix} v_1 + v_3 \\ (v_1 + v_3) + v_2 \\ v_3 \end{pmatrix}.$$

In the following, we provide an alternative definition which includes the cases described above. Note that the improved notion of the XOR count was first introduced in [Jea+17b].

Definition 4.1. An invertible matrix A has an *XOR count of d*, denoted $\mathrm{wt}_{\oplus}(A) = d$, if d is the minimal number such that A can be written as

$$A = P \prod_{k=1}^{d} (I + E_{i_k,j_k})$$

with $i_k \neq j_k$ for all k. ◊

Note that if a matrix can be represented in the form $P \prod_{k=1}^{d} (I + E_{i_k,j_k})$, the number of factors $(I + E_{i_k,j_k})$ clearly gives an upper bound on the actual XOR count. It is worth pointing out that the definition above just counts the number of XOR operations without using temporary registers. Those are technically somewhat easier to handle. However, this restriction does not make a difference for matrices with XOR count less or equal to 2, which we are most concerned about in the following. In general, allowing temporary registers might well reduce the number of XOR operations needed for an implementation.

Our definition coincides with the one from [Sim+15] for the case that $d = 1$, that is, for matrices of XOR count 1. For other cases, the number of additional non-zero entries can increase. We will often consider $d = 2$ within this work. By evaluating the product, it follows that any A with $\mathrm{wt}_{\oplus}(A) = 2$ is of the form

$$A = \begin{cases} P + P(E_{i_1,j_1} + E_{i_2,j_2}) & \text{iff } i_2 \neq j_1 \\ P + P(E_{i_1,j_1} + E_{i_2,j_2} + E_{i_1,j_2}) & \text{iff } i_2 = j_1. \end{cases}$$

The XOR count is invariant under permutation-similarity. Moreover, naturally in the setting not allowing temporary registers, the XOR count is invariant under taking the inverse. This is summarized and formally proven in the following Lemma and Corollary.

Lemma 4.1. *If $A \sim_{\pi} A'$, then* $\mathrm{wt}_{\oplus}(A) = \mathrm{wt}_{\oplus}(A')$. ◊

Proof. Let $A' = QAQ^{-1}$ where Q is the permutation matrix representing the permutation $\sigma \in S_m$. Let $I + E_{i_k,j_k}$ be a factor in the XOR count representation of $A = P \prod_{k=1}^{d} (I + E_{i_k,j_k})$ where $d = \mathrm{wt}_{\oplus}(A)$. Then the following identity holds:

$$(I + E_{i_k,j_k})Q^{-1} = Q^{-1} + E_{i_k,\sigma^{-1}(j_k)} = Q^{-1}(I + E_{\sigma^{-1}(i_k),\sigma^{-1}(j_k)}).$$

One is able to commute Q^{-1} to the front before the first factor by proceeding for all of the d factors and finally obtain

$$A' = QPQ^{-1}\prod_{k=1}^{d}(I + E_{\sigma^{-1}(i_k),\sigma^{-1}(j_k)}).$$

It follows that $\mathrm{wt}_{\oplus}(A') \leq \mathrm{wt}_{\oplus}(A)$. By reverting the above steps we obtain $\mathrm{wt}_{\oplus}(A) \leq \mathrm{wt}_{\oplus}(A')$. □

Corollary 4.1. *If* $\mathrm{wt}_{\oplus}(A) = d$*, then also* $\mathrm{wt}_{\oplus}(A^{-1}) = d$. ◊

Proof. We show that A^{-1} is permutation-similar to a matrix with an XOR count of d.

$$\left(P\prod_{k=1}^{d}(I + E_{i_k,j_k})\right)^{-1} = \prod_{k=d}^{1}(I + E_{i_k,j_k})P^{-1} \sim_{\pi} P^{-1}\prod_{k=d}^{1}(I + E_{i_k,j_k})$$

□

Later, we would like to be able to exhaustively search over all matrices with low XOR count for a given dimension m. Since the number of permutation matrices (which is $m!$) rapidly increases with m, an exhaustive search will quickly become infeasible if we do not restrict the structure of P. By a well-known fact from combinatorics, one is able to assume P to be in a specific form.

Lemma 4.2. *For any permutation matrix P of dimension m, it is*

$$P \sim_{\pi} \bigoplus_{k=1}^{d} C_{x^{m_k}+1}$$

for some m_k with $\sum_{k=1}^{d} m_k = m$ and $m_1 \geq \cdots \geq m_d \geq 1$. ◊

Proof. It is well-known that two permutations with the same cycle type are conjugate [DF04, Chapter 4.3, Proposition 11]. That is, given the permutations $\sigma, \tau \in S_m$ as

$$\sigma = (s_1, s_2, \ldots, s_{d_1})(s_{d_1+1}, \ldots, s_{d_2})\ldots(s_{d_{m-1}+1}, \ldots, s_{d_m})$$
$$\tau = (t_1, t_2, \ldots, t_{d_1})(t_{d_1+1}, \ldots, t_{d_2})\ldots(t_{d_{m-1}+1}, \ldots, t_{d_m})$$

in cycle notation, one can find some $\pi \in S_m$ such that $\pi\sigma\pi^{-1} = \tau$. This π operates as a relabeling of indices.

Let σ in the form above be the permutation defined by P. Now, there exists a permutation π such that $\pi\sigma\pi^{-1} = (d_1, 1, 2, \ldots, d_1 - 1)(d_2, d_1 + 1, d_1 + 2, \ldots, d_2 - 1)\ldots(d_m, d_{m-1} + 1, d_{m-1} + 2, \ldots, d_m - 1)$. If Q denotes the permutation matrix defined by π, one obtains QPQ^{-1} in the desired form. □

We say that any permutation matrix of this structure is in *cycle normal form*. The cycle normal form of P is denoted by $C(P)$. Up to permutation-similarity, we can always assume that the permutation matrix P of a given matrix with XOR count d is in cycle normal form, as stated in the following corollary.

Corollary 4.2.

$$P\prod_{k=1}^{d}(I+E_{i_k,j_k}) \sim_\pi C(P)\prod_{k=1}^{d}(I+E_{\sigma^{-1}(i_k),\sigma^{-1}(j_k)})$$

for some permutation $\sigma \in S_m$. ◊

4.3 Efficient Multiplication in Finite Fields

In this section, we first present some theoretic results towards understanding the structure of matrices $T_{\alpha,B}$ representing (left-)multiplication by some finite field element $\alpha \in \mathbb{F}_{2^m}^*$. The parameter B indicates a basis of $\mathbb{F}_{2^m}$ considered as an m-dimensional vector space over $\mathbb{F}_2$. The XOR count of $T_{\alpha,B}$ is indeed depending on the choice of the basis B. As described in Corollary 4.2, we can assume a certain normal form for matrices with an XOR count of d.

Not every (invertible) matrix is a representation of a field multiplication. For example, an obvious condition for that is that the multiplicative order of the matrix divides $2^m - 1$. In order to understand exactly which matrices indeed represent multiplication with some field element α, Theorem 4.1 below gives a characterization that allows to efficiently decide when a given matrix corresponds to multiplication by a field element. The crucial part is the minimal polynomial of α. It is a property of the linear mapping

$$f_\alpha : \mathbb{F}_{2^m} \to \mathbb{F}_{2^m}, \beta \mapsto \alpha\beta$$

and is invariant under changing the specific representation of f_α to $\beta \mapsto T_{\alpha,B}\beta$.

Theorem 4.1. *Let $A \in \mathrm{Mat}_m(\mathbb{F}_2) \setminus \{\mathbf{0}_m\}$. Then $A = T_{\alpha,B}$ for some element $\alpha \in \mathbb{F}_{2^m}^*$ with respect to some basis B if and only if m_A is irreducible.* ◊

Proof. As described in [War94], the ring generated by some matrix A defines a field of order 2^m if and only if the characteristic polynomial χ_A is irreducible. This is the case since $\chi_A(A) = 0$ and thus A is the root of an irreducible polynomial of degree m. One can see that $\mathbb{F}_2(A) = \{\sum_{i=0}^{m-1} \alpha_i A^i \mid \alpha_i \in \mathbb{F}_2\}$ since it must contain all sums of powers of A. However, for $\mathbb{F}_2(A)$ being a field it is not necessary that A has an irreducible characteristic polynomial. It can be possible that A generates a subfield $\mathbb{F}_{2^{m'}}$ of $\mathbb{F}_{2^m}$. As we show now, this is the case if and only if the minimal polynomial of α is irreducible and has degree m'.

If m_A is not irreducible, $\mathbb{F}_2(A)$ is not a field and thus A cannot represent a field multiplication. Let now m_A be irreducible. The characteristic polynomial χ_A is necessarily a power of m_A, since both of these polynomials share the same irreducible factors. So, $\chi_A = (m_A)^d$ for some positive integer d. Both d and $\deg(m_A)$ divide m. Because of the irreducibility of m_A, the rational canonical form of A consists of d blocks of C_{m_A}. Thus, we obtain the similarity

$$A \sim \bigoplus_{k=1}^{d} C_{m_A}.$$

Since $\chi_{C_{m_A}} = m_A$, the matrix A defines a multiplication with some element in a subfield of $\mathbb{F}_{2^m}$. □

Note that, any field element α is, up to its conjugates $\alpha, \alpha^2, \alpha^{2^2}, \dots, \alpha^{2^{m-1}}$, uniquely identified by its minimal polynomial. For every field element α, the minimal polynomial m_α is exactly the minimal polynomial m_A of a matrix A representing multiplication with α. Furthermore, two matrices $A, A' \in \mathrm{Mat}_m(\mathbb{F}_2)$ with the same irreducible minimal polynomial are similar. Thus, given a matrix A, identifying the element α such that $A = T_{\alpha,B}$ is equivalent to computing the (irreducible) minimal polynomial of A.

The main question is which field elements can be implemented with a minimal number of XOR operations, or in particular, what is the minimal XOR count for a given (non-trivial) field element $\alpha \in \mathbb{F}_{2^m}^*$. Trivially, multiplication with $\alpha = 1$ can be implemented with zero additions since $T_{1,B} = I_m$ for all bases B. On the other hand, if the XOR count is 0, the element is equal to 1. In the first place, we thus aim for an XOR count of 1 whenever possible. By a simple observation, this optimal result can be realized if the minimal polynomial of α is a trinomial of degree m.

Example 4.1. Let the field with 2^m elements be represented as $\mathbb{F}_{2^m} = \mathbb{F}_2[x]/(q)$ for an irreducible q of degree m. For the (left-)multiplication with x in the canonical basis $B = \{1, x, x^2, \dots, x^{m-1}\}$, it is $T_{x,B} = C_q$. Thus, $\mathrm{wt}_\oplus(T_{x,B}) = \mathrm{hw}(q) - 2$ and the XOR count of $T_{x,B}$ equals 1 if q is a trinomial. $\lozenge$

Since our approach is about finding any (non-trivial) element $\alpha \in \mathbb{F}_{2^m}^*$ such that multiplication with α can be implemented with minimal additions, this fact implies that we cannot hope to improve upon the implementation costs if there exists an irreducible trinomial of degree m. However, for several m, including the interesting case where m is a multiple of 8, there does not exist such a trinomial [Swa62]. The question is what happens in these cases. As one of our main results, we show that the condition on the minimal polynomial is not only sufficient but also necessary.

4.3.1 Characterizing Elements with Optimal XOR Count

In this section, we prove the converse of the fact described in Example 4.1, namely the necessary condition on the minimal (resp. characteristic) polynomial of α resulting in an XOR count of 1.

Theorem 4.2. *Let $\alpha \in \mathbb{F}_{2^m}$. Then there exists a matrix A with $\mathrm{wt}_\oplus(A) = 1$ such that $A = T_{\alpha,B}$ for some basis B if and only if m_α is a trinomial of degree m.* $\lozenge$

Proof. Let $T_{\alpha,B}$ represent multiplication by some element $\alpha \in \mathbb{F}_{2^m}$ with respect to the basis $B = \{b_1, \dots, b_m\}$ and let further $\mathrm{wt}_\oplus(T_{\alpha,B}) = 1$. We show that the characteristic polynomial $\chi_{T_{\alpha,B}}$ is a trinomial and coincides with m_α. Since the XOR count is 1, we can assume without loss of generality that $T_{\alpha,B} = P + E_{i,j}$ such that $P = \bigoplus_{k=1}^{l} C_{x^{m_k}+1}$ is in cycle normal form. We first show that $l = 1$. Suppose $l > 1$, then, depending on $E_{i,j}$, the matrix $T_{\alpha,B}$ is either in upper or lower triangular form consisting of at least two diagonal blocks. Since one of them must be of the form $C_{x^{m'}+1}$, the polynomial $x^{m'} + 1$ must divide the characteristic polynomial $\chi_{T_{\alpha,B}}$. Since further $(x+1) \mid (x^{m'}+1)$, the minimal polynomial of α is necessarily a multiple of $x + 1$. This is a contradiction since $\alpha \neq 1$ and m_α must be irreducible. Hence, $T_{\alpha,B}$ is permutation-similar to $C_{x^m+1} + E_{i,j}$. It is further $i \neq j + 1 \mod m$ since otherwise $T_{\alpha,B}$ would be singular.

We now investigate how α operates on the basis elements $b_k \in B$. Considering the structure of $T_{\alpha,B}$, we obtain the following list of equations.

$$\begin{aligned} \alpha b_1 &= b_2 \\ &\vdots \\ \alpha b_{j-1} &= b_j \\ \alpha b_j &= b_{j+1} + b_i \\ \alpha b_{j+1} &= b_{j+2} \\ &\vdots \\ \alpha b_n &= b_1. \end{aligned}$$

By defining $\gamma := b_{j+1}$, one can express every basis element b_k as a power of α multiplied by γ. In particular,

$$b_{j+k \mod m} = \alpha^{k-1}\gamma \tag{4.1}$$

for $k \in \{1, \dots, m\}$. Combining this observation with the identity $\alpha b_j = b_{j+1} + b_i$, one obtains

$$\alpha^m \gamma = \gamma + \alpha^t \gamma \tag{4.2}$$

for some exponent $t \neq 0$. Since $\gamma \neq 0$, the field element α is a root of the trinomial $p = x^m + x^t + 1$. It is left to show that p is exactly the minimal polynomial of α. Suppose that $m_\alpha = x^{m'} + \sum_{k=0}^{m'-1} c_k x^k$ with constants $c_k \in \{0,1\}$ and $m' < m$. By multiplying $m_\alpha(\alpha)$ with γ, one obtains

$$\alpha^{m'}\gamma = \sum_{k=0}^{m'-1} c_k \alpha^k \gamma$$

and thus $b_{t_{m'}} = \sum_{k=0}^{m'-1} c_k b_{t_k}$ for some basis elements b_{t_k}. We are now able to express one basis element b_{t_k} as a sum of other elements from B which is contradictory to the linear independence of the basis. Hence, $\deg(m_\alpha) = m$ and thus $m_\alpha = p$ which finally proves the theorem. □

Note that the polynomial p is exactly the characteristic polynomial of $T_{\alpha,B}$ since it must be a monic multiple of m_α having degree m. An alternative way of proving that the characteristic polynomial of a matrix $C_{x^m+1} + E_{i,j}$ is a trinomial is given in the following lemma which holds in general, even if T does not represent a multiplication with a field element. This lemma will be helpful later on.

Lemma 4.3. *For $T = C_{x^m+1} + E_{i,j}$ with $\mathrm{hw}(T) = m+1$, the characteristic polynomial χ_T of T is a trinomial of degree m.* ◊

Proof. It is to compute $\chi_T = \det(\lambda I_m - T) = \det(\lambda I_m + C_{x^m+1} + E_{i,j})$. If $j = m$, then $T = C_{x^m+x^{i-1}+1}$ and $\chi_T = \lambda^m + \lambda^{i-1} + 1$ is a trinomial of degree m. Thus, without loss of generality one can assume $j < m$. To compute the determinant we use Laplace's formula by expanding along the m-th column. One obtains

$$\chi_T = \det\left(\begin{pmatrix} 1 & \lambda & & & \\ & 1 & \lambda & & \\ & & \ddots & \ddots & \\ & & & 1 & \lambda \\ & & & & 1 \end{pmatrix} + E_{i-1,j}\right) + \lambda \det\left(\begin{pmatrix} \lambda & & & & \\ 1 & \lambda & & & \\ & \ddots & \ddots & & \\ & & 1 & \lambda & \\ & & & 1 & \lambda \end{pmatrix} + E_{i,j}\right),$$

where $E_{0,j} := \mathbf{0}$ and $E_{m,j} := \mathbf{0}$. Both of these remaining matrices are of dimension $(m-1)\times(m-1)$. We now distinguish three cases:

(i) $i < j$: The additional 1 lies in the upper triangle of T. Now, χ_T reduces to $\chi_T = 1 + \lambda\det(\lambda I_{m-1} + C_{x^{m-1}} + E_{i,j}))$. In order to compute the remaining determinant, we keep on expanding along the last column for $m-1-j$ times until the additional 1 is located in the rightmost column. We now obtain the determinant of a companion matrix. Thus,

$$\begin{aligned}\chi_T &= 1 + \lambda^{m-j}\det(\lambda I_j + C_{x^j+x^{i-1}}) \\ &= 1 + \lambda^{m-j}(\lambda^j + \lambda^{i-1}) = \lambda^m + \lambda^{m-j+i-1} + 1.\end{aligned}$$

(ii) $i = j$: In this case, the additional 1 lies on the main diagonal of T and

$$\chi_T = 1 + \lambda(\lambda^{m-2}(\lambda+1)) = \lambda^m + \lambda^{m-1} + 1.$$

(iii) $i > j$: The additional 1 lies in the lower triangle of T. Because of the structure of T, it is further $i > (j+1)$. Defining the $m' \times m'$ matrix $S^{\lambda}_{m'}$ as

$$S^{\lambda}_{m'} := \begin{pmatrix} 1 & \lambda & & & \\ & 1 & \lambda & & \\ & & \ddots & \ddots & \\ & & & 1 & \lambda \\ & & & & 1 \end{pmatrix},$$

the characteristic polynomial of T reduces to $\chi_T = \det(S^{\lambda}_{m-1} + E_{i-1,j}) + \lambda^m$. We expand along the last row of $S^{\lambda}_{m-1} + E_{i-1,j}$ for $m-i$ times and get $\chi_T = \det(S^{\lambda}_{i-1} + E_{i-1,j}) + \lambda^m$. Now, the additional 1 lies in the last row of the remaining $(i-1)\times(i-1)$-dimensional matrix. The goal is now to shift this 1 to the first column. This is done by expanding $j-1$ times along the first column. We now obtain $\chi_T = \det(S^{\lambda}_{i-j} + E_{i-j,1}) + \lambda^m$ and the additional 1 is in the lower left corner of the matrix. As the last step, we expand along the first column for one more time and finally get

$$\begin{aligned}\chi_T &= \lambda^m + \det(S^{\lambda}_{i-j} + E_{i-j,1}) = \lambda^m + \det(\lambda I_{i-j-1} + C_{x^{i-j-1}}) + 1 \\ &= \lambda^m + \lambda^{i-j-1} + 1.\end{aligned}$$

□

As a simple corollary one obtains that any $\alpha \in \mathbb{F}_{2^m}^*$ with an XOR count of 1 cannot be contained in a proper subfield.

Corollary 4.3. *Let $\alpha \in \mathbb{F}_{2^m}^* \setminus \{1\}$ and let further $\deg(m_\alpha) < m$, indicating that α lies in a proper subfield of $\mathbb{F}_{2^m}$. Then, any matrix $T_{\alpha,B}$ representing multiplication by a field element α with respect to some basis B has $\mathrm{wt}_\oplus(T_{\alpha,B}) > 1$.* ◊

This result implies that building MDS layers using a block interleaving construction [Alb+14], also called subfield construction in [Kho+14], almost always results in suboptimal implementation costs. Note that specific instances of this construction are also implicitly used in the AES, LS-Designs [Gro+15] and the hash function Whirlwind [Bar+10].

Now let α be an element with XOR count 1. From Corollary 4.1 we know that α^{-1} has the same XOR count. Next, we show that there do not exist any further elements with an XOR count equal to 1.

Theorem 4.3. *For any given basis B of $\mathbb{F}_{2^m}$, there exist at most two field elements α and α^{-1} with $\mathrm{wt}_\oplus(T_{\alpha,B}) = \mathrm{wt}_\oplus(T_{\alpha^{-1},B}) = 1$.* ◊

Proof. Let $\alpha \in \mathbb{F}_{2^m}^*$ with $\mathrm{wt}_\oplus(T_{\alpha,B}) = 1$ for the basis $B = \{b_1, \ldots, b_m\}$. We show that for any $\beta \in \mathbb{F}_{2^m}$ with $\mathrm{wt}_\oplus(T_{\beta,B}) = 1$ it holds that $\beta = \alpha^{\pm 1}$.

Since without loss of generality $T_{\alpha,B}$ can be assumed to be of the form $C_{x^m+1} + E_{i,j}$, we know that Eq. (4.1) and Eq. (4.2) hold. We further know that $T_{\beta,B}$ is of the form $P + E_{i',j'}$ and thus there exist $l, l' \in \{1, \ldots, m\}$ with $l \neq l'$ and $\beta b_{j+l \bmod m} = b_{j+l' \bmod m}$. Using Eq. (4.1), we can write $\beta = \alpha^{l'-l} =: \alpha^s$ where $s \in \{-(m-1), \ldots, m-1\}$. We directly see that $s \neq 0$. It remains to show that $-1 \leq s \leq 1$.

Assume $s \geq 2$. We use equations Eqs. (4.1) and (4.2) to obtain

$$\beta b_{j+(m-s+1) \bmod m} = \alpha^m \gamma = \gamma + \alpha^t \gamma = b_{j+1 \bmod m} + b_{j+t+1 \bmod m}.$$

Since $0 < t < m$, it holds that $b_{j+1 \bmod m} \neq b_{j+t+1 \bmod m}$ and thus the according column contains an additional 1. For the next column, we have

$$\begin{aligned} \beta b_{j+(m-s+2) \bmod m} &= \alpha^{m+1}\gamma = \alpha\gamma + \alpha^{t+1}\gamma \\ &= \begin{cases} b_{j+2 \bmod m} + b_{j+t+2 \bmod m}, & \text{for } t < m-1 \\ b_{j+2 \bmod m} + b_{j+1 \bmod m} + b_{j \bmod m}, & \text{for } t = m-1 \end{cases} \end{aligned}$$

Hence, this column also contains at least one additional 1 which is contradictory to the XOR count of 1.

For $-s \geq 2$ we can construct the same contradiction by considering β^{-1}. □

We now understand the structure of field elements α that can be implemented with a single addition. One might think that also for the other cases, the weight of the minimal polynomial of α strictly lower-bounds XOR count as $\mathrm{hw}(m_\alpha) - 2$. As we will see next, this is not the case.

Table 4.1: Minimal XOR counts for all elements in $\mathbb{F}_{2^4}^*$.

minimal polynomial m_α	**min** $\mathrm{wt}_\oplus(\alpha)$	**matrix**
$x+1$	0	I
x^2+x+1	2	$C_{m_\alpha} \oplus C_{m_\alpha}$
x^4+x+1	1	C_{m_α}
x^4+x^3+1	1	C_{m_α}
$x^4+x^3+x^2+x+1$	2	$C_{x^4+1}+E_{2,2}+E_{3,4}$

Table 4.2: Minimal XOR counts for all elements in $\mathbb{F}_{2^5}^*$.

minimal polynomial m_α	**min** $\mathrm{wt}_\oplus(\alpha)$	**matrix**
$x+1$	0	I
x^5+x^2+1	1	C_{m_α}
x^5+x^3+1	1	C_{m_α}
$x^5+x^3+x^2+x+1$	2	$C_{x^5+1}+E_{2,4}+E_{4,2}$
$x^5+x^4+x^2+x+1$	2	$C_{x^5+1}+E_{2,2}+E_{3,5}$
$x^5+x^4+x^3+x+1$	2	$C_{x^5+1}+E_{2,3}+E_{3,1}+E_{3,3}$
$x^5+x^4+x^3+x^2+1$	2	$C_{x^5+1}+E_{2,2}+E_{3,4}$

4.3.2 Experimental Search for Optimal XOR Counts

Surprisingly, we often can improve the XOR count, compared to using the companion matrix for multiplication, if the weight of the minimal polynomial is greater than 3. For instance, if m_α is an irreducible pentanomial, that is of weight 5, of degree m there often exists a basis B such that $\mathrm{wt}_\oplus(T_{\alpha,B}) = 2$. Indeed, for all $m \leq 2048$ for which no irreducible trinomial of degree m exists, we found some element $\alpha \in \mathbb{F}_{2^m}^*$ with an XOR count of 2 for some basis B. For every such dimension, we present an example of such a matrix in Table B.1 in the appendix. Thus, for all practically relevant fields, we are able to identify an element such that multiplication can be implemented with one or two XOR operations. By Theorem 4.2, these results are proven to be optimal.

Moreover, as fields of small size are most interesting for SP-networks, we investigated those in full detail. For the fields $\mathbb{F}_{2^4}$, $\mathbb{F}_{2^5}$, $\mathbb{F}_{2^6}$, $\mathbb{F}_{2^7}$ and $\mathbb{F}_{2^8}$ we present the optimal XOR count for each non-trivial element α in Table 4.1, Table 4.2, Table 4.3, Table 4.4 and Table 4.5, respectively. The main observation is that each element that is not contained in a proper subfield can be implemented with at most 3 additions. Furthermore, whenever an XOR count of 2 is possible, the minimal polynomial of α is a pentanomial in all those cases. However, a more thorough characterization of elements with non-optimal XOR count is left as an open problem (see Section 4.6 for more details).

Those results are based on a search. Since we are only interested in matrices up to similarity (due to the change of basis), we just need to consider all matrices in the normal form described in Corollary 4.2. This will exhaust all possibilities of similarity classes for a given XOR count d. In particular, the search space is reduced from $m!(m(m-1))^d$ to only $p(m)(m(m-1))^d$ where

Table 4.3: Minimal XOR counts for all elements in $\mathbb{F}_{2^6}^*$.

minimal polynomial m_α	**min** $\mathrm{wt}_\oplus(\alpha)$	**matrix**
$x+1$	0	I
x^2+x+1	3	$C_{m_\alpha} \oplus C_{m_\alpha} \oplus C_{m_\alpha}$
x^3+x+1	2	$C_{m_\alpha} \oplus C_{m_\alpha}$
x^3+x^2+1	2	$C_{m_\alpha} \oplus C_{m_\alpha}$
x^6+x+1	1	C_{m_α}
x^6+x^3+1	1	C_{m_α}
$x^6+x^4+x^2+x+1$	2	$(C_{x^4+1} \oplus C_{x^2+1})(I+E_{1,5}+E_{5,4})$
$x^6+x^4+x^3+x+1$	2	$C_{x^6+1}+E_{2,3}+E_{4,6}$
x^6+x^5+1	1	C_{m_α}
$x^6+x^5+x^2+x+1$	2	$C_{x^6+1}+E_{2,2}+E_{3,6}$
$x^6+x^5+x^3+x^2+1$	2	$C_{x^6+1}+E_{2,2}+E_{3,5}$
$x^6+x^5+x^4+x+1$	2	$C_{x^6+1}+E_{2,3}+E_{3,1}+E_{3,3}$
$x^6+x^5+x^4+x^2+1$	2	$(C_{x^4+1} \oplus C_{x^2+1})(I+E_{1,5}+E_{6,1}+E_{6,5})$

Table 4.4: Minimal XOR counts for all elements in $\mathbb{F}_{2^7}^*$.

minimal polynomial m_α	**min** $\mathrm{wt}_\oplus(\alpha)$	**matrix**
$x+1$	0	I
x^7+x+1	1	C_{m_α}
x^7+x^3+1	1	C_{m_α}
$x^7+x^3+x^2+x+1$	2	$C_{x^7+1}+E_{2,6}+E_{4,2}$
x^7+x^4+1	1	C_{m_α}
$x^7+x^4+x^3+x^2+1$	2	$(C_{x^4+1} \oplus C_{x^3+1})(I+E_{1,5}+E_{5,3})$
$x^7+x^5+x^2+x+1$	2	$(C_{x^5+1} \oplus C_{x^2+1})(I+E_{1,6}+E_{6,5})$
$x^7+x^5+x^3+x+1$	2	$C_{x^7+1}+E_{2,3}+E_{4,7}$
$x^7+x^5+x^4+x^3+1$	2	$(C_{x^4+1} \oplus C_{x^3+1})(I+E_{1,5}+E_{7,2})$
$x^7+x^5+x^4+x^3+x^2+x+1$	3	$C_{x^7+1}+E_{2,3}+E_{4,6}+E_{4,7}$
x^7+x^6+1	1	C_{m_α}
$x^7+x^6+x^3+x+1$	2	$(C_{x^6+1} \oplus C_{x^1+1})(I+E_{1,7}+E_{7,4})$
$x^7+x^6+x^4+x+1$	2	$(C_{x^6+1} \oplus C_{x^1+1})(I+E_{1,7}+E_{7,3})$
$x^7+x^6+x^4+x^2+1$	2	$C_{x^7+1}+E_{2,4}+E_{4,1}+E_{4,4}$
$x^7+x^6+x^5+x^2+1$	2	$(C_{x^5+1} \oplus C_{x^2+1})(I+E_{1,6}+E_{7,1}+E_{7,6})$
$x^7+x^6+x^5+x^3+x^2+x+1$	3	$C_{x^7+1}+E_{2,2}+E_{2,3}+E_{4,7}$
$x^7+x^6+x^5+x^4+1$	2	$C_{x^7+1}+E_{2,2}+E_{3,4}$
$x^7+x^6+x^5+x^4+x^2+x+1$	3	$C_{x^7+1}+E_{2,2}+E_{3,4}+E_{3,7}$
$x^7+x^6+x^5+x^4+x^3+x^2+1$	3	$C_{x^7+1}+E_{2,2}+E_{2,3}+E_{4,6}$

Table 4.5: Minimal XOR counts for all elements in $\mathbb{F}_{2^8}^*$.

minimal polynomial m_α	**min** $\mathrm{wt}_\oplus(\alpha)$	**matrix**
$x+1$	0	I
x^2+x+1	4	$\bigoplus_{k=1}^{4} C_{m_\alpha}$
x^4+x+1	2	$C_{m_\alpha} \oplus C_{m_\alpha}$
x^4+x^3+1	2	$C_{m_\alpha} \oplus C_{m_\alpha}$
$x^4+x^3+x^2+x+1$	4	$\bigoplus_{k=1}^{2} (C_{x^4+1}+E_{2,2}+E_{3,4})$
$x^8+x^4+x^3+x+1$	2	$C_{x^8+1}+E_{2,6}+E_{4,2}$
$x^8+x^4+x^3+x^2+1$	3	C_{m_α}
$x^8+x^5+x^3+x+1$	2	$(C_{x^5+1} \oplus C_{x^3+1})(I+E_{1,6}+E_{6,5})$
$x^8+x^5+x^3+x^2+1$	2	$C_{x^8+1}+E_{2,6}+E_{5,2}$
$x^8+x^5+x^4+x^3+1$	2	$(C_{x^5+1} \oplus C_{x^3+1})(I+E_{1,6}+E_{6,2})$
$x^8+x^5+x^4+x^3+x^2+x+1$	3	$C_{x^8+1}+E_{2,5}+E_{2,7}+E_{4,2}$
$x^8+x^6+x^3+x^2+1$	2	$(C_{x^6+1} \oplus C_{x^2+1})(I+E_{1,7}+E_{8,5})$
$x^8+x^6+x^4+x^3+x^2+x+1$	3	$C_{x^8+1}+E_{2,3}+E_{4,7}+E_{4,8}$
$x^8+x^6+x^5+x+1$	2	$C_{x^8+1}+E_{2,4}+E_{4,2}$
$x^8+x^6+x^5+x^2+1$	2	$(C_{x^6+1} \oplus C_{x^2+1})(I+E_{1,7}+E_{7,2})$
$x^8+x^6+x^5+x^3+1$	2	$C_{x^8+1}+E_{2,3}+E_{4,6}$
$x^8+x^6+x^5+x^4+1$	3	C_{m_α}
$x^8+x^6+x^5+x^4+x^2+x+1$	3	$C_{x^8+1}+E_{2,3}+E_{2,4}+E_{5,8}$
$x^8+x^6+x^5+x^4+x^3+x+1$	3	$C_{x^8+1}+E_{2,3}+E_{2,5}+E_{6,8}$
$x^8+x^7+x^2+x+1$	2	$C_{x^8+1}+E_{2,2}+E_{3,8}$
$x^8+x^7+x^3+x+1$	2	$(C_{x^7+1} \oplus C_{x+1})(I+E_{1,8}+E_{8,5})$
$x^8+x^7+x^3+x^2+1$	2	$C_{x^8+1}+E_{2,2}+E_{3,7}$
$x^8+x^7+x^4+x^3+x^2+x+1$	3	$C_{x^8+1}+E_{2,2}+E_{3,6}+E_{3,8}$
$x^8+x^7+x^5+x+1$	2	$(C_{x^7+1} \oplus C_{x+1})(I+E_{1,8}+E_{8,3})$
$x^8+x^7+x^5+x^3+1$	2	$(C_{x^5+1} \oplus C_{x^3+1})(I+E_{1,6}+E_{8,1}+E_{8,6})$
$x^8+x^7+x^5+x^4+1$	2	$C_{x^8+1}+E_{2,2}+E_{3,5}$
$x^8+x^7+x^5+x^4+x^3+x^2+1$	3	$C_{x^8+1}+E_{2,2}+E_{3,5}+E_{3,7}$
$x^8+x^7+x^6+x+1$	2	$C_{x^8+1}+E_{2,3}+E_{3,1}+E_{3,3}$
$x^8+x^7+x^6+x^3+x^2+x+1$	3	$C_{x^8+1}+E_{2,2}+E_{2,3}+E_{4,8}$
$x^8+x^7+x^6+x^4+x^2+x+1$	3	$(C_{x^6+1} \oplus C_{x^2+1})(I+E_{1,7}+E_{7,3}+E_{7,8})$
$x^8+x^7+x^6+x^4+x^3+x^2+1$	3	$C_{x^8+1}+E_{2,2}+E_{2,3}+E_{4,7}$
$x^8+x^7+x^6+x^5+x^2+x+1$	3	$C_{x^8+1}+E_{2,2}+E_{3,4}+E_{3,8}$
$x^8+x^7+x^6+x^5+x^4+x+1$	3	$C_{x^8+1}+E_{2,3}+E_{3,1}+E_{3,3}+E_{8,3}$
$x^8+x^7+x^6+x^5+x^4+x^2+1$	3	$C_{x^8+1}+E_{2,2}+E_{2,5}+E_{6,7}$
$x^8+x^7+x^6+x^5+x^4+x^3+1$	3	$C_{x^8+1}+E_{2,2}+E_{2,3}+E_{4,6}$

$p(m)$ denotes the number of partitions of m, which is exactly the number of possible cycle normal forms of dimension m. This allows us to exhaustively search over all similarity classes up to $d = 3$ XOR operations for the fields of small size. The key-point here is that, instead of searching for an optimal basis for a given field element, we generated all matrices with small XOR count and used Theorem 4.1 in order to check which field element (if any) the given matrix corresponds to.

In order to identify a single lightweight element for larger field sizes, we identified conditions in which cases the characteristic polynomial of a matrix with XOR count 2 has weight 5, see Theorem 4.4 below. During the search, one only has to check for irreducibility. This allows to compute the results presented in Table B.1 extremely fast, that is, within a couple of minutes on a standard PC.

Theorem 4.4. *Let $T = C_{x^m+1} + E_{i_1,j_1} + E_{i_2,j_2}$ such that the following relations hold:*

$$i_1 < j_1 \neq m, \quad i_2 > j_2 + 1, \quad i_1 \leq j_2, \quad i_2 \leq j_1, \quad j_1 - (i_1 - 1) \neq m, \quad m - (j_1 - i_1) \neq i_2 - j_2$$

The characteristic polynomial of T is a pentanomial of degree m. In particular

$$\chi_T = \lambda^m + \lambda^{m+i_1-j_1+i_2-j_2-2} + \lambda^{m+i_1-j_1-1} + \lambda^{i_2-j_2-1} + 1.$$

◊

Proof. The first two conditions ensure that T has exactly one additional non-zero entry in the upper and one in the lower triangle (not on the main diagonal). Since $j_1, j_2, i_2 \neq m$, we can expand along the last column and obtain

$$\chi_T = \det(S^{\lambda}_{m-1} + E_{i_1-1,j_1} + E_{i_2-1,j_2}) + \lambda \det(\lambda I_{m-1} + C_{x^{m-1}} + E_{i_1,j_1} + E_{i_2,j_2}).$$

For simplicity, we define $A := S^{\lambda}_{m-1} + E_{i_1-1,j_1} + E_{i_2-2,j_2}$ and $B := \lambda I_{m-1} + C_{x^{m-1}} + E_{i_1,j_1} + E_{i_2,j_2}$. In order to compute the latter part, we "push" the additional non-zero entry from the upper triangle to the top-right corner by first expanding $m - 1 - j_1$ times along the last column and then expanding $i_1 - 1$ times along the first row. The condition $i_2 \leq j_1$ ensures that E_{i_2,j_2} will not be eliminated from expanding along the last column and the condition $i_1 \leq j_2$ ensures that E_{i_2,j_2} will not be eliminated from expanding along the first row. Using Lemma 4.3, one obtains

$$\begin{aligned}\lambda \det(B) &= \lambda\lambda^{m-1-j_1}\lambda^{i_1-1} \det(\lambda I_{j_1-i_1+1} + C_{x^{j_1-i_1+1}+1} + E_{i_2-i_1+1,j_2-i_1+1}) \\ &= \lambda^{m-1-j_1+i_1}(\lambda^{j_1-i_1+1} + \lambda^{i_2-i_1+1-j_2+i_1-1-1} + 1) \\ &= \lambda^m + \lambda^{m+i_1-j_1+i_2-j_2-2} + \lambda^{m+i_1-j_1-1}.\end{aligned}$$

For $\det(A)$, we proceed similar to case (iii) in Lemma 4.3. We first expand $j_2 - 1$ times along the first column in order to get the additional non-zero value from the lower triangle to the leftmost column. Because of the condition $i_1 \leq j_2$, this eliminates E_{i_1-1,j_1}. Now, one can expand $m - j_2 - (i_2 - j_2)$ times along the last row, until the remaining additional non-zero entry lies in the lower left corner of the remaining matrix. We finally expand along the first column one more time and obtain

$$\det(A) = \det(S^{\lambda}_{m-j_2} + E_{i_2-j_2,1}) = \det(S^{\lambda}_{i_2-j_2} + E_{i_2-j_2,1}) = \lambda^{i_2-j_2-1} + 1.$$

The last two assumptions make sure that all of the five coefficients of $\det(A) + \lambda \det(B)$ are distinct such that χ_T is indeed a pentanomial. □

4.4 Constructing Lightweight MDS Matrices

Our goal is now to construct lightweight MDS matrices. We use the results obtained in the previous sections and restrict our search to circulant matrices and entries with low XOR count. This simplifies checking the MDS property and computing an upper bound of the XOR count of the whole matrix. The complexity of our algorithm enables us to easily search for MDS matrices up to dimension 8. Our construction is generic and works for all finite fields $\mathbb{F}_{2^m}$ with $m > b$ for a given bound b.

More precisely, we construct circulant matrices with entries of the form $\alpha^{\pm i}$ where α is an element in $\mathbb{F}_{2^m}$. Choosing entries of this form enables us to easily upper-bound the XOR count of the elements since

$$\mathrm{wt}_{\oplus}(x^{\pm k}) \leq k\,\mathrm{wt}_{\oplus}(x).$$

This can be easily seen by using Corollary 4.1 and the fact that α^k can be implemented by k times implementing α. We want to keep the size of the finite field, over which the matrix is defined, generic. Thus, we choose the matrix entries from a subgroup of the *field of fractions* of the polynomial ring $\mathbb{F}_2[x]$, denoted $\mathrm{Quot}(\mathbb{F}_2[x])$. That is, every element is of the form

$$\frac{x^s + a_{s-1}x^{s-1} + \cdots + a_1x + a_0}{x^{s'} + b_{s'-1}x^{s'-1} + \cdots + b_1x + b_0}.$$

More precisely, and as mentioned above, we restrict our search to elements from $\langle x \rangle$ which is the multiplicative subgroup of $\mathrm{Quot}(\mathbb{F}_2[x])$ generated by x. Our search works by constructing MDS conditions for a $t \times t$ matrix M with entries in $\langle x \rangle$. This approach later allows us to substitute the indeterminate x by any $\alpha \in \mathbb{F}_{2^m}$ that fulfills all of the conditions given below. In this context, we let $M(\alpha) \in \mathrm{Mat}_t(\mathbb{F}_{2^m})$ denote the matrix obtained by substituting x with $\alpha \in \mathbb{F}_{2^m}$.

We define the *weight* of some circulant matrix with entries in $\langle x \rangle$ as the sum of the absolute values of the exponents in its first row, that is, the number of times α has to be applied *per row*. Then, for a given dimension, we are interested in finding the lightest matrix M which can be made MDS for as many finite fields as possible. Note that the higher priority here was to find a lightweight matrix. Thus, there might exist matrices which can be made MDS for even more fields, but with a probably higher cost.

MDS conditions

Note that a matrix is MDS, if and only if all its square submatrices are invertible [MS77, page 321, Theorem 8]. Thus, given a matrix $M \in \mathrm{Mat}_t(\mathrm{Quot}(\mathbb{F}_2[x]))$, we compute the determinants of all square submatrices (called *minors*) of M in order to check the MDS property. This way one obtains a list of conditions (polynomials in $\mathbb{F}_2$) for a matrix to be MDS. Since the determinant of a matrix with elements from a field is an element of the field itself, all of these determinants can be represented as the fraction of two polynomials. Thus, M is MDS if and only if the numerator of all minors is non-zero. One can decompose the numerators into their irreducible factors and collect all of them in a set S. This set now defines the MDS conditions. In particular, $M(\alpha)$ is MDS if and only if α is not a root of any of these irreducible polynomials in S, that is, iff $m_\alpha \notin S$. This trivially holds for $m > \max_{p \in S}\{\deg(p)\}$ and any $\alpha \in \mathbb{F}_{2^m}$ which is not contained in a proper subfield. In

Listing 4.1: Sage code for computing the set S.

```
P.<x> = GF(2)[]
K = FractionField(P)

def mds_equations(M):
    S = [P(x)]
    for i in range(len(M.rows())+1)[1:]:
        L = M.minors(i)
        for l in L:
            if (l != 0):
                F = list(l.numerator().factor())
                for f in F:
                    S.append(f[0])
            else:
                return
    return list(set(S))
```

general, if α is not contained in a proper subfield, the necessary and sufficient condition for the existence of an MDS matrix $M(\alpha)$ is that not all irreducible polynomials of degree m are contained in S. We note that there exists a value b which lower bounds the field size for which M can always be made MDS. That is, for all $b' > b$, there exists an irreducible polynomial of degree b' which is not in S.

4.4.1 Generic Lightweight MDS Matrices

We now present some results obtained by the approach described above. Given the restrictions, these matrices achieve the smallest weight, i. e. the smallest sum of (absolute) exponents of x. Later, we will use these generic matrices to build concrete instantiations of $t \times t$ MDS matrices $M(\alpha)$ for $t \in \{2, 3, \dots, 8\}$ over a finite field $\mathbb{F}_{2^m}$ with $m > b$. We note that the given results are not necessarily the only possible constructions with the smallest weight.

We also present the conditions for the matrix to be MDS, that is, the irreducible polynomials that must not be equal to m_α. However, since the number of conditions rapidly increases with the dimension of the matrix, we refrain from presenting a complete list for dimensions 6 to 8. Instead, we give the SageMath source code that was used to compute the set S of irreducible polynomials in Listing 4.1.

2×2 **and** 3×3 **matrices.**

The matrices

$$\mathrm{circ}(1, \alpha) = \begin{pmatrix} 1 & \alpha \\ \alpha & 1 \end{pmatrix}$$

and

$$\mathrm{circ}(1, 1, \alpha) = \begin{pmatrix} 1 & 1 & \alpha \\ \alpha & 1 & 1 \\ 1 & \alpha & 1 \end{pmatrix}$$

are MDS for all $\alpha \neq 0, 1$.

4×4 **matrices.**

For $m > 3$, there exists an $\alpha \in \mathbb{F}_{2^m}$ such that the matrix $\operatorname{circ}(1, 1, \alpha, \alpha^{-2})$ is MDS. More precisely, the matrix is MDS iff α is not a root of any of the following polynomials:

$$\begin{aligned}
x\\
x+1\\
x^2+x+1\\
x^3+x+1\\
x^3+x^2+1\\
x^4+x^3+x^2+x+1\\
x^5+x^2+1
\end{aligned}$$

5×5 **matrices.**

For $m > 3$, there exists an $\alpha \in \mathbb{F}_{2^m}$ such that the matrix $\operatorname{circ}(1, 1, \alpha, \alpha^{-2}, \alpha)$ is MDS. More precisely, the matrix is MDS iff α is not a root of any of the following polynomials:

$$\begin{aligned}
x\\
x+1\\
x^2+x+1\\
x^3+x+1\\
x^3+x^2+1\\
x^4+x+1\\
x^4+x^3+1
\end{aligned}$$

6×6 **matrices.**

For $m > 5$, there exists an $\alpha \in \mathbb{F}_{2^m}$ such that the matrix $\operatorname{circ}(1, \alpha, \alpha^{-1}, \alpha^{-2}, 1, \alpha^3)$ is MDS.

7×7 **matrices.**

For $m > 5$, there exists an $\alpha \in \mathbb{F}_{2^m}$ such that the matrix $\operatorname{circ}(1, 1, \alpha^{-2}, \alpha, \alpha^2, \alpha, \alpha^{-2})$ is MDS.

8×8 **matrices.**

For $m > 7$, there exists an $\alpha \in \mathbb{F}_{2^m}$ such that the matrix $\operatorname{circ}(1, 1, \alpha^{-1}, \alpha, \alpha^{-1}, \alpha^3, \alpha^4, \alpha^{-3})$ is MDS.

4.4.2 Instantiating Lightweight MDS Matrices

We now combine the efficient multiplication in finite fields from Section 4.3 with our construction of MDS matrices. That is, the presented generic MDS matrices are instantiated with elements α with low XOR count.

Table 4.6: Optimal instantiations of the generic MDS matrices for $2 \le t \le 8$. In each cell, the first entry describes the minimal polynomial of $\alpha \in \mathbb{F}_2^m$ and the second entry describes the overhead of the instantiated $t \times t$ matrix $M(\alpha)$. The trinomial $x^m + x^a + 1$ is denoted by (a) and the pentanomial $x^m + x^a + x^b + x^c + 1$ is denoted by (a, b, c).

t \ m	2	3	4	5	6	7	8	9	10	11	12	13
2	(1), 1	(1), 1	(1), 1	(2), 1	(1), 1	(1), 1	(6, 5, 1), 2	(1), 1	(3), 1	(2), 1	(3), 1	(10, 9, 1), 2
3	(1), 1	(1), 1	(1), 1	(2), 1	(1), 1	(1), 1	(6, 5, 1), 2	(1), 1	(3), 1	(2), 1	(3), 1	(10, 9, 1), 2
4	–	–	(1), 3	(3), 3	(1), 3	(1), 3	(6, 5, 1), 6	(1), 3	(3), 3	(2), 3	(3), 3	(10, 9, 1), 6
5	–	–	(3, 2, 1), 8	(2), 4	(1), 4	(1), 4	(6, 5, 1), 8	(1), 4	(3), 4	(2), 4	(3), 4	(10, 9, 1), 8
6	–	–	–	–	(1), 7	(1), 7	(6, 5, 1), 14	(1), 7	(3), 7	(2), 7	(3), 7	(10, 9, 1), 14
7	–	–	–	–	(1), 8	(1), 8	(6, 5, 1), 16	(1), 8	(3), 8	(2), 8	(3), 8	(10, 9, 1), 16
8	–	–	–	–	–	–	(6, 5, 2), 26	(8), 13	(3), 13	(2), 13	(3), 13	(10, 9, 1), 26

In a matrix multiplication, every element is computed as the sum over multiplications. The according XOR count was already discussed in [Kho+14] and [Sim+15]. For our matrices, the total number of XOR operations needed *per row* is upper bounded by

$$(t-1)m + w \cdot \mathrm{wt}_{\oplus}(\alpha).$$

Here, $(t-1)m$ XORs are the static part which comes from summing over the multiplication results and w is the weight as defined above. The *overhead* of $w \cdot \mathrm{wt}_{\oplus}(\alpha)$ XORs is needed for multiplying with the single elements. The static part cannot be changed by fast multiplication. Therefore, this overhead is the part that has to be minimized.

The cost per bit for the whole matrix is given by

$$\frac{t((t-1)m + w\,\mathrm{wt}_{\oplus}(\alpha))}{tm} = t - 1 + \frac{w\,\mathrm{wt}_{\oplus}(\alpha)}{m}.$$

One can notice that it decreases for larger field sizes.

For each of the matrices M described in Section 4.4.1, Table 4.6 presents choices for α such that $M(\alpha)$ is MDS. Note that concrete instantiations are only given up to the field size $m = 13$. The reason is that for larger m, all possible C_p with p as an irreducible degree-m polynomial of weight 3 are valid choices. If no such trinomial exists, one can choose $T_{\alpha,B}$ as in Table B.1.

Table 4.7 compares the results presented in this section to the best constructions known at the time of presenting the results in 2016. It turned out that our construction of the 4×4 MDS matrix in $\mathbb{F}_{2^4}$ is identical to the $\mathbb{F}_2$-linear matrix constructed in [LS16; LW16]. We stress that our construction led to the lightest MDS matrices in 2016, improving the results described in [LS16; Sim+15] for 8×8 MDS matrices in $\mathbb{F}_{2^4}$ and $\mathbb{F}_{2^8}$ respectively. This is also the case when considering an unrolled implementation of the serial implementations in [WWW13]. Unrolled variants of their implementations have an XOR count that is slightly larger than ours. Moreover, and more importantly, the circuit depth is considerably increased due to the optimization with respect to a serial implementation.

Note that our results in Table 4.7 are measured by the XOR count from Definition 4.1 while the results from [LS16; LW16; Sim+15] use the old XOR count definition. Additionally to these results, our understanding of how to choose an optimal basis can also be used to improve existing

Table 4.7: Comparison of our results with the (non-involutory) $\mathbb{F}_{2^m}$-linear MDS matrices from [Sim+15, Section 6.2],[LS16] and [LW16] by overhead. †: In these constructions, the XOR count is measured by counting the number of additional 1's in the corresponding matrix.

(t, m)	our construction	construction in [Sim+15]†	construction in [LS16]†	construction in [LW16]†
(4,4)	3	5	3	3
(4,8)	6	10	8	
(8,8)	26	40	30	

results in the old XOR count definition. For example, we can represent the 8×8 MDS matrix in $\mathbb{F}_{2^8}$ from [LS16] with 28 additional ones instead of 30 by a change of basis.

4.5 Generalizing the MDS Property

Here, following e. g. [WWW13], we consider a generalization to additive MDS codes in order to improve efficiency.

There are some dimensions for which no field element with an XOR count of 1 exists, for instance $m = 8$. However, especially this dimension is very important since lots of block cipher designs are byte-oriented. One would wish to have some element α with $\mathrm{wt}_\oplus(\alpha) = 1$. A way of solving this problem is to not restrict to field elements. Instead, α can be chosen to be some other matrix in the ring $R = \mathrm{Mat}_m(\mathbb{F}_2)$. Given a $t \times t$ matrix M with elements in $\mathrm{Quot}(\mathbb{F}_2[x])$, the substitution $M(\alpha)$ now consists of elements in a commutative ring with unity, which is the subring of R generated by α. In general, given a commutative ring with unity R, one can define the determinant $\det_R : \mathrm{Mat}_t(R) \to R$ in a similar way than for matrices over fields. As described in [Kna07, p. 212 - 215], any $A \in \mathrm{Mat}_t(R)$ is invertible if and only if $\det_R(A)$ is a unit in R. We now define the MDS property for matrices over a commutative ring.

Definition 4.2. Let R be a commutative ring with unity. A matrix $M \in \mathrm{Mat}_t(R)$ is *MDS* if and only if for every $1 \le t' \le t$, any $t' \times t'$ submatrix of M is invertible. ◊

For checking the MDS property in our case, we use a well-known fact about block matrices.

Theorem 4.5 (Theorem 1 in [Sil00]). *Let K be a field and let R be a commutative subring of $\mathrm{Mat}_m(K)$ for some integer m. For any matrix $M \in \mathrm{Mat}_d(R)$, it is*

$$\det(M) = \det(\det_R(M)),$$

where $\det(M)$ is the determinant of M considered as $M \in \mathrm{Mat}_{dm}(K)$. ◊

As an implication, $M(\alpha)$ is MDS if and only if $p(\alpha)$ is invertible for all $p \in S$, if and only if $\det(p(\alpha)) \neq 0$ for all $p \in S$.

2×2 and 3×3 matrices.

Given $M = \mathrm{circ}(1, x)$ (resp. $M = \mathrm{circ}(1, 1, x)$), one has to make sure that both x and $x + 1$ are invertible for M to be MDS. This is the case if x is substituted by the companion matrix C_{x^m+x+1} for $m \ge 2$. Thus, $M(C_{x^m+x+1})$ is MDS and each entry has an XOR count of 1.

4×4 matrices.

The MDS conditions are more complex than above. So, we only present some improvements for $m \in \{8, 13, 16\}$. The matrix $M = \mathrm{circ}(1, 1, \alpha, \alpha^{-2})$ is MDS for

$$\alpha \in \{C_{x^8+x^2+1}, C_{x^{13}+x+1}, C_{x^{16}+x+1}\}.$$

Note that a similar matrix for $m = 8$ was recently constructed in [LW16].

4.6 Conclusion and Further Research

We presented a study of optimal multiplication bases with respect to the XOR count. When applied to MDS matrices those lead to very efficient round-based implementations. We expect our results to be applicable in other domains as well.

Our investigations leave many possibilities for future research. While we have been able to characterize exactly which field elements can be implemented with one XOR operation only, the general case is still open. For small fields of dimension smaller or equal to eight, we were able to compute the optimal bases with the help of an exhaustive computer search. However, for larger dimensions, this approach turns quickly inefficient and more insight would be needed. As a first step, we conjectured the following statement in 2016.

Conjecture 4.1. *If* $\mathrm{wt}_{\oplus}(T_{\alpha,B}) = 2$*, then* m_α *is of weight smaller or equal to* 5. $\Diamond$

Note that the converse of the conjectured statement is (unlike the case of trinomials) wrong. As can be seen in Table 4.5, there exists a pentanomial of degree 8 which cannot be implemented with two XOR operations only. Beyond that, our intuition was that the larger the weight of the minimal polynomial, the larger the gap between the most efficient multiplication and the efficiency of multiplying by means of the companion matrix. Quantifying and demonstrating such a statement is an interesting and challenging problem which was later picked up by Kölsch who was able to prove the above conjecture [Köl19]. Another proof of the conjecture was independently found by Mesnager et al. [Mes+19].

A further interesting question is to get an improved understanding of how to most efficiently multiply with elements in proper subfields. More specifically, as a generalization of Corollary 4.3, one may ask the following question.

Question 4.1. Is the most efficient way to multiply with a subfield element given by multiplying in the subfield d times, where d is the extension degree of the field when viewed as an extension of the subfield? More precisely, given an $\alpha \in \mathbb{F}^*_{2^{m'}} \subset \mathbb{F}^*_{2^m}$ in a proper subfield of dimension $m' = \frac{m}{d}$ and let $T_{\alpha\in\mathbb{F}_{2^{m'}},B'}$ be the multiplication matrix in $\mathbb{F}_{2^{m'}}$ with an optimal XOR count. Is $T_{\alpha\in\mathbb{F}_{2^m},B} = \bigoplus_{k=1}^{d} T_{\alpha\in\mathbb{F}_{2^{m'}},B'}$ a matrix with the lowest possible XOR count for multiplication with $\alpha \in \mathbb{F}_{2^m}$? In particular, is $\mathrm{wt}_{\oplus}(T_{\alpha\in\mathbb{F}_{2^m},B}) = d\,\mathrm{wt}_{\oplus}(T_{\alpha\in\mathbb{F}_{2^{m'}},B'})$? $\Diamond$

Furthermore, when optimizing for software, similar questions can be phrased and investigating solutions that are valid for more than one specific platform is a challenging research topic.

After the publication of our results, many more papers about finding MDS matrices with few XOR operations have been published. A comprehensive overview of the related work can be found in Section 5.3.1 and Section 5.5. As in our results from Section 4.4, the typical procedure is to locally optimize the finite field multiplication, leading to an overall (global) decreased number of XOR operations. However, rather than finding *globally* optimized solutions by *local* optimizations, the problem of global optimization could be tackled more directly. This is exactly what we are doing in the next chapter, where we significantly improve all previous results with respect to this problem.

5 Shorter Linear Straight-Line Programs for MDS Matrices

The results from this paper have been published in the IACR journal *Transactions on Symmetric Cryptology* [Kra+17] and presented at the conference *Fast Software Encryption 2018* in Brugges, Belgium. This is joint work with Gregor Leander, Ko Stoffelen, and Friedrich Wiemer. All authors equally contributed.

5.1 Introduction

In the area of lightweight cryptography, several researchers started to optimize the construction of many parts of block ciphers. As already motivated in the previous chapter, there recently was a special focus on the linear layers and even more specifically the implementation of MDS matrices. That is, linear layers with an optimal branch number.

The first line of work focused solely on minimizing the chip area of the implementation. This started with the block cipher PRESENT [Bog+07] and goes over to many more designs, such as LED [Guo+11] and the hash function PHOTON [GPP11], where MDS matrices were constructed that are especially optimized for chip area by allowing a serialized implementation. However, there seem to be only a few practical applications where a small chip area is the only optimization goal and for those applications very good solutions are already available by now.

Later, starting with [Kho+14], researchers focused on round-based implementations with the goal of finding MDS constructions that minimize the number of XOR operations needed for their implementation. As explained in Section 4.2.2, the number of XOR operations needed was initially bounded by the number of ones in the binary representation of the matrix.

However, as the number of ones only gives an upper bound on the number of required XORs, several papers started to deviate from this conceptually easier but less accurate definition of

XOR count and started to consider more efficient ways of implementing MDS matrices. One of such contributions was presented in the previous chapter. Considering a $t \times t$ MDS matrix over a finite field $\mathbb{F}_{2^m}$ given as $M = (\alpha_{i,j})$ the aim was to choose the elements $\alpha_{i,j}$ in such a way that implementing all of the multiplications $x \mapsto \alpha_{i,j}x$ in parallel becomes as cheap as possible. In order to compute the matrix M entirely, those partial results have to be added together, for which an additional amount of XORs is required. It became common to denote the former cost as the overhead and the later cost, i. e., the cost of combining the partial results as a fixed, incompressible part. A whole series of papers [BKL16; Jea+17b; LS16; LW16; LW17; Sim+15; SS16a; SS16b; SS17; ZWS17], including the work from Chapter 4, managed to reduce the overhead.

From a different viewpoint, what happened was that parts of the matrix, namely the cost of multiplication with the $\alpha_{i,j}$, were extensively optimized, while taking the overall part of combining the parts as a given. That is, researchers have focused on local optimization instead of global optimization.

Indeed the task of globally optimizing is far from trivial, and thus the local optimization is a good step forward. However, as discussed in Section 4.6, the natural next step is to directly focus on the global optimization.

Interestingly, the task to optimize the cost of implementing the multiplication with a relatively large, e. g., 32×32 binary matrix, is another extensively studied line of research. It is known that the problem is NP-hard [BMP08; BMP13] and thus renders quickly infeasible for increasing matrix dimension. However, quite a number of heuristic algorithms for finding the shortest linear straight-line program, which exactly corresponds to minimizing the number of XORs, have been proposed in the literature [BFP17; BMP08; BMP13; BP10; FS10; FS12; Paa97; VSP17]. Those algorithms produce very competitive results with a rather reasonable running time for arbitrary binary matrices of dimension up to at least 32.

Thus, the natural next step in order to optimize the cost of implementing MDS matrices is to combine those two approaches. This is exactly what we are doing in our work.

Our contribution, which we achieve by applying the heuristic algorithms to find a short linear straight-line program to the case of MDS matrices, is threefold.

First, we use several well-locally-optimized MDS matrices from the literature and apply the known algorithms to all of them. This is conceptually easy, with the main problem being the implementation of those algorithms. In order to simplify this for follow-up works, we make our implementation publicly available.

This simple application leads immediately to significant improvements. For instance, we get an implementation of the AES MixColumn matrix that outperforms all previous implementations in the literature, i. e., we use 97 XORs while the best implementation before used 103 XORs ([Jea+17a]). In the case of applying it to the other constructions, we often get an implementation using *less XOR operations than what was considered fixed costs before*. That is, when (artificially) computing it, the overhead would actually be negative. This confirms our intuition that the overhead was already very well optimized in previous work, such that now optimizing globally is much more meaningful.

Second, we took a closer look at how the previous constructions compare when being globally optimized. Interestingly, the previous best construction, i. e., the MDS matrix with the smallest overhead, was most of the time *not the one with the fewest XORs*. Thus, with respect to the global

Table 5.1: Best-known MDS matrices. Matrices in the lower half are involutory. The implementations are available on GitHub and in Appendix D.

Type	Previously Best-Known		New Best-Known	
$GL(4,\mathbb{F}_2)^{4\times4}$	58	[Jea+17b; SS16b]	36^*	Eq. (5.1) (Hadamard)
$GL(8,\mathbb{F}_2)^{4\times4}$	106	[LW16]	72	Eq. (5.2) (Subfield)
$(\mathbb{F}_2[x]/0x13)^{8\times8}$	384	[Sim+15]	$196^\dagger$	Eq. (5.3) (Cauchy)
$GL(8,\mathbb{F}_2)^{8\times8}$	640	[LS16]	392	Eq. (5.4) (Subfield)
$(\mathbb{F}_2[x]/0x13)^{4\times4}$	63	[Jea+17b]	42^*	[SS16b]
$GL(8,\mathbb{F}_2)^{4\times4}$	126	[Jea+17b]	84	Eq. (5.5) (Subfield)
$(\mathbb{F}_2[x]/0x13)^{8\times8}$	424	[Sim+15]	$212^\dagger$	Eq. (5.6) (Vandermonde)
$GL(8,\mathbb{F}_2)^{8\times8}$	736	[Jea+17b]	424	Eq. (5.7) (Subfield)

* Computed with heuristic from [BMP13].
† Computed with heuristic from [Paa97].

optimum, the natural question was, which known construction actually performs best. In order to analyze that, we did extensive experimental computations to compare the distribution of the optimized implementation cost for the various constructions. The, somewhat disappointing, result is that all known constructions behave basically the same. The one remarkable exception is the subfield construction for MDS matrices, first introduced in Whirlwind [Bar+10].

Third, we looked at finding matrices that perform exceptionally well with respect to the global optimization, i. e., which can be implemented with an exceptionally low *total* number of XORs. Those results are summarized in Table 5.1. Compared to previous known matrices, ours improved on all – except for one, where the best-known matrix is the already published matrix from [SS16b].

Finally, we like to point out two restrictions of our approach. First, we do not try to minimize the amount of temporary registers needed for the implementation. Second, in line with all previous constructions, we do not minimize the circuit depth. The latter restriction is out of the scope of the current work but certainly an interesting task for the future.

All our implementations are publicly available on GitHub:

https://github.com/rub-hgi/shorter_linear_slps_for_mds_matrices

Additionally, our straight-line programs for the matrices from Table 5.1 are given in Appendix D and our straight-line program for the AES MixColumn matrix is given in Section 5.4.

5.2 Preliminaries

Before getting into details about the XOR count and previous work, let us recall some basic notations and matrix constructions.

5.2.1 Basic Notations

In this chapter, in favor of a more compact notation, we stick to the common habit and write a polynomial as its coefficient vector interpreted as a hexadecimal number, i. e., $x^4 + x + 1$ corresponds to 0x13.

We know from Section 1.2.2 and from the previous chapter that every multiplication by an element $\alpha \in \mathbb{F}_{2^m}$ can be described by a left-multiplication with a matrix $T_\alpha \in \mathbb{F}_2^{m\times m}$, called the multiplication matrix of the element α. Given a $t \times t$ matrix $M = (\alpha_{i,j})$ with $\alpha_{i,j} \in \mathbb{F}_{2^m}$ for $1 \le i, j \le t$, we define $\mathscr{B}(M) := (T_{\alpha_{i,j}}) \subseteq \mathrm{GL}(m, \mathbb{F}_2)^{t\times t} \subseteq (\mathbb{F}_2^{m\times m})^{t\times t} \cong \mathbb{F}_2^{tm\times tm}$. Its corresponding binary $tm \times tm$ matrix is called the *binary representation*. Here, $\mathrm{GL}(m, \mathbb{F}_2)$ denotes the general linear group, that is the group of invertible matrices over $\mathbb{F}_2$ of dimension $m \times m$.

Given a matrix M and a vector u, the Hamming weights $\mathrm{hw}(M)$ and $\mathrm{hw}(u)$ are again defined as the number of non-zero entries in M and u, respectively. In the case of a binary vector $v \in \mathbb{F}_2^{tm}$, we define $\mathrm{hw}_m(v) := \mathrm{hw}(v')$, where $v' \in (\mathbb{F}_2^m)^t$ is the vector that has been constructed by partitioning v into groups of m bits. Furthermore, the branch number of a matrix M is defined as $\mathrm{bn}(M) := \min_{u\neq 0}\{\mathrm{hw}(u) + \mathrm{hw}(Mu)\}$. For a binary matrix $B \in \mathbb{F}_2^{tm\times tm}$, the branch number for m-bit words is defined as $\mathrm{bn}_m(B) := \min_{u\in\mathbb{F}_2^{tm}\setminus\{0\}}\{\mathrm{hw}_m(u) + \mathrm{hw}_m(Mu)\}$.

Now, we recall the MDS definition from Section 1.3.3.

Definition 5.1. A $t \times t$ matrix M is *MDS* if and only if $\mathrm{bn}(M) = t + 1$. ◊

It has been shown, that a matrix is MDS if and only if all its square submatrices are invertible [MS77, page 321, Theorem 8]. (This theorem was used in Section 4.4 for finding the MDS conditions.) MDS matrices do not exist for every choice of t, m. The exact parameters for which MDS matrices do or do not exist are investigated in the context of the famous MDS conjecture which was initiated in [Seg55]. For binary matrices, we need to modify Definition 5.1.

Definition 5.2. A binary matrix $B \in \mathbb{F}_2^{tm\times tm}$ is *MDS* for m-bit words if and only if $\mathrm{bn}_m(M) = t + 1$. ◊

We typically deal with $t \times t$ MDS matrices over $\mathbb{F}_2^m$, respectively binary $\mathbb{F}_2^{tm\times tm}$ matrices that are MDS for m-bit words where $m \in \{4, 8\}$ is the size of the S-box. In either case, when we call a matrix MDS, the size of m will always be clear from the context when not explicitly mentioned.

It is easy to see that, if $M \in \mathbb{F}_{2^m}^{t\times t}$ is MDS, then also $\mathscr{B}(M)$ is MDS for m-bit words. On the other hand, there might also exist binary MDS matrices for m-bit words that have no according representation over $\mathbb{F}_2^m$.

Other, non-MDS matrices are also common in cipher designs. To name only a few examples: PRESENT's permutation matrix [Bog+07], lightweight implementable matrices from PRINCE [Bor+12], or PRIDE [Alb+14], or the recently used almost-MDS matrices, e. g. in MIDORI [Ban+15], or QARMA [Ava17].

5.2.2 MDS Constructions

Cauchy and Vandermonde matrices are two famous constructions for building MDS matrices. They have the advantage of being provably MDS.

However, as known from the MDS conjecture, for some parameter choices, MDS matrices are unlikely to exist. E. g., we do not know how to construct MDS matrices over $\mathbb{F}_{2^2}$ of dimension 4×4.

Definition 5.3 (Cauchy matrix). Given two disjoint sets of t elements of a field $\mathbb{F}_{2^m}$, $A = \{a_1, \dots, a_t\}$, and $B = \{b_1, \dots, b_t\}$. Then the matrix

$$M = \operatorname{cauchy}(a_1, \dots, a_t, b_1, \dots, b_t) := \begin{pmatrix} \frac{1}{a_1-b_1} & \frac{1}{a_1-b_2} & \cdots & \frac{1}{a_1-b_t} \\ \frac{1}{a_2-b_1} & \frac{1}{a_2-b_2} & \cdots & \frac{1}{a_2-b_t} \\ \vdots & & \ddots & \vdots \\ \frac{1}{a_t-b_1} & \frac{1}{a_t-b_2} & \cdots & \frac{1}{a_t-b_t} \end{pmatrix}$$

is a *Cauchy* matrix. ◊

Every Cauchy matrix is MDS, e. g. see [GR13, Lemma 1].

Definition 5.4 (Vandermonde matrix). Given a t-tuple $(a_1, \dots, a_t)$ with $a_i \in \mathbb{F}_{2^m}$. Then the matrix

$$M = \operatorname{vandermonde}(a_1, \dots, a_t) := \begin{pmatrix} a_1^0 & a_1^1 & \cdots & a_1^{t-1} \\ a_2^0 & a_2^1 & \cdots & a_2^{t-1} \\ \vdots & & \ddots & \vdots \\ a_t^0 & a_t^1 & \cdots & a_t^{t-1} \end{pmatrix}$$

is a *Vandermonde* matrix. ◊

Given two Vandermonde matrices A and B with pairwise different a_i, b_j, then the matrix AB^{-1} is MDS, see [LF04, Theorem 2]. Furthermore, if $a_i = b_i + \Delta$ for all i and an arbitrary non-zero Δ, then the matrix AB^{-1} is also involutory [LF04; Saj+12a].

5.2.3 Specially Structured Matrix Constructions

Other constructions, such as circulant, Hadamard, or Toeplitz, are not per se MDS, but they have the advantage that they greatly reduce the search space by restricting the number of submatrices that appear in the matrix. For circulant matrices, this was e. g. already noted by Daemen et al. [DKR97].

In order to generate a random MDS matrix with one of these constructions, we can choose random elements for the matrix and then check for the MDS condition. Because of many repeated submatrices, the probability to find an MDS matrix is much higher than for a fully random matrix.

Definition 5.5 (Circulant matrices). A *right-circulant* $t \times t$ matrix is defined by the elements of its first row $a_1, \dots, a_t$ as

$$M = \operatorname{circ}_{\mathrm{r}}(a_1, \dots, a_t) := \begin{pmatrix} a_1 & a_2 & \cdots & a_t \\ a_t & a_1 & \cdots & a_{t-1} \\ \vdots & & \ddots & \vdots \\ a_2 & \cdots & a_t & a_1 \end{pmatrix}.$$

A *left-circulant* $t \times t$ matrix is analogously defined as

$$M = \mathrm{circ}_\ell(a_1, \ldots, a_t) := \begin{pmatrix} a_1 & a_2 & \cdots & a_t \\ a_2 & a_3 & \cdots & a_1 \\ \vdots & & \ddots & \vdots \\ a_t & a_1 & \cdots & a_{t-1} \end{pmatrix}.$$

◊

While in the literature circulant matrices are almost always right-circulant, left-circulant matrices are equally fine for cryptographic applications. The often noted advantage of right-circulant matrices, the ability to implement the multiplication serialized and with shifts in order to save XORs, of course also applies to left-circulant matrices. Additionally, it is easy to see that $\mathrm{bn}(\mathrm{circ}_\mathrm{r}(a_1, \ldots, a_t)) = \mathrm{bn}(\mathrm{circ}_\ell(a_1, \ldots, a_t))$, since the matrices only differ in a permutation of the rows. Thus, for cryptographic purposes, it does not matter if a matrix is right-circulant or left-circulant and we will therefore simply talk about circulant matrices in general. The common practice of restricting the matrix entries to elements from a finite field comes with the problem that circulant involutory MDS matrices over finite fields do not exist, see [JA09]. But Li and Wang [LW16] showed that this can be avoided by taking the matrix elements from the general linear group.

Definition 5.6 (Hadamard matrix)**.** A *(finite field) Hadamard* matrix M is of the form

$$M = \begin{pmatrix} M_1 & M_2 \\ M_2 & M_1 \end{pmatrix},$$

where M_1 and M_2 are either Hadamard matrices themselves or one-dimensional. ◊

The biggest advantage of Hadamard matrices is the possibility to construct involutory matrices. If we choose the elements of our matrix such that the first row sums to one, the resulting matrix is involutory, see [GR13].

Definition 5.7 (Toeplitz matrix)**.** A $t \times t$ *Toeplitz* matrix M is defined by the elements of its first row $a_1, \ldots, a_t$ and its first column $a_1, a_{t+1}, \ldots, a_{2t-1}$ as

$$M = \mathrm{toep}(a_1, \ldots, a_t, a_{t+1}, \ldots, a_{2t-1}) := \begin{pmatrix} a_1 & a_2 & \cdots & a_t \\ a_{t+1} & a_1 & \ddots & a_{t-1} \\ \vdots & \ddots & \ddots & \vdots \\ a_{2t-1} & a_{2t-2} & \cdots & a_1 \end{pmatrix},$$

that is, every element defines one minor diagonal of the matrix. ◊

To the best of our knowledge, Sarkar and Syed [SS16b] were the first to scrutinize Toeplitz matrices in the context of XOR counts.

Finally, the subfield construction was first used to construct lightweight linear layers in the WHIRLWIND hash function [Bar+10, Section 2.2.2] and later used in [Alb+14; Cho+12; Jea+17b;

Kho+14; Sim+15]. As its name suggests, the subfield construction was originally defined only for matrices over finite fields: a matrix with coefficients in $\mathbb{F}_{2^m}$ can be used to construct a matrix with coefficients in $\mathbb{F}_{2^{2m}}$. Here, we use the natural extension to binary matrices.

Definition 5.8 (Subfield matrix)**.** Given a $t \times t$ matrix M with entries $\alpha_{i,j} \in \mathbb{F}_2^{m \times m}$. The subfield construction of M is then a $t \times t$ matrix M' with

$$M' = \text{subfield}(M) := \left(\alpha'_{i,j}\right),$$

where each $\alpha'_{i,j} = \left(\begin{smallmatrix} \alpha_{i,j} & 0 \\ 0 & \alpha_{i,j} \end{smallmatrix}\right) \in \mathbb{F}_2^{2m \times 2m}$. $\lozenge$

This definition is straightforward to extend for more than one copy of the matrix M.

The subfield construction has some very useful properties, see [Bar+10; Jea+17b; Kho+14; Sim+15].

Lemma 5.1. *For the subfield construction, the following properties hold:*

1. *Let M be a matrix that can be implemented with d XORs. Then the matrix $M' = \text{subfield}(M)$ can be implemented with $2d$ XORs.*
2. *Let M be an MDS matrix for m-bit words. Then $M' = \text{subfield}(M)$ is MDS for $2m$-bit words.*
3. *Let M be an involutory matrix. Then $M' = \text{subfield}(M)$ is also involutory.*

$\lozenge$

Proof.

(1) Due to the special structure of the subfield construction, we can split the multiplication by M' into two multiplications by M, each on one half of the input bits. Hence, the XOR count doubles.

(2) We want to show that $\text{hw}_{2m}(u) + \text{hw}_{2m}(M'u) \geq t + 1$ for every non-zero u. We split u into two parts u_1 and u_2, each containing alternating halves of the elements of u. As described in [Kho+14], the multiplication of M' and u is the same as the multiplication of the original matrix M and each of the two u_i, if we combine the results according to our splitting. Let $b = \text{hw}_{2m}(u) > 0$. Then, we have $b \geq \text{hw}_m(u_1)$ and $b \geq \text{hw}_m(u_2)$. Without loss of generality, let $\text{hw}_m(u_1) > 0$. Since M is MDS for m-bit words, we have $\text{hw}_m(Mu_1) \geq n - b + 1$ which directly leads to $\text{hw}_{2m}(M'u) \geq n - b + 1$.

(3) As in the above proof, this property is straightforward to see. We want to show that $M'M'u = u$ for any vector u. Again, we split u into two parts, u_1 and u_2, each containing alternating halves of the elements of u. Now, we need to show that $MMu_i = u_i$. This trivially holds, as M is involutory.

□

With respect to cryptographic designs, this means the following: assume we have a linear straight-line program with d XORs for an (involutory) $t \times t$ MDS matrix and m-bit S-boxes. We can then easily construct a linear straight-line program with $2d$ XORs for an (involutory) $t \times t$ MDS matrix and $2m$-bit S-boxes.

5.3 Related Work

In 2014, [Kho+14] introduced the notion of XOR count as a metric to compare the area-efficiency of matrix multiplications. Following that, there has been a lot of work [BKL16; Jea+17b; LS16; LW16; LW17; Sim+15; SS16a; SS16b; SS17; ZWS17] , including the work from Chapter 4, on finding MDS matrices that can be implemented with as few XOR gates as possible in the round-based scenario.

In an independent line of research, the problem of implementing binary matrix multiplications with as few XORs as possible was extensively studied [BFP17; BMP08; BMP13; BP10; FS10; FS12; Paa97; VSP17].

In this section, we depict these two fields of research and show how they can be combined.

5.3.1 Local Optimizations

Let us first recall the scenario. In a round-based implementation, the matrix is implemented as a fully unrolled circuit. Thus, in the XOR count metric, the goal is to find a matrix that can be implemented with a circuit of as few (2-input) XOR gates as possible. Of course, the matrix has to fulfill some criteria, typically it is MDS. For finding matrices with a low XOR count, the question of how to create a circuit for a given matrix must be answered.

The usual way for finding an implementation of $t \times t$ matrices over $\mathbb{F}_{2^m}$ was introduced in [Kho+14]. As each of the t output components of a matrix-vector multiplication is computed as a sum over t products, the implementation is divided into two parts. Namely the single multiplications on the one hand and addition of the products on the other hand. As $\mathbb{F}_{2^m} \cong \mathbb{F}_2^m$, an addition of two elements from $\mathbb{F}_2^m$ requires m XORs and thus adding up the products for all rows requires $t(t-1)m$ XORs in the case of an MDS matrix where every element is non-zero. If one implements the matrix like this, these $t(t-1)m$ XORs are a fixed part that cannot be changed. Accordingly, many papers [BKL16; LS16; LW16; ZWS17], including the work from Chapter 4, just state the number of XORs for the single field multiplications when presenting results. The other costs are regarded as inevitable. The goal then boils down to constructing matrices with elements for which multiplication can be implemented with few XORs. Thus, the original goal of finding a global implementation for the matrix is approached by locally looking at the single matrix elements.

To count the number of XORs for implementing a single multiplication with an element $\alpha \in \mathbb{F}_{2^m}$, the multiplication matrix $T_\alpha \in \mathbb{F}_2^{m\times m}$ is considered. Such a matrix can be implemented in a straightforward way with $\text{hw}(T_\alpha) - m$ XORs by simply implementing every XOR of the output components. We call this the *naive* implementation of a matrix and when talking about the naive XOR count of a matrix, we mean the $\text{hw}(T_\alpha) - m$ XORs required for the naive implementation. In [Jea+17b], this is called d-XOR. It is the easiest and most frequently used method of counting XORs. Of course, in the same way, we can also count the XORs of other matrices over $\mathbb{F}_2^{m\times m}$, i. e., also matrices that were not originally defined over finite fields.

For improving the XOR count of the single multiplications, two methods have been introduced in Chapter 4. First, if the matrix is defined over some finite field, one can consider different field representations that lead to different multiplication matrices with potentially different Hamming weights, see [BKL16; Sim+15; SS16a]. Second, by reusing intermediate results, a

$m \times m$ binary matrix T_α might be implemented with less than $\mathrm{hw}(T_\alpha) - m$ XORs, see [BKL16; Jea+17b]. In [Jea+17b], this is called s-XOR. The according definitions from [Jea+17b] and Chapter 4 [BKL16] require that all operations must be carried out on the input registers. That is, in contrast to the naive XOR count, no temporary registers were allowed. However, as we consider round-based hardware implementations, there is no need to avoid temporary registers since these are merely wires between gates.

Nowadays, the XOR count of implementations is mainly dominated by the $t(t-1)m$ XORs for putting together the locally optimized multiplications. Lastly, we seem to hit a threshold and new results often improve existing results only by very few XORs. The next natural step is to shift the focus from local optimization of the single elements to the global optimization of the whole matrix. This was also formulated as future work in [Jea+17b]. As described in Section 5.2, we can use the binary representation to write a $t \times t$ matrix over $\mathbb{F}_{2^m}$ as a binary $tm \times tm$ matrix. First, we note, that the naive XOR count of the binary representation is exactly the naive XOR count of implementing each element multiplication and finally adding the results. But if we look at the optimization technique of reusing intermediate results for the whole $tm \times tm$ matrix, we now have many more degrees of freedom. For the MixColumn matrix, there already exists some work that goes beyond local optimization. An implementation with 108 XORs has been presented in [BBR16a; BBR16b; Sat+01] and an implementation with 103 XORs in [Jea+17a]. A first step to a global optimization algorithm was done in [Zha+16]. However, their heuristic did not yield very good results and they finally had to go back to optimizing submatrices.

Interestingly, much better algorithms for exactly this problem are already known from a different line of research.

5.3.2 Global Optimizations

Implementing binary matrices with as few XOR operations as possible is also known as the problem of finding the *shortest linear straight-line program* [BMP13; FS10] over the finite field with two elements. Although this problem is NP-hard [BMP08; BMP13], attempts have been made to find exact solutions for the minimum number of XORs. Fuhs and Schneider-Kamp [FS10; FS12] suggested reducing the problem to satisfiability of Boolean logic. They presented a general encoding scheme for deciding if a matrix can be implemented with a certain number of XORs. Now, for finding the optimal implementation, they repeatedly use SAT solvers for a decreasing number of XORs. Then, when they know that a matrix can be implemented with d XORs, but cannot be implemented with $d-1$ XORs, they are able to present d as the optimal XOR count. They used this technique to search for the minimum number of XORs necessary to compute a binary matrix of size 21×8, which is the first linear part of the AES S-box when it is decomposed into two linear parts and a minimal non-linear core. While it worked to find a solution with 23 XORs and to show that no solution with 20 XORs exists, it turned out to be infeasible to prove that a solution with 22 XORs does not exist and that 23 is therefore the minimum. In general, this approach quickly becomes infeasible for larger matrices. Stoffelen [Sto16] applied it successfully to a small 7×7 matrix, but did not manage to find a provably minimal solution with a specific matrix of size 19×5. However, there do exist heuristics to efficiently find short linear straight-line programs also for larger binary matrices.

Back in 1997, Paar [Paa97] studied how to optimize the arithmetic used by Reed-Solomon encoders. Essentially, this boils down to reducing the number of XORs that are necessary for a constant multiplier over the field $\mathbb{F}_{2^m}$. Paar described two algorithms that find a local optimum. Intuitively, the idea of the algorithms is to iteratively eliminate *common subexpressions*. Let T_α be the multiplication matrix, to be applied to a variable field element $x = (x_1, \ldots, x_m) \in \mathbb{F}_2^m$. The first algorithm for computing $T_\alpha x$, denoted PAAR1 in the rest of this work, finds a pair (i, j), with $i \neq j$, where the bitwise AND between columns i and j of T_α has the highest Hamming weight. In other words, it finds a pair (x_i, x_j) that occurs most frequently as a subexpression in the output bits of $T_\alpha x$. The XOR between those is then computed, and M is updated accordingly, with $x_i + x_j$ as a newly available variable. This is repeated until there are no common subexpressions left.

The second algorithm, denoted PAAR2, is similar, but differs when multiple pairs are equally common. Instead of just taking the first pair, it recursively tries all of them. The algorithm is therefore much slower, but can yield slightly improved results. Compared to the naive XOR count, Paar noted an average reduction in the number of XORs of 17.5% for matrices over $\mathbb{F}_{2^4}$ and 40% for matrices over $\mathbb{F}_{2^8}$.

In 2009, Bernstein [Ber09] presented an algorithm for efficiently implementing linear maps modulo 2. Based on this and on [Paa97], a new algorithm was presented in [BC14]. However, the algorithms from [BC14; Ber09] have a different framework in mind and yield a higher number of XORs compared to [Paa97].

Paar's algorithms lead to so-called *cancellation-free* programs. This means that for every XOR operation $u + v$, none of the input bit variables x_i occurs in both u and v. Thus, the possibility that two variables cancel each other out is never taken into consideration, while this may in fact yield a more efficient solution in terms of the total number of XORs. In 2008, Boyar et al. [BMP08] showed that cancellation-free techniques can often not be expected to yield optimal solutions for non-trivial inputs. They also showed that, even under the restriction to cancellation-free programs, the problem of finding an optimal program is NP-complete.

Around 2010, Boyar and Peralta [BP10] came up with a heuristic that is not cancellation-free and that improved on Paar's algorithms in most scenarios. Their idea was to keep track of a distance vector that contains, for each targeted expression of an output bit, the minimum number of additions of the already computed intermediate values that are necessary to obtain that target. To decide which values will be added, the pair that minimizes the sum of new distances is picked. If there is a tie, the pair that maximizes the Euclidean norm of the new distances is chosen. Additionally, if the addition of two values immediately leads to a targeted output, this can always be done without searching further. This algorithm works very well in practice, although it is slower compared to PAAR1.

Next to using the Euclidean norm as tie breaker, they also experimented with alternative criteria. For example, choosing the pair that maximizes the Euclidean norm minus the largest distance, or choosing the pair that maximizes the Euclidean norm minus the difference between the two largest distances. The results were then actually very similar. Another tie-breaking method is to flip a coin and choose a pair randomly. The algorithm is now no longer deterministic and can be run multiple times. The lowest result can then be used. This performed slightly better, but of course processing again takes longer.

The results of [BMP08] and [BP10] were later improved and published in [BMP13].

In early 2017, Visconti et al. [VSP17] explored the special case where the binary matrix is dense. They improved the heuristic on average for dense matrices by first computing a *common path*, an intermediate value that contains most variables. The original algorithm is then run starting from this common path.

At BFA 2017, Boyar et al. [BFP17] presented an improvement that simultaneously reduces the number of XORs and the depth of the resulting circuit.

We refer to this family of heuristics [BFP17; BMP08; BMP13; BP10; VSP17] as the BP heuristics.

5.4 Results

Using the techniques described above, we now give optimized XOR counts and implementations of published matrices. Next, we analyze the statistical behavior of matrix constructions. Finally, we summarize the best results.

5.4.1 Improved Implementations of Matrices

Using the heuristic methods that are described in the previous section, we can easily and significantly reduce the XOR counts for many matrices that have been used in the literature. The running times for the optimizations are in the range of seconds to minutes. All corresponding implementations are available in the GitHub repository. Table 5.2 lists results for matrices that have been suggested in previous works where it was an explicit goal to find a lightweight MDS matrix. While the constructions themselves will be compared in Section 5.4.2, this table deals with the suggested instances.

A number of issues arise from this that are worth highlighting. First of all, it should be noted that without any exception, the XOR count for every matrix could be reduced with little effort. Second, it turns out that there are many cases where the $t(t-1)m$ XORs for summing the products for all rows is not even a correct lower bound. In fact, all the 4×4 matrices over $\mathrm{GL}(4, \mathbb{F}_2)$ that we studied can be implemented in *at most* 48 XORs.

What may be more interesting, is whether the XOR count as it was used previously is in fact a good predictor for the actual implementation cost as given by the heuristical methods. Here we see that there are some differences. For example, [LW16]'s circulant 4×4 matrices over $\mathrm{GL}(8, \mathbb{F}_2)$ first compared very favorably, but we now find that the subfield matrix of [Jea+17b] requires fewer XORs.

Regarding involutory matrices, it was typically the case that there was an extra cost involved to meet this additional criterion. However, the heuristics sometimes find implementations with even fewer XORs than the non-involutory matrix that was suggested. See for example the matrices of [SS16b] in the table.

Aside from these matrices, we also looked at MDS matrices that are used by various ciphers and hash functions. Table 5.3 lists their results. Not all MDS matrices that are used in ciphers are incorporated here. In particular, Led [Guo+11], Photon [GPP11], and Primates [And+14] use efficient serialized MDS matrices. Comparing these to our "unrolled" implementations would be somewhat unfair.

The implementation of the MDS matrix used in AES with 97 XORs was, by the time of presenting the results, the most efficient implementation so far and improved on the previous

Table 5.2: Comparison of 4×4 and 8×8 MDS matrices over $\mathrm{GL}(4, \mathbb{F}_2)$ and $\mathrm{GL}(8, \mathbb{F}_2)$.

Matrix	Naive	Literature	PAAR1	PAAR2	BP
4×4 matrices over $\mathrm{GL}(4, \mathbb{F}_2)$					
[Sim+15] (Hadamard)	68	20 + 48	50	48[*]	48
[LS16] (Circulant)	60	12 + 48	49	46[*]	44
[LW16] (Circulant)[†]	60	12 + 48	48	47[*]	44
Section 4.4.2, Table 4.7 [BKL16] (Circulant)[†]	64	12 + 48	48	47	**42**
[SS16b] (Toeplitz)	58	**10 + 48**	46	45[*]	43
[Jea+17b]	61	**10 + 48**	48	47	43
[Sim+15] (Hadamard, Involutory)	72	24 + 48	52	48[*]	48
[LW16] (Hadamard, Involutory)	72	24 + 48	51	48[*]	48
[LW16] (Circulant, Involutory)	68	20 + 48	48	48	48
[SS16b] (Involutory)	64	16 + 48	50	48	**42**
[Jea+17b] (Involutory)	68	**15 + 48**	51	47[*]	47
4×4 matrices over $\mathrm{GL}(8, \mathbb{F}_2)$					
[Sim+15] (Subfield)	136	40 + 96	100	98[*]	100
[LS16] (Circulant)	128	28 + 96[1]	116	116	112
[LW16]	106	**10 + 96**	102	102	102
Section 4.4.2, Table 4.7 [BKL16] (Circulant)	136	24 + 96	116	112[*]	110
[SS16b] (Toeplitz)	123	24 + 96[1]	110	108	107
[Jea+17b] (Subfield)	122	20 + 96	96	95[*]	**86**
[Sim+15] (Subfield, Involutory)	144	40 + 96[1]	104	101[*]	100
[LW16] (Hadamard, Involutory)	136	40 + 96	101	97[*]	**91**
[LW16] (Circulant, Involutory)	132	36 + 96	104	104[*]	97
[SS16b] (Involutory)	160	64 + 96	110	109[*]	100
[Jea+17b] (Subfield, Involutory)	136	**30 + 96**	102	100[*]	**91**
8×8 matrices over $\mathrm{GL}(4, \mathbb{F}_2)$					
[Sim+15] (Hadamard)	432	**160 + 224**[1]	210	209[*]	**194**
[SS17] (Toeplitz)	394	170 + 224	205	205[*]	201
[Sim+15] (Hadamard, Involutory)	512	**200 + 224**[1]	222	222[*]	**217**
8×8 matrices over $\mathrm{GL}(8, \mathbb{F}_2)$					
[Sim+15] (Hadamard)	768	256 + 448[1]	474	—	467
[LS16] (Circulant)	688	**192 + 448**[1]	464	—	447
Section 4.4.2, Table 4.7 [BKL16] (Circulant)	784	208 + 448[1]	506	—	498
[SS17] (Toeplitz)	680	232 + 448	447	—	**438**
[Sim+15] (Hadamard, Involutory)	816	320 + 448[1]	430	—	**428**
[Jea+17b] (Hadamard, Involutory)	1152	**288 + 448**	620	—	599

[*] Stopped algorithm after three hours runtime.

[†] In Chapter 4 [BKL16] and[LW16], not only one matrix is given, but instead whole classes of MDS matrices were given. For Chapter 4 [BKL16], we chose the canonical candidate from its class. For [LW16], we chose the matrix presented as an example in the paper.

[1] Reported by [Jea+17b].

Table 5.3: Matrices used in ciphers or hash functions. Note that matrices in the lower part of the table, marked with ‖, are not MDS. Additionally, these matrices are commonly not a target for “XOR count”-based implementation optimizations, as they are per se very efficiently implementable. Additionally, these matrices are commonly not a target for “XOR count”-based implementation optimizations, as they are per se very efficiently implementable.

Cipher	Type	Naive	Literature	PAAR1	PAAR2	BP
AES [DR02]‡ (Circ.)	$(\mathbb{F}_2[x]/\texttt{0x11b})^{4\times4}$	152	$7+96$[1]	108	108*	97†
ANUBIS [BRa] (Had., Invol.)	$(\mathbb{F}_2[x]/\texttt{0x11d})^{4\times4}$	184	$80+96$[2]	121	121*	106
CLEFIA M_0 [Shi+07] (Had.)	$(\mathbb{F}_2[x]/\texttt{0x11d})^{4\times4}$	184	$80+96$[2]	121	121*	106
CLEFIA M_1 [Shi+07] (Had.)	$(\mathbb{F}_2[x]/\texttt{0x11d})^{4\times4}$	208	—[5]	121	121*	111
FOX MU4 [JV04]	$(\mathbb{F}_2[x]/\texttt{0x11b})^{4\times4}$	219	—[5]	144	143*	137
TWOFISH [Sch+98]	$(\mathbb{F}_2[x]/\texttt{0x169})^{4\times4}$	327	—[5]	151	149*	129
FOX MU8 [JV04]	$(\mathbb{F}_2[x]/\texttt{0x11b})^{8\times8}$	1257	—[5]	611	—	594
GRØSTL [Gau+] (Circ.)	$(\mathbb{F}_2[x]/\texttt{0x11b})^{8\times8}$	1112	$504+448$[2]	493	—	475
KHAZAD [BRb] (Had., Invol.)	$(\mathbb{F}_2[x]/\texttt{0x11d})^{8\times8}$	1232	$584+448$[2]	488	—	507
WHIRLPOOL [BRc]§ (Circulant)	$(\mathbb{F}_2[x]/\texttt{0x11d})^{8\times8}$	840	$304+448$[2]	481	—	465
JOLTIK [JNP14a] (Had., Invol.)	$(\mathbb{F}_2[x]/\texttt{0x13})^{4\times4}$	72	$20+48$[2]	52	48	48
SMALLSCALE AES [CMR05] (Circ.)	$(\mathbb{F}_2[x]/\texttt{0x13})^{4\times4}$	72	—[5]	54	54	47
WHIRLWIND M_0 [Bar+10] (Had., Subf.)	$(\mathbb{F}_2[x]/\texttt{0x13})^{8\times8}$	488	$168+224$[2]	218	218*	212
WHIRLWIND M_1 [Bar+10] (Had., Subf.)	$(\mathbb{F}_2[x]/\texttt{0x13})^{8\times8}$	536	$184+224$[2]	246	244*	235
QARMA128 [Ava17]‖ (Circ.)	$(\mathbb{F}_2[x]/\texttt{0x101})^{4\times4}$	64	—[5]	48	48	48
ARIA [Kwo+03]‖ (Invol.)	$(\mathbb{F}_2)^{128\times128}$	768	480[3]	416	—	—
MIDORI [Ban+15]‖,¶ (Circ.)	$(\mathbb{F}_{2^4})^{4\times4}$	32	—[5]	24	24	24
PRINCE $\widehat{M}_0$, $\widehat{M}_1$ [Bor+12]‖	$(\mathbb{F}_2)^{16\times16}$	32	—[5]	24	24	24
PRIDE L_0–L_3 [Alb+14]‖	$(\mathbb{F}_2)^{16\times16}$	32	—[5]	24	24	24
QARMA64 [Ava17]‖ (Circ.)	$(\mathbb{F}_2[x]/\texttt{0x11})^{4\times4}$	32	—[5]	24	24	24
SKINNY64 [Bei+16]‖	$(\mathbb{F}_{2^4})^{4\times4}$	16	12[4]	12	12	12

* Stopped algorithm after three hours runtime.
† For the implementation see our GITHUB repository.
‡ Also used in other primitives, e. g. its predecessor SQUARE [DKR97], and MUGI [Wat+02].
§ Also used in MAELSTROM [FBR06].
¶ Also used in other ciphers, e. g. MANTIS [Bei+16], and FIDES [Bil+13].
‖ Not an MDS matrix.

[1] Reported by [Jea+17a].
[2] Reported by [Jea+17b].
[3] Reported by [Bir+04].
[4] Reported by the designers.
[5] We are not aware of any reported results for this matrix.

implementation of 103 XORs, reported by [Jea+17a]. As this is probably the most interesting of the presented implementations, the according straight-line program is presented in Listing 5.1, where x, y, t denote input, output, and temporary values, respectively. However, as noted before, all implementations are available online on GitHub (https://github.com/rub-hgi/shorter_linear_slps_for_mds_matrices). Very recently, an implementation of the AES MixColumn matrix with 95 XORs has been reported by Banik et al. [BFI19].

Listing 5.1: Straight-line program for the AES MixColumn matrix that uses 97 XORs.

```
t0 = x7 + x15
t1 = x23 + x31
t2 = x7 + x31
t3 = x15 + x23
t4 = x0 + x8
t5 = x6 + x14
t6 = x5 + x29
t7 = x16 + x24
t8 = x22 + x30
t9 = x13 + x21
t10 = x1 + x9
t11 = x10 + x18
t12 = x2 + x26
t13 = x17 + x25
t14 = x4 + x12
t15 = x3 + x27
t16 = x20 + x28
t17 = x11 + x19
t18 = x0 + t3
y8 = t7 + t18
t20 = x16 + t4
y24 = t2 + t20
t22 = t1 + t7
y16 = t20 + t22
t24 = x8 + t7
y0 = t0 + t24
t26 = x6 + t8
y14 = t9 + t26
t28 = x7 + x22
t29 = x21 + t5
t30 = x5 + y14
y6 = t29 + t30
t32 = x10 + t12
y18 = t13 + t32
t34 = x25 + t11
t35 = x1 + x2
y26 = t34 + t35
t37 = x9 + y18
y10 = t34 + t37
t39 = t0 + t32
t40 = x13 + t6
y21 = t16 + t40
t42 = x28 + t9
t43 = x4 + x5
y29 = t42 + t43
t45 = x12 + y21
y13 = t42 + t45
t47 = x14 + t1
y15 = t28 + t47
t49 = x6 + x15
y7 = t47 + t49
t51 = x15 + t2
y23 = t8 + t51
t53 = x22 + t5
y30 = t6 + t53
t55 = x25 + t10
y17 = t22 + t55
t57 = x26 + t10
y2 = t11 + t57
t59 = x29 + x30
y22 = t29 + t59
t61 = x29 + t9
y5 = t14 + t61
t63 = t3 + t26
y31 = t28 + t63
t65 = t4 + t37
y1 = t39 + t65
t67 = x1 + t13
t68 = t18 + t20
y9 = t67 + t68
t70 = x3 + t2
t71 = x27 + t14
t72 = x20 + t70
y28 = t71 + t72
t74 = t12 + t17
y27 = t70 + t74
t76 = x2 + x27
t77 = t39 + t74
y3 = t76 + t77
```

```
t79 = t3 + t11
t80 = x19 + t15
y11 = t79 + t80
t82 = x4 + t3
t83 = t16 + t17
y12 = t82 + t83
t85 = x19 + t1
t86 = x28 + t71
y20 = t85 + t86
t88 = x9 + y17
t89 = y1 + t68
y25 = t88 + t89
t91 = x11 + y27
t92 = y3 + t79
y19 = t91 + t92
t94 = t14 + t70
t95 = y12 + t85
y4 = t94 + t95
```

As a side note, cancellations do occur in this implementation, we thus conjecture that such a low XOR count is not possible with cancellation-free programs.

5.4.2 Statistical Analysis

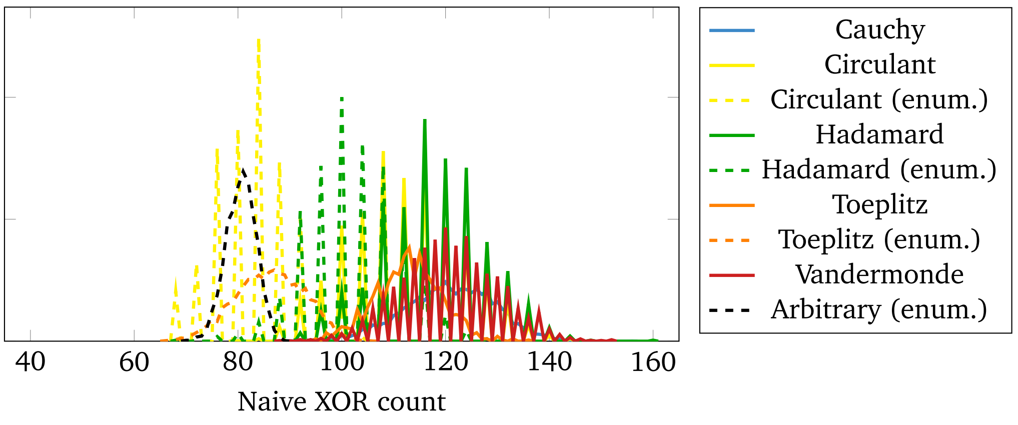
Several constructions for building MDS matrices are known. But it is not clear which one is the best when we want to construct matrices with a low XOR count. In this section, we present experimental results on different constructions and draw conclusions for the designer. We also examine the correlation between naive and heuristically improved XOR counts. When designing MDS matrices with a low XOR count, we are faced with two major questions. First, which construction is preferable? Our intuition in this case is, a better construction has better statistical properties, compared to an inferior construction. We are aware that the statistical behavior of a construction might not be very important for a designer who only looks for a single, very good instance. Nevertheless, we use this as a first benchmark. Second, is it a good approach to choose the matrices as sparse as possible? In order to compare the listed constructions, we construct random instances of each and then analyze them with statistical means.

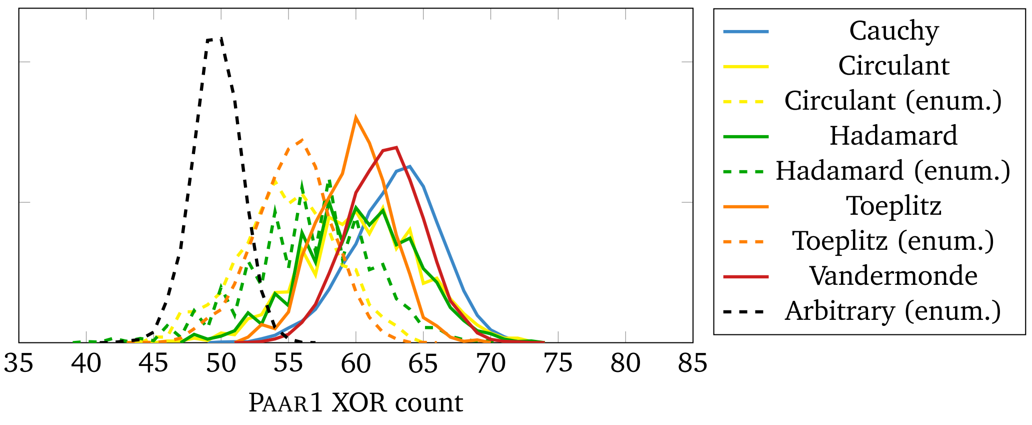
Building Cauchy and Vandermonde matrices is straightforward as we only need to choose the defining elements randomly from the underlying field. For the other constructions, we use the following backtracking method to build random MDS constructions of dimension 4×4. Choose the new random elements from $\mathrm{GL}(m, \mathbb{F}_2)$ that are needed for the matrix construction in a step-by-step manner. In each step, construct all new square submatrices. If any of these is not invertible, discard the chosen element and try a new one. In the case that no more elements are left, go one step back and replace that element with a new one, then again check the according square submatrices, and so on. Eventually, we end up with an MDS matrix because we iteratively checked that every square submatrix is invertible. The method is also trivially derandomizable, by not choosing the elements randomly, but simply enumerating them in any determined order.

Apart from applying this method to the above-mentioned constructions, we can also use it to construct an *arbitrary*, i. e. unstructured, matrix that is simply defined by its 16 elements. This approach was already described in [Jea+17b].

In this manner, we generated 1 000 matrices for each construction and computed the distributions for the naive XOR count, the optimized XOR count of PAAR1, and BP. Table 5.4 lists the statistical parameters of the resulting distributions and Fig. 5.1 depicts them (the sample size N is the same for Table 5.4 and Figs. 5.1, 5.2 and C.1 to C.4).

The most obvious characteristic of the statistical distributions is that the means μ do not differ much for all randomized constructions. The variance σ^2, on the contrary, differs much





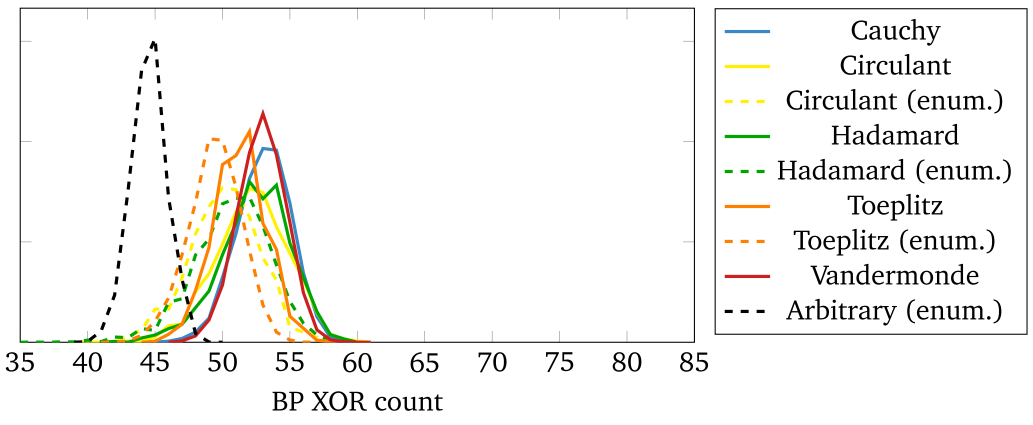


Figure 5.1: *XOR count distributions for* 4×4 *MDS matrix constructions over* $\mathrm{GL}(4, \mathbb{F}_2)$.

Table 5.4: Distributions for differently optimized XOR counts. By N we denote the sample size, μ is the mean, and σ^2 the variance.

		Naive		PAAR1		BP	
Construction	N	μ	σ^2	μ	σ^2	μ	σ^2
4×4 matrices over $\mathrm{GL}(4, \mathbb{F}_2)$							
Cauchy	1 000	120.7	77.3	62.9	11.0	53.1	4.0
Circulant	1 000	111.8	117.1	60.4	19.2	52.1	7.1
Hadamard	1 000	117.5	99.6	60.2	17.8	52.4	6.9
Toeplitz	1 000	112.8	39.9	59.9	7.4	51.3	3.9
Vandermonde	1 000	120.6	87.6	62.2	8.1	52.9	3.1
enumerated 4×4 matrices over $\mathrm{GL}(4, \mathbb{F}_2)$							
Circulant	1 000	82.9	53.0	54.9	13.5	50.1	6.7
Hadamard	1 000	102.1	76.0	56.7	20.6	50.6	8.0
Toeplitz	1 000	86.1	43.9	55.3	8.3	49.4	3.9
Arbitrary	1 000	80.5	8.3	49.7	3.2	44.5	1.8
4×4 matrices over $\mathrm{GL}(8, \mathbb{F}_2)$							
Cauchy	1 000	454.1	467.2	215.1	39.6	—	—
Vandermonde	1 000	487.3	597.4	220.2	44.3	—	—
4×4 subfield matrices over $\mathrm{GL}(4, \mathbb{F}_2)$							
Cauchy	1 000	241.1	312.1	125.8	44.2	—	—
Vandermonde	1 000	240.6	452.8	121.8	47.1	—	—

more. This is most noticeable for the naive XOR count, while the differences get much smaller when the XOR count is optimized with the PAAR1 or BP heuristic. One might think that the construction with the lowest optimized average XOR count, which is for matrices over $\mathrm{GL}(4, \mathbb{F}_2)$ the arbitrary construction with enumerated elements, yields the best results. However, the best matrix we could find for this dimension was a Hadamard matrix. An explanation for this might be the higher variance that leads to some particularly bad and some particularly good results.

The graphs in Fig. 5.1 convey a similar hypothesis. Looking only at the naive XOR count, we can notice some differences. For example, circulant matrices seem to give better results than, e. g., Hadamard matrices. Additionally, the naive XOR count increases step-wise as not every possible count occurs. But when optimizing the XOR count, the distributions get smoother and more similar.

We conclude that all constructions give similarly good matrices when we are searching for

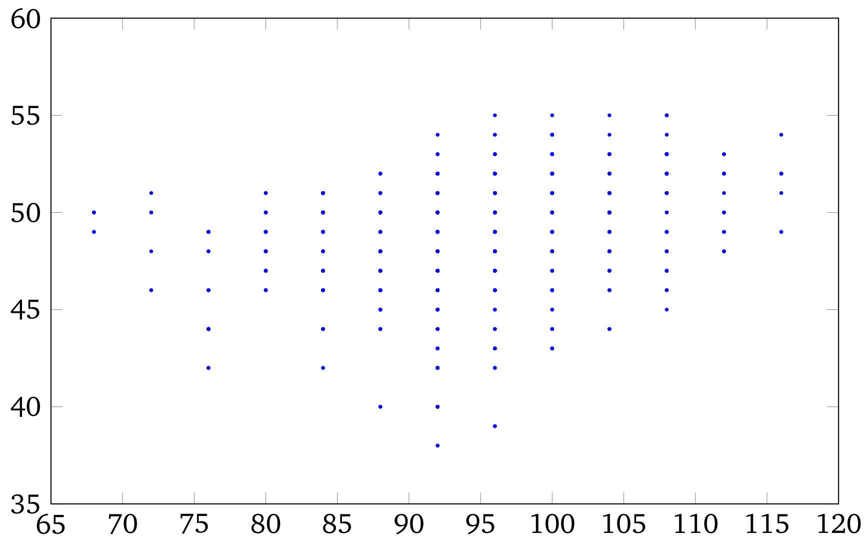


Figure 5.2: *Correlations between naive (x-axis) and* BP *(y-axis) XOR counts for enumerated Hadamard matrices.*

the matrix with the lowest XOR count, with one important exception. For randomly generated matrices the XOR count increases by a factor of four if we double the parameter m. Table 5.4 covers this for Cauchy and Vandermonde matrices. We do not compute the statistical properties for Circulant, Hadamard and Toeplitz matrices with elements of $\mathrm{GL}(8, \mathbb{F}_2)$, as the probability to find a random MDS instance for these constructions is quite low. Thus, generating enough instances for a meaningful statistical comparison is computationally tough and – as we deduce from a much smaller sample size – the statistical behavior looks very similar to that of the Cauchy and Vandermonde matrices. Instead, and as already mentioned in Lemma 5.1, the subfield construction has a much more interesting behavior. It simply doubles the XOR count. The lower half of Table 5.4 confirms this behavior.

Thus, when it is computationally infeasible to exhaustively search through all possible matrices, it seems to be a very good strategy to use the subfield construction with the best-known results from smaller dimensions. This conclusion is confirmed by the fact that our best results for matrices over $\mathrm{GL}(8, \mathbb{F}_2)$ are always subfield constructions based on matrices over $\mathrm{GL}(4, \mathbb{F}_2)$.

Next, we want to approach the question if choosing MDS matrices with low Hamming weight entries is a good approach for finding low XOR count implementations. To give a first intuition of the correlation between naive and optimized XOR count, we plot the naive XOR count against the optimized one. For one exemplary plot see Fig. 5.2, which corresponds to the construction that we used to find the best 4×4 MDS matrix for $m = 4$. The remaining plots are in the appendix, see Figs. C.1 to C.4.

In Fig. 5.2 one can see that several options can occur. While there is a general tendency of higher naive XOR counts leading to higher optimized XOR counts, the contrary is also possible. For example, there are matrices which have a low naive XOR count (left in the figure), while still having a somewhat high optimized XOR count (top part of the figure). But there are also matrices where a higher naive XOR count results in a much better optimized XOR count. The consequence is that we cannot restrict ourselves to very sparse matrices when searching for the best XOR count, but also have to take more dense matrices into account. A possible explanation for this behavior is that the heuristics have more possibilities for optimizations when the matrix is not sparse.

5.4.3 Best results

Let us conclude by specifying the currently best MDS matrices. The notation $M_{t,m}$ denotes a $t \times t$ matrix with entries from $\mathrm{GL}(m, \mathbb{F}_2)$, an involutory matrix is labeled with the superscript i. Table 5.1 covers non-involutory and involutory matrices of dimension 4×4 and 8×8 over $\mathrm{GL}(4, \mathbb{F}_2)$ and $\mathrm{GL}(8, \mathbb{F}_2)$. $M_{8,4}$ and $M^i_{8,4}$ are defined over $\mathbb{F}_2[x]/\mathtt{0x13}$.

The matrices mentioned there are the following:

$$M_{4,4} = \mathrm{hadamard}(\begin{pmatrix} 0&0&0&1\\0&0&1&0\\0&1&0&0\\1&0&0&0 \end{pmatrix}, \begin{pmatrix} 0&0&1&1\\1&0&0&1\\1&1&0&0\\0&1&0&0 \end{pmatrix}, \begin{pmatrix} 1&1&0&1\\1&1&0&0\\0&1&0&1\\0&0&1&0 \end{pmatrix}, \begin{pmatrix} 1&1&0&0\\0&1&0&1\\1&0&1&1\\0&0&0&1 \end{pmatrix}) \tag{5.1}$$

$$M_{4,8} = \mathrm{subfield}(M_{4,4}) \tag{5.2}$$

$$M_{8,4} = \mathrm{cauchy}\begin{pmatrix} x^3{+}x^2,x^3,x^3{+}x{+}1,x{+}1,0,x^3{+}x^2{+}x{+}1,x^2,x^2{+}x{+}1, \\ 1,x^2{+}1,x^3{+}x^2{+}x,x^3{+}1,x^3{+}x^2{+}1,x^2{+}x,x^3{+}x,x \end{pmatrix} \tag{5.3}$$

$$M_{8,8} = \mathrm{subfield}(M_{8,4}) \tag{5.4}$$

$$M^i_{4,8} \text{ is the subfield construction applied to [SS16b, Example 3]} \tag{5.5}$$

$$M^i_{8,4} = \mathrm{vandermonde}\begin{pmatrix} x^3{+}x{+}1,x{+}1,x^3{+}x^2{+}x,x^3{+}x^2{+}1,x^3{+}1,x^3,0,x^3{+}x \\ x^2{+}x{+}1,x^3{+}x^2{+}x{+}1,x,1,x^2{+}1,x^2,x^3{+}x^2,x^2{+}x \end{pmatrix} \tag{5.6}$$

$$M^i_{8,8} = \mathrm{subfield}(M^i_{8,4}) \tag{5.7}$$

All these matrices improve over the previously known matrices, with the only exception being the involutory matrix from [SS16b] of dimension 4×4 over $\mathrm{GL}(4, \mathbb{F}_2)$. $M_{4,4}$ was found after enumerating a few thousand Hadamard matrices, while $M_{8,4}$ and $M^i_{8,4}$ are randomly generated and were found after a few seconds. The according linear straight-line programs with 36, 196, and 212 XORs, respectively, are given in Appendix D. Every best matrix over $\mathrm{GL}(8, \mathbb{F}_2)$ uses the subfield construction.

With these results we want to highlight that, when applying global optimizations, it is quite easy to improve (almost) all previous best-known results. We would like to mention that our results should not be misunderstood as an attempt to construct matrices, which cannot be improved. Another point that was not covered in this work is the depth of the critical path as considered in [BFP17]. This might well be a criterion for optimization in other scenarios.

5.5 Conclusion and Further Research

The results from this chapter constitute an important milestone in the theoretic research for efficient linear layers. We showed that the widespread technique of local optimization can easily be outperformed by global optimization. An example of the practical relevance of this work is the subsequent application of the PAAR1 algorithm in the design of the block cipher PYJAMASK [Gou+19] which is a submission to the current lightweight cryptography standardization process of NIST[1].

A different approach to find MDS matrices with a globally optimized implementation was presented by Duval and Leurent [DL18]. Instead of looking for an efficient implementation for a given MDS matrix, they are turning the tables. That is, they are searching through circuits with increasing XOR count until one of them corresponds to an MDS matrix. While this approach obviously cannot be used to improve implementations for existing matrices, the authors were able to improve some of the above presented results. Specifically, they improved our results (see Table 5.1) for 4×4 matrices over $\mathrm{GL}(4, \mathbb{F}_2)$ from 36 to 35 and for 4×4 matrices over $\mathrm{GL}(8, \mathbb{F}_2)$ from 72 to 67. Following our approach of global optimization, Banik et al. [BFI19] have recently improved many of our above presented results, including an implementation of the AES MixColumn matrix with 95 instead of 97 XORs. Improvements of the heuristics have also been presented in [TP19] together with a 94 XOR implementation of the AES MixColumn matrix. The currently best implementation in terms of the number of XORs for the AES MixColumn matrix was presented by Maximov [Max19] and needs 92 XORs.

While our work was solely motivated by the goal of small hardware circuits, also the depth of the implementation (i. e. the length of the circuit's critical path) is very important because it directly influences the latency. The above mentioned work of Duval and Leurent [DL18] already presented some trade-offs between the depth and the XOR count of an implementation . Afterward, Li et al. [Li+19] focused on this problem while concentrating on involutory matrices. They enhanced the BP algorithm with circuit-depth awareness and were able to find an MDS matrix with the same (minimal) depth as the AES MixColumn matrix while using only 88 XORs and being involutory. Furthermore, they improved our results (see Table 5.1) for 4×4 involutory matrices over $\mathrm{GL}(8, \mathbb{F}_2)$ from 84 to 78 while using the same depth.

While there has been very much research about optimizing the XOR count of linear layers, we should not forget that it is a simplified metric which is especially suitable for the easy comparison of research results. It is important to also ask the question of how much practical relevance this metric really has. In practice, the used hardware area also depends on the applied synthesis tools and the technology. The linear program will usually not be mapped one-to-one to a circuit. Instead, also XOR gates with more than two inputs are available which leads to more degrees of freedom. Duval and Leurent [DL18] pointed out that in particular modern FPGAs contain large lookup tables, so that a multi-input XOR gate is not much more expensive than a 2-input XOR gate. They expected this effect to be rather limited for ASICs. However, Banik et al. [BFI19] noted that in most standard cell libraries the area of a 3-input XOR gate is smaller than the area of two 2-input XOR gates. They also presented an according method to convert a circuit with only 2-input XOR gates into a smaller circuit with 2 and 3-input XOR gates.

The goal of minimizing the circuit area is of course not restricted to the linear layer. It is natural to also look at the other basic building blocks of an SPN cipher, i. e. S-boxes. Here, one cannot restrict to XOR gates because of the non-linearity. Instead, the typical metric is "gate equivalents", which basically just means that different gates are weighted differently according to the occupied area. While the details in non-linear circuit minimization are out of the scope of

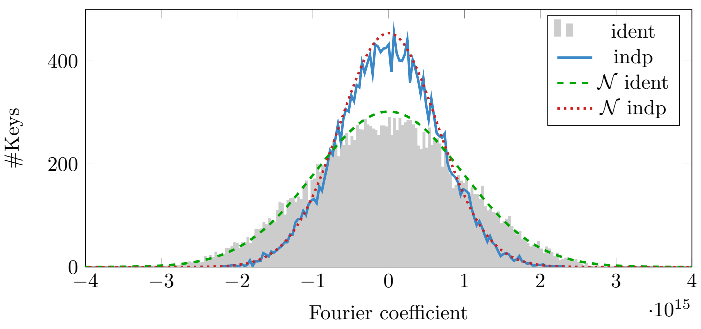
[1]https://csrc.nist.gov/projects/lightweight-cryptography

this thesis, we just want to mention that one of the most important constructions in this field is the so-called Canright S-box [Can05]. More recently, further important improvements have been published in [RTA18] and [ME19]. In the former work, Reyhani-Masoleh et al. [RTA18] also present several improvements for the above discussed BP algorithm.

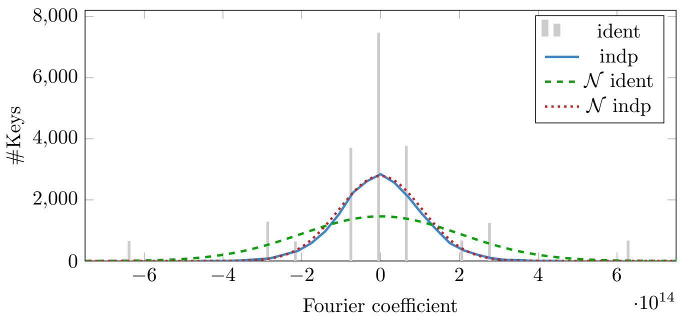
The efficiency of logical metrics for actual hardware implementation was studied by Raghuraman [Rag19]. Here, he also focused on non-linear circuits and did not explicitly investigate the relevance of the XOR count for practical implementations of linear layers. A comprehensive study on the practical relevance of the XOR count is still open work.

A

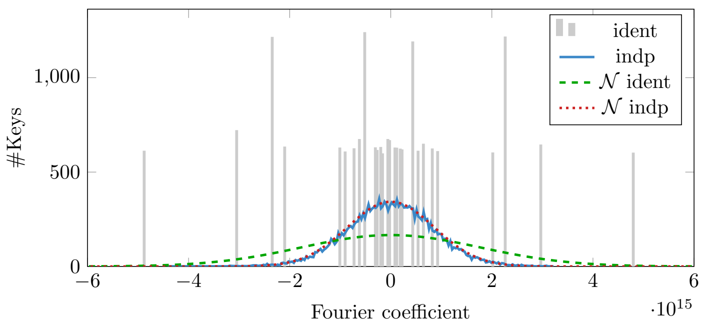
Plots for Serpent-type S-boxes



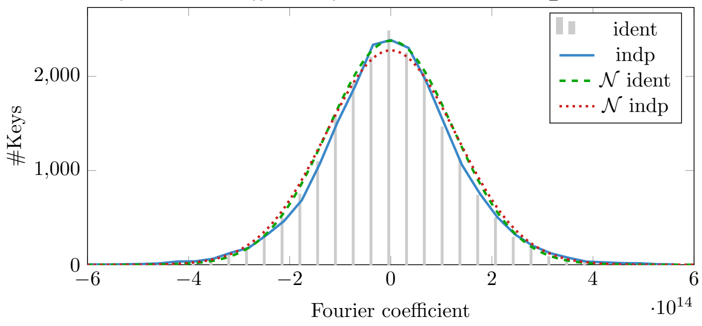
Distribution of Fourier coefficient for Present *with R_0 and 10 rounds.*



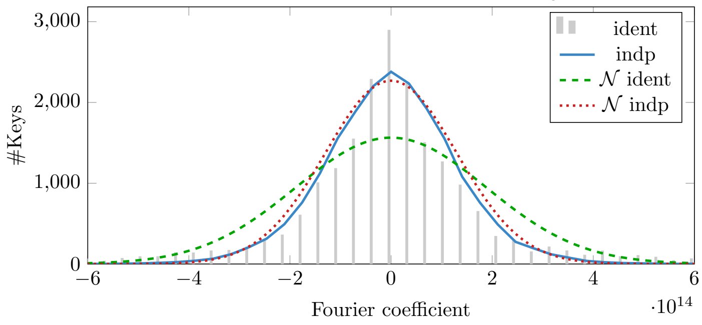
Distribution of Fourier coefficient for Present *with R_1 and 10 rounds.*



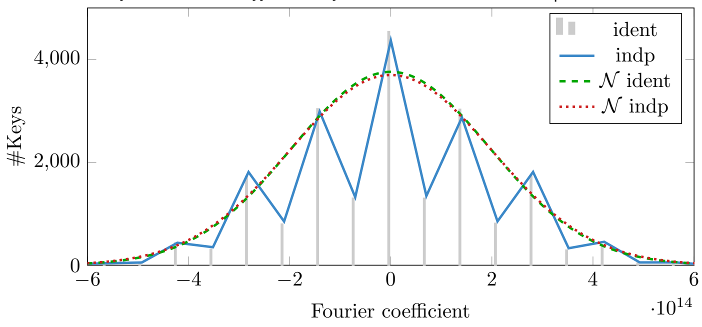
Distribution of Fourier coefficient for PRESENT *with R_2 and 10 rounds.*



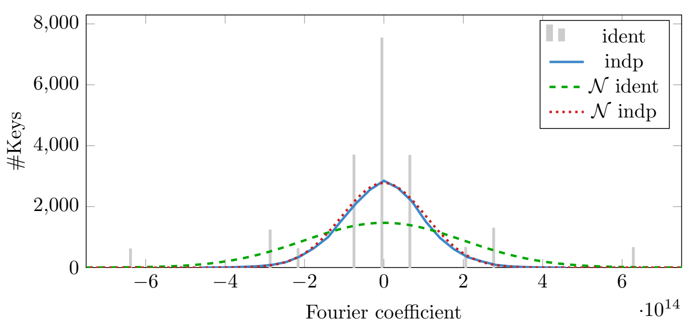
Distribution of Fourier coefficient for PRESENT *with R_3 and 10 rounds.*



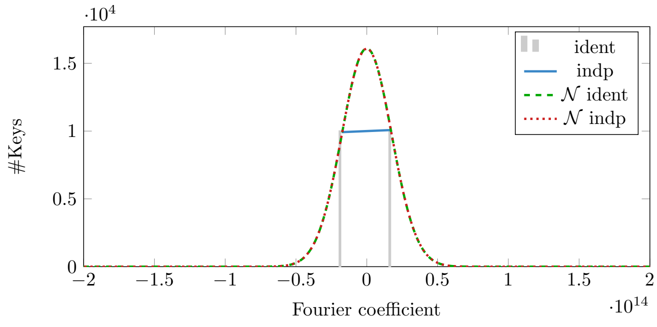
Distribution of Fourier coefficient for PRESENT *with R_4 and 10 rounds.*



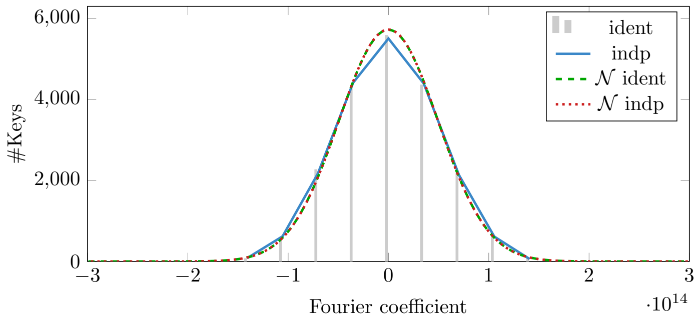
Distribution of Fourier coefficient for PRESENT *with R_5 and 10 rounds.*



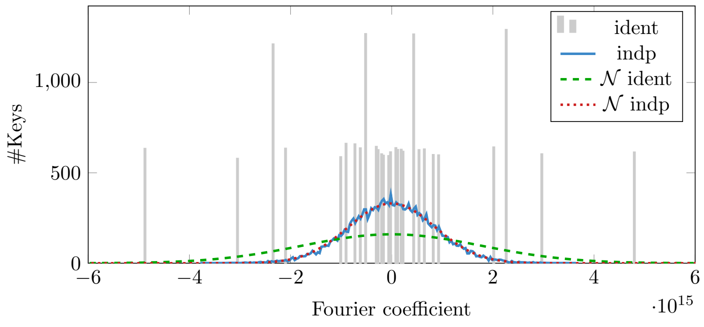
Distribution of Fourier coefficient for PRESENT *with* R_6 *and 10 rounds.*



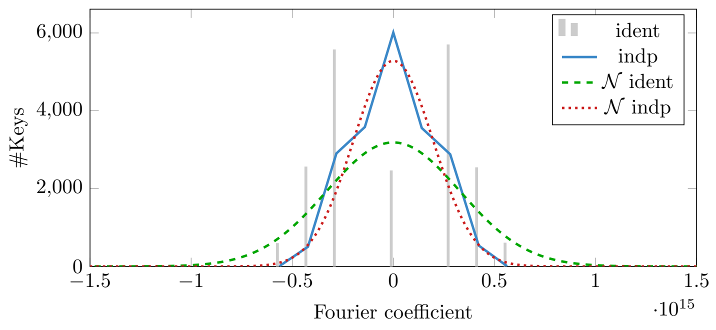
Distribution of Fourier coefficient for PRESENT *with* R_7 *and 10 rounds.*



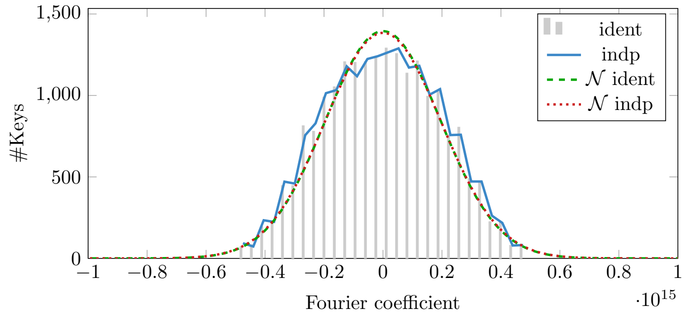
Distribution of Fourier coefficient for PRESENT *with* R_8 *and 10 rounds.*



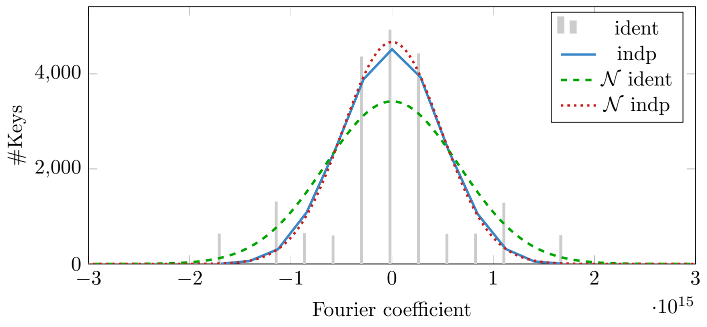
Distribution of Fourier coefficient for PRESENT *with* R_9 *and 10 rounds.*



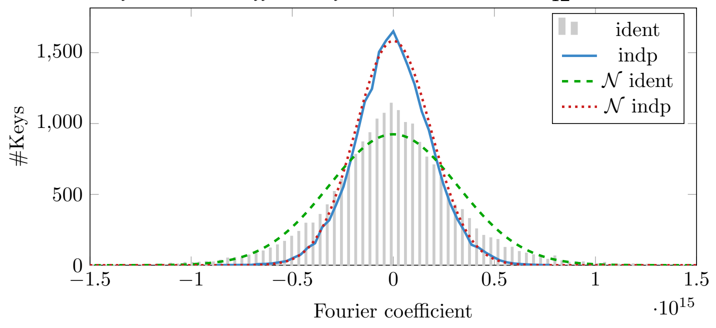
Distribution of Fourier coefficient for PRESENT *with R_{10} and 10 rounds.*



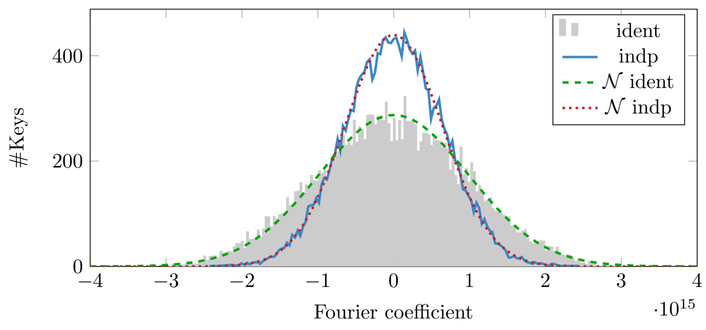
Distribution of Fourier coefficient for PRESENT *with R_{11} and 10 rounds.*



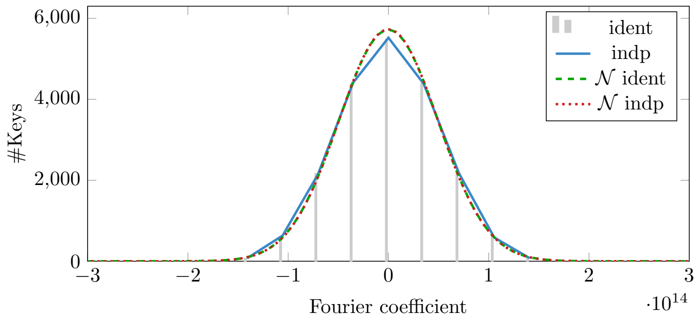
Distribution of Fourier coefficient for PRESENT *with R_{12} and 10 rounds.*



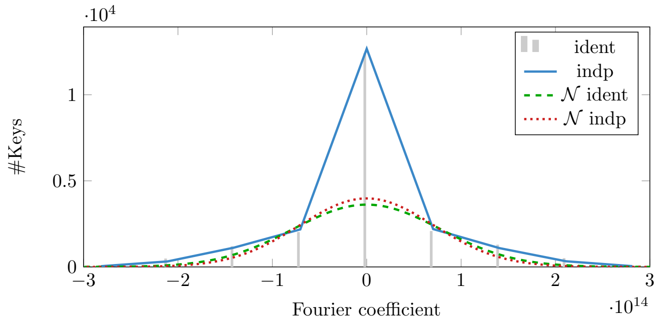
Distribution of Fourier coefficient for PRESENT *with R_{13} and 10 rounds.*



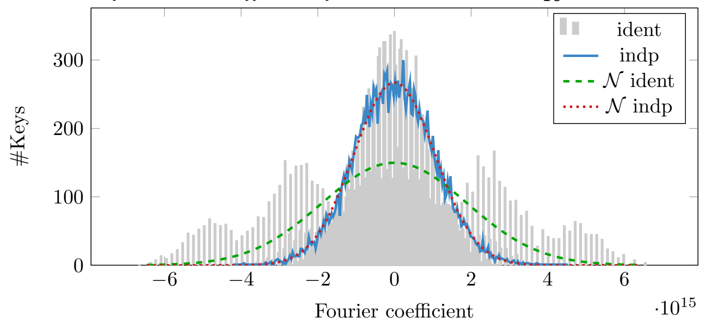
Distribution of Fourier coefficient for PRESENT *with R_{14} and 10 rounds.*



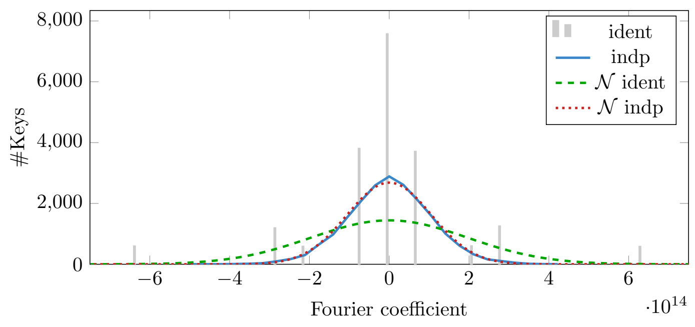
Distribution of Fourier coefficient for PRESENT *with R_{15} and 10 rounds.*



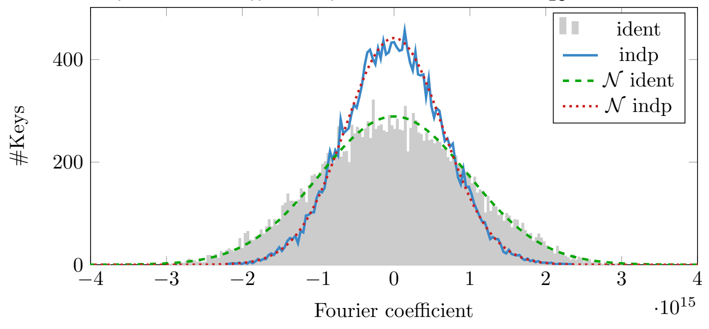
Distribution of Fourier coefficient for PRESENT *with R_{16} and 10 rounds.*



Distribution of Fourier coefficient for PRESENT *with R_{17} and 10 rounds.*



Distribution of Fourier coefficient for PRESENT *with* R_{18} *and 10 rounds.*



Distribution of Fourier coefficient for PRESENT *with* R_{19} *and 10 rounds.*

B
Elements with an XOR Count of 2

Table B.1: For each $m \leq 2048$ for which no irreducible trinomial of degree m exists, this table presents a matrix of the form $C_{x^m+1} + E_{i_1,j_1} + E_{i_2,j_2}$ with irreducible characteristic pentanomial. Such a matrix is represented as a 4-tuple (i_1, j_1, i_2, j_2). In all cases, the characteristic polynomial is equal to $\lambda^m + \lambda^{m+i_1-j_1+i_2-j_2-2} + \lambda^{m+i_1-j_1-1} + \lambda^{i_2-j_2-1} + 1$.

m		m		m		m		m		m		m		m		m		m	
8	(1,3,3,1)	237	(1,168,3,1)	451	(1,104,3,1)	659	(1,250,3,1)	869	(1,128,3,1)	1067	(1,960,5,1)	1274	(1,1176,3,1)	1480	(1,413,3,1)	1680	(1,645,3,1)	1867	(1,670,3,1)
13	(1,4,3,1)	240	(1,121,4,1)	452	(1,90,3,1)	661	(1,224,3,1)	872	(1,405,3,1)	1068	(1,54,3,1)	1275	(1,1265,3,1)	1483	(1,412,3,1)	1682	(1,5,3,1)	1868	(1,420,3,1)
16	(1,9,4,1)	243	(1,38,3,1)	453	(1,302,3,1)	664	(1,149,3,1)	874	(1,83,3,1)	1069	(1,338,3,1)	1277	(1,230,3,1)	1484	(1,41,3,1)	1683	(1,1278,3,1)	1869	(1,384,3,1)
19	(1,8,3,1)	245	(1,38,3,1)	454	(1,314,3,1)	666	(1,117,3,1)	875	(1,386,3,1)	1070	(1,228,3,1)	1280	(1,81,3,1)	1485	(1,1086,3,1)	1684	(1,730,3,1)	1872	(1,183,3,1)
24	(1,5,3,1)	246	(1,71,3,1)	456	(1,129,3,1)	667	(1,38,4,1)	877	(1,248,5,1)	1072	(1,789,4,1)	1283	(1,344,3,1)	1488	(1,1017,3,1)	1685	(1,816,3,1)	1874	(1,35,4,1)
26	(1,11,3,1)	248	(1,29,3,1)	459	(1,270,3,1)	669	(1,48,3,1)	878	(1,3,3,1)	1073	(1,362,3,1)	1285	(1,1174,3,1)	1491	(1,666,3,1)	1686	(1,72,3,1)	1875	(1,1386,3,1)
27	(1,3,3,1)	251	(1,24,3,1)	461	(1,170,3,1)	672	(1,567,3,1)	880	(1,11,3,1)	1074	(1,801,3,1)	1288	(1,379,3,1)	1493	(1,1002,3,1)	1688	(1,255,3,1)	1876	(1,1204,3,1)
32	(1,3,3,1)	254	(1,19,3,1)	464	(1,55,3,1)	674	(1,307,4,1)	883	(1,589,3,1)	1075	(1,142,3,1)	1290	(1,149,3,1)	1494	(1,620,3,1)	1690	(1,733,3,1)	1877	(1,628,3,1)
37	(1,16,4,1)	256	(1,157,3,1)	466	(1,451,3,1)	675	(1,225,3,1)	885	(1,512,3,1)	1076	(1,49,3,1)	1291	(1,302,3,1)	1496	(1,21,3,1)	1691	(1,26,3,1)	1880	(1,207,3,1)
38	(1,4,3,1)	259	(1,20,3,1)	467	(1,72,3,1)	677	(1,647,3,1)	886	(1,10,3,1)	1077	(1,706,4,1)	1292	(1,473,3,1)	1498	(1,223,3,1)	1693	(1,394,3,1)	1882	(1,399,4,1)
40	(1,14,3,1)	261	(1,12,3,1)	469	(1,188,3,1)	678	(1,312,3,1)	888	(1,501,3,1)	1080	(1,75,3,1)	1293	(1,212,3,1)	1499	(1,3,3,1)	1696	(1,19,3,1)	1883	(1,680,3,1)
43	(1,8,3,1)	262	(1,13,3,1)	472	(1,385,3,1)	680	(1,21,3,1)	891	(1,12,3,1)	1083	(1,92,3,1)	1296	(1,257,3,1)	1501	(1,1222,3,1)	1699	(1,404,3,1)	1885	(1,352,3,1)
45	(1,6,3,1)	264	(1,63,3,1)	475	(1,94,3,1)	681	(1,51,3,1)	893	(1,827,4,1)	1088	(1,3,3,1)	1299	(1,144,3,1)	1502	(1,5,3,1)	1701	(1,540,3,1)	1888	(1,905,3,1)
48	(1,21,3,1)	267	(1,182,3,1)	477	(1,286,4,1)	683	(1,104,3,1)	896	(1,87,3,1)	1091	(1,1026,3,1)	1301	(1,160,3,1)	1504	(1,559,3,1)	1702	(1,262,3,1)	1891	(1,280,3,1)
50	(1,7,3,1)	269	(1,64,3,1)	480	(1,273,5,1)	685	(1,172,3,1)	899	(1,64,3,1)	1093	(1,310,3,1)	1303	(1,380,3,1)	1506	(1,215,3,1)	1704	(1,1617,4,1)	1892	(1,440,3,1)
51	(1,12,3,1)	272	(1,165,3,1)	482	(1,115,3,1)	688	(1,149,3,1)	901	(1,504,4,1)	1096	(1,947,3,1)	1304	(1,391,3,1)	1507	(1,200,3,1)	1706	(1,843,3,1)	1893	(1,344,3,1)
53	(1,4,3,1)	275	(1,20,3,1)	483	(1,26,3,1)	691	(1,606,7,1)	904	(1,241,5,1)	1099	(1,644,3,1)	1307	(1,1200,3,1)	1509	(1,128,5,1)	1707	(1,150,3,1)	1894	(1,391,3,1)
56	(1,13,3,1)	277	(1,208,3,1)	485	(1,158,3,1)	693	(1,278,3,1)	907	(1,142,3,1)	1101	(1,474,3,1)	1309	(1,26,3,1)	1512	(1,381,3,1)	1709	(1,688,3,1)	1896	(1,1053,4,1)
59	(1,14,3,1)	280	(1,73,3,1)	488	(1,359,3,1)	696	(1,77,3,1)	909	(1,480,3,1)	1104	(1,515,3,1)	1312	(1,901,3,1)	1515	(1,14,3,1)	1712	(1,95,3,1)	1897	(1,80,3,1)
61	(1,4,3,1)	283	(1,154,3,1)	491	(1,477,3,1)	699	(1,360,3,1)	910	(1,8,3,1)	1107	(1,936,3,1)	1315	(1,508,3,1)	1517	(1,698,3,1)	1714	(1,1021,3,1)	1898	(1,241,3,1)
64	(1,61,3,1)	285	(1,158,3,1)	493	(1,20,3,1)	701	(1,238,3,1)	912	(1,627,3,1)	1109	(1,278,3,1)	1316	(1,204,3,1)	1520	(1,131,3,1)	1715	(1,250,3,1)	1899	(1,986,3,1)
67	(1,58,3,1)	288	(1,206,3,1)	496	(1,149,3,1)	703	(1,19,3,1)	914	(1,81,3,1)	1112	(1,35,3,1)	1317	(1,820,4,1)	1522	(1,985,3,1)	1717	(1,142,3,1)	1901	(1,230,3,1)
69	(1,42,4,1)	290	(1,96,3,1)	499	(1,40,3,1)	704	(1,195,5,1)	915	(1,320,3,1)	1114	(1,143,3,1)	1318	(1,109,3,1)	1523	(1,906,3,1)	1718	(1,242,3,1)	1904	(1,535,3,1)
70	(1,19,3,1)	291	(1,200,3,1)	501	(1,144,4,1)	706	(1,503,3,1)	917	(1,572,3,1)	1115	(1,328,3,1)	1320	(1,167,3,1)	1525	(1,602,3,1)	1720	(1,133,3,1)	1907	(1,780,3,1)
72	(1,15,5,1)	293	(1,16,3,1)	502	(1,245,3,1)	707	(1,376,3,1)	920	(1,535,3,1)	1117	(1,220,3,1)	1322	(1,405,3,1)	1528	(1,79,3,1)	1723	(1,322,3,1)	1909	(1,46,3,1)
75	(1,36,3,1)	296	(1,109,3,1)	504	(1,141,3,1)	709	(1,230,3,1)	922	(1,299,4,1)	1118	(1,168,3,1)	1323	(1,272,3,1)	1531	(1,910,3,1)	1725	(1,90,3,1)	1910	(1,266,3,1)
77	(1,47,3,1)	298	(1,55,3,1)	507	(1,429,6,1)	710	(1,17,3,1)	923	(1,52,4,1)	1120	(1,1043,3,1)	1325	(1,22,3,1)	1532	(1,66,3,1)	1727	(1,207,3,1)	1912	(1,157,3,1)
78	(1,23,3,1)	299	(1,116,3,1)	509	(1,10,3,1)	712	(1,455,3,1)	925	(1,859,3,1)	1123	(1,410,3,1)	1328	(1,917,3,1)	1533	(1,1160,3,1)	1728	(1,1227,3,1)	1914	(1,639,4,1)
80	(1,5,3,1)	301	(1,154,3,1)	512	(1,425,3,1)	715	(1,110,3,1)	928	(1,537,4,1)	1124	(1,424,3,1)	1330	(1,187,3,1)	1536	(1,39,3,1)	1730	(1,443,3,1)	1915	(1,40,3,1)
82	(1,47,5,1)	304	(1,109,3,1)	515	(1,236,3,1)	717	(1,448,4,1)	929	(1,51,3,1)	1125	(1,254,3,1)	1331	(1,978,3,1)	1538	(1,509,3,1)	1731	(1,338,3,1)	1916	(1,663,3,1)
83	(1,4,3,1)	306	(1,81,3,1)	517	(1,172,3,1)	720	(1,369,4,1)	931	(1,544,3,1)	1128	(1,21,3,1)	1333	(1,530,3,1)	1539	(1,30,3,1)	1732	(1,1250,3,1)	1917	(1,1026,3,1)
85	(1,16,3,1)	307	(1,192,4,1)	520	(1,331,3,1)	723	(1,414,3,1)	933	(1,264,3,1)	1131	(1,552,3,1)	1336	(1,97,3,1)	1541	(1,698,3,1)	1733	(1,128,3,1)	1920	(1,39,3,1)
88	(1,11,3,1)	309	(1,155,3,1)	523	(1,140,3,1)	725	(1,68,3,1)	934	(1,424,3,1)	1132	(1,52,3,1)	1339	(1,820,3,1)	1544	(1,411,3,1)	1736	(1,493,3,1)	1922	(1,1497,3,1)
91	(1,8,3,1)	311	(1,25,3,1)	525	(1,328,4,1)	728	(1,393,3,1)	936	(1,23,3,1)	1133	(1,238,3,1)	1341	(1,732,3,1)	1546	(1,351,4,1)	1739	(1,420,3,1)	1923	(1,326,3,1)
96	(1,9,3,1)	312	(1,7,5,1)	528	(1,35,3,1)	731	(1,80,3,1)	939	(1,288,3,1)	1136	(1,957,3,1)	1342	(1,610,3,1)	1547	(1,62,3,1)	1741	(1,476,3,1)	1925	(1,210,3,1)
99	(1,78,3,1)	315	(1,50,3,1)	530	(1,17,3,1)	733	(1,310,3,1)	940	(1,8,3,1)	1139	(1,246,3,1)	1344	(1,185,3,1)	1549	(1,1220,3,1)	1744	(1,1183,3,1)	1928	(1,83,3,1)
101	(1,20,3,1)	317	(1,90,3,1)	531	(1,44,3,1)	734	(1,56,3,1)	941	(1,382,3,1)	1141	(1,196,3,1)	1346	(1,57,3,1)	1552	(1,325,3,1)	1747	(1,1402,3,1)	1930	(1,569,3,1)
104	(1,94,3,1)	320	(1,53,3,1)	533	(1,56,3,1)	736	(1,271,5,1)	944	(1,191,3,1)	1143	(1,191,3,1)	1347	(1,18,3,1)	1555	(1,404,3,1)	1749	(1,56,3,1)	1931	(1,3,3,1)
107	(1,44,4,1)	323	(1,120,3,1)	535	(1,104,3,1)	739	(1,286,3,1)	946	(1,469,3,1)	1144	(1,335,3,1)	1349	(1,138,5,1)	1557	(1,822,3,1)	1752	(1,1585,4,1)	1933	(1,940,3,1)
109	(1,20,3,1)	325	(1,244,3,1)	536	(1,117,3,1)	741	(1,18,3,1)	947	(1,16,3,1)	1147	(1,598,3,1)	1352	(1,595,3,1)	1560	(1,1027,5,1)	1754	(1,103,3,1)	1936	(1,1657,3,1)
112	(1,91,3,1)	326	(1,27,3,1)	539	(1,3,3,1)	744	(1,635,3,1)	949	(1,550,4,1)	1149	(1,902,5,1)	1355	(1,216,3,1)	1563	(1,198,3,1)	1755	(1,308,3,1)	1939	(1,110,3,1)
114	(1,33,3,1)	328	(1,237,4,1)	541	(1,364,5,1)	747	(1,548,3,1)	950	(1,154,3,1)	1150	(1,130,3,1)	1357	(1,302,5,1)	1565	(1,66,3,1)	1757	(1,988,3,1)	1941	(1,1262,3,1)
115	(1,32,3,1)	331	(1,8,3,1)	542	(1,50,3,1)	749	(1,316,3,1)	952	(1,573,4,1)	1152	(1,11,3,1)	1360	(1,485,3,1)	1568	(1,231,3,1)	1758	(1,213,3,1)	1942	(1,76,3,1)
116	(1,20,3,1)	334	(1,64,3,1)	544	(1,65,5,1)	752	(1,117,3,1)	955	(1,80,3,1)	1155	(1,8,3,1)	1363	(1,1018,3,1)	1571	(1,430,3,1)	1760	(1,273,3,1)	1944	(1,429,3,1)
117	(1,99,4,1)	335	(1,20,3,1)	546	(1,213,3,1)	755	(1,230,3,1)	957	(1,372,3,1)	1157	(1,328,3,1)	1365	(1,488,3,1)	1573	(1,112,3,1)	1761	(1,186,3,1)	1947	(1,210,3,1)
120	(1,15,3,1)	336	(1,35,3,1)	547	(1,302,4,1)	757	(1,751,3,1)	958	(1,80,3,1)	1160	(1,87,3,1)	1368	(1,533,3,1)	1574	(1,284,3,1)	1762	(1,277,3,1)	1949	(1,686,3,1)
122	(1,60,3,1)	338	(1,15,3,1)	548	(1,18,3,1)	760	(1,187,3,1)	960	(1,765,3,1)	1162	(1,451,3,1)	1370	(1,35,3,1)	1576	(1,131,3,1)	1763	(1,266,3,1)	1952	(1,205,3,1)
125	(1,18,3,1)	339	(1,20,3,1)	549	(1,304,4,1)	763	(1,334,3,1)	962	(1,213,3,1)	1163	(1,162,3,1)	1371	(1,528,3,1)	1579	(1,706,3,1)	1765	(1,458,3,1)	1954	(1,71,4,1)
128	(1,61,3,1)	341	(1,278,3,1)	552	(1,195,3,1)	764	(1,40,3,1)	963	(1,38,3,1)	1165	(1,32,3,1)	1373	(1,154,3,1)	1581	(1,1446,3,1)	1766	(1,453,3,1)	1955	(1,1256,4,1)
131	(1,36,3,1)	344	(1,85,3,1)	554	(1,191,3,1)	765	(1,216,3,1)	965	(1,88,3,1)	1168	(1,629,3,1)	1376	(1,151,3,1)	1584	(1,621,3,1)	1768	(1,157,3,1)	1957	(1,1486,3,1)
133	(1,28,3,1)	347	(1,10,3,1)	555	(1,10,4,1)	766	(1,161,3,1)	968	(1,319,3,1)	1171	(1,466,3,1)	1378	(1,611,4,1)	1587	(1,250,7,1)	1771	(1,518,3,1)	1960	(1,31,3,1)
136	(1,11,3,1)	349	(1,58,3,1)	557	(1,318,3,1)	768	(1,117,3,1)	970	(1,415,3,1)	1172	(1,288,3,1)	1379	(1,306,3,1)	1589	(1,128,7,1)	1773	(1,1352,3,1)	1963	(1,1504,3,1)
138	(1,53,3,1)	352	(1,125,3,1)	560	(1,125,3,1)	770	(1,547,3,1)	971	(1,346,3,1)	1173	(1,108,3,1)	1381	(1,1276,3,1)	1592	(1,1135,3,1)	1776	(1,669,3,1)	1965	(1,732,3,1)
139	(1,8,5,1)	355	(1,164,3,1)	562	(1,25,3,1)	771	(1,138,3,1)	973	(1,56,3,1)	1176	(1,57,3,1)	1382	(1,270,3,1)	1594	(1,821,3,1)	1779	(1,1046,3,1)	1968	(1,9,3,1)
141	(1,47,3,1)	356	(1,10,3,1)	563	(1,60,3,1)	773	(1,138,3,1)	974	(1,30,3,1)	1179	(1,56,3,1)	1384	(1,221,3,1)	1595	(1,192,3,1)	1781	(1,36,3,1)	1970	(1,649,3,1)
143	(1,19,3,1)	357	(1,266,3,1)	565	(1,178,3,1)	776	(1,569,3,1)	976	(1,715,3,1)	1181	(1,650,4,1)	1387	(1,140,3,1)	1597	(1,1150,3,1)	1784	(1,1481,3,1)	1971	(1,6,3,1)
144	(1,39,3,1)	360	(1,5,3,1)	568	(1,373,3,1)	779	(1,338,3,1)	978	(1,311,3,1)	1184	(1,701,3,1)	1389	(1,204,3,1)	1598	(1,289,3,1)	1786	(1,1429,3,1)	1972	(1,820,3,1)
149	(1,40,3,1)	361	(1,25,3,1)	571	(1,10,4,1)	781	(1,278,3,1)	980	(1,369,3,1)	1187	(1,530,3,1)	1392	(1,1283,3,1)	1600	(1,529,3,1)	1787	(1,112,4,1)	1973	(1,1918,3,1)
152	(1,3,3,1)	363	(1,258,3,1)	572	(1,35,3,1)	784	(1,169,3,1)	981	(1,42,3,1)	1189	(1,136,3,1)	1394	(1,371,3,1)	1603	(1,134,3,1)	1789	(1,578,3,1)	1976	(1,361,3,1)
157	(1,50,3,1)	365	(1,294,3,1)	573	(1,6,3,1)	786	(1,387,3,1)	984	(1,917,5,1)	1192	(1,397,3,1)	1395	(1,380,3,1)	1605	(1,1328,5,1)	1792	(1,1001,4,1)	1978	(1,289,3,1)
158	(1,12,3,1)	368	(1,25,3,1)	576	(1,41,3,1)	787	(1,52,3,1)	987	(1,308,3,1)	1194	(1,153,3,1)	1397	(1,128,3,1)	1608	(1,273,3,1)	1794	(1,27,4,1)	1979	(1,1190,4,1)
160	(1,91,3,1)	371	(1,20,3,1)	578	(1,507,3,1)	788	(1,26,3,1)	989	(1,138,3,1)	1195	(1,962,3,1)	1400	(1,489,3,1)	1610	(1,561,3,1)	1795	(1,130,3,1)	1981	(1,1930,3,1)
163	(1,104,3,1)	373	(1,34,3,1)	579	(1,114,3,1)	789	(1,294,3,1)	992	(1,33,3,1)	1197	(1,12,3,1)	1403	(1,3,3,1)	1611	(1,506,3,1)	1796	(1,48,3,1)	1982	(1,260,3,1)
164	(1,18,3,1)	374	(1,38,3,1)	581	(1,318,3,1)	790	(1,269,3,1)	995	(1,558,5,1)	1200	(1,179,3,1)	1405	(1,658,3,1)	1613	(1,892,4,1)	1797	(1,8,3,1)	1984	(1,739,3,1)
165	(1,135,3,1)	376	(1,113,3,1)	584	(1,63,3,1)	792	(1,131,3,1)	997	(1,76,3,1)	1203	(1,788,3,1)	1406	(1,97,3,1)	1614	(1,138,3,1)	1800	(1,819,3,1)	1987	(1,1196,3,1)
168	(1,21,3,1)	379	(1,8,3,1)	586	(1,469,3,1)	795	(1,450,3,1)	1000	(1,101,3,1)	1205	(1,590,5,1)	1408	(1,817,3,1)	1616	(1,801,3,1)	1803	(1,1158,3,1)	1989	(1,232,5,1)
171	(1,60,3,1)	381	(1,198,4,1)	587	(1,256,3,1)	796	(1,26,3,1)	1002	(1,225,3,1)	1208	(1,319,3,1)	1411	(1,634,3,1)	1619	(1,578,3,1)	1805	(1,1312,3,1)	1992	(1,177,3,1)
173	(1,32,3,1)	384	(1,195,3,1)	589	(1,70,3,1)	797	(1,336,3,1)	1003	(1,716,3,1)	1211	(1,686,3,1)	1413	(1,880,4,1)	1621	(1,852,4,1)	1808	(1,545,3,1)	1995	(1,710,3,1)
176	(1,19,3,1)	387	(1,102,3,1)	591	(1,95,3,1)	800	(1,523,3,1)	1004	(1,106,3,1)	1213	(1,664,3,1)	1416	(1,461,3,1)	1622	(1,158,3,1)	1811	(1,618,3,1)	1997	(1,366,3,1)
179	(1,10,3,1)	389	(1,38,3,1)	592	(1,241,3,1)	802	(1,47,4,1)	1005	(1,692,3,1)	1216	(1,709,3,1)	1418	(1,149,3,1)	1624	(1,1375,3,1)	1812	(1,702,3,1)	1998	(1,155,3,1)
181	(1,175,3,1)	392	(1,47,3,1)	595	(1,364,3,1)	803	(1,60,3,1)	1006	(1,40,3,1)	1219	(1,676,4,1)	1419	(1,324,3,1)	1626	(1,573,3,1)	1813	(1,14,3,1)	2000	(1,333,3,1)
184	(1,83,3,1)	395	(1,126,3,1)	597	(1,540,3,1)	805	(1,248,3,1)	1008	(1,521,3,1)	1221	(1,200,3,1)	1421	(1,496,3,1)	1627	(1,130,3,1)	1816	(1,83,3,1)	2002	(1,5,3,1)
187	(1,22,3,1)	397	(1,208,3,1)	598	(1,248,3,1)	808	(1,379,3,1)	1011	(1,330,3,1)	1222	(1,181,3,1)	1424	(1,323,3,1)	1629	(1,500,3,1)	1819	(1,926,3,1)	2003	(1,710,3,1)
188	(1,21,3,1)	398	(1,104,3,1)	600	(1,87,3,1)	811	(1,394,3,1)	1013	(1,296,3,1)	1224	(1,545,3,1)	1427	(1,1110,3,1)	1632	(1,791,3,1)	1821	(1,668,3,1)	2005	(1,1074,4,1)
189	(1,63,3,1)	400	(1,283,3,1)	603	(1,453,3,1)	813	(1,98,3,1)	1016	(1,131,3,1)	1227	(1,212,3,1)	1429	(1,400,3,1)	1635	(1,545,3,1)	1822	(1,856,3,1)	2008	(1,445,3,1)
190	(1,16,3,1)	403	(1,124,3,1)	605	(1,248,3,1)	816	(1,285,3,1)	1017	(1,261,3,1)	1229	(1,40,3,1)	1432	(1,1241,3,1)	1637	(1,90,3,1)	1824	(1,1125,4,1)	2011	(1,922,3,1)
192	(1,155,3,1)	405	(1,68,5,1)	608	(1,73,3,1)	819	(1,496,4,1)	1018	(1,937,3,1)	1232	(1,477,3,1)	1435	(1,484,3,1)	1640	(1,401,3,1)	1826	(1,335,3,1)	2012	(1,40,3,1)
195	(1,158,3,1)	408	(1,27,3,1)	611	(1,334,3,1)	821	(1,424,3,1)	1019	(1,3,3,1)	1235	(1,50,3,1)	1437	(1,380,3,1)	1643	(1,3,3,1)	1827	(1,1308,3,1)	2013	(1,108,3,1)
197	(1,126,3,1)	410	(1,117,3,1)	613	(1,368,4,1)	824	(1,233,3,1)	1021	(1,832,3,1)	1237	(1,380,3,1)	1439	(1,300,3,1)	1644	(1,221,3,1)	1829	(1,760,3,1)	2014	(1,170,3,1)
200	(1,53,3,1)	411	(1,108,7,1)	616	(1,597,4,1)	827	(1,680,3,1)	1024	(1,643,3,1)	1240	(1,826,3,1)	1440	(1,901,4,1)	1645	(1,176,3,1)	1832	(1,295,3,1)	2016	(1,645,3,1)
203	(1,3,3,1)	413	(1,36,3,1)	619	(1,118,3,1)	829	(1,560,3,1)	1027	(1,830,3,1)	1243	(1,34,3,1)	1443	(1,348,3,1)	1646	(1,68,3,1)	1834	(1,1820,3,1)	2018	(1,173,3,1)
205	(1,176,3,1)	416	(1,87,3,1)	621	(1,344,3,1)	830	(1,60,3,1)	1032	(1,541,4,1)	1244	(1,46,3,1)	1445	(1,68,3,1)	1648	(1,1249,3,1)	1835	(1,1352,3,1)	2019	(1,1506,3,1)
206	(1,7,3,1)	419	(1,40,5,1)	624	(1,609,3,1)	832	(1,685,3,1)	1035	(1,208,5,1)	1245	(1,360,3,1)	1448	(1,93,3,1)	1651	(1,490,3,1)	1837	(1,1162,4,1)	2021	(1,102,3,1)
208	(1,197,4,1)	421	(1,20,7,1)	627	(1,377,3,1)	835	(1,92,3,1)	1037	(1,758,3,1)	1248	(1,327,3,1)	1450	(1,1183,3,1)	1653	(1,834,3,1)	1840	(1,409,3,1)	2024	(1,1949,4,1)
211	(1,112,3,1)	424	(1,359,3,1)	629	(1,268,3,1)	836	(1,8,3,1)	1038	(1,708,3,1)	1250	(1,313,3,1)	1451	(1,22,3,1)	1654	(1,1000,3,1)	1842	(1,303,3,1)	2027	(1,1980,3,1)
213	(1,24,3,1)	427	(1,22,3,1)	630	(1,80,3,1)	837	(1,542,3,1)	1040	(1,311,3,1)	1251	(1,38,3,1)	1453	(1,146,3,1)	1656	(1,179,3,1)	1843	(1,904,3,1)	2029	(1,4,3,1)
216	(1,11,3,1)	429	(1,18,3,1)	632	(1,117,3,1)	840	(1,173,3,1)	1043	(1,304,3,1)	1253	(1,318,3,1)	1456	(1,29,3,1)	1658	(1,801,3,1)	1845	(1,786,3,1)	2030	(1,337,3,1)
219	(1,201,3,1)	430	(1,49,3,1)	635	(1,60,3,1)	843	(1,200,3,1)	1045	(1,298,3,1)	1254	(1,56,3,1)	1459	(1,496,3,1)	1659	(1,446,3,1)	1848	(1,1077,3,1)	2032	(1,1479,5,1)
221	(1,80,3,1)	432	(1,117,5,1)	637	(1,38,6,1)	848	(1,243,3,1)	1046	(1,38,3,1)	1256	(1,605,3,1)	1461	(1,122,3,1)	1661	(1,156,3,1)	1850	(1,1015,4,1)	2035	(1,442,3,1)
222	(1,71,3,1)	434	(1,119,3,1)	638	(1,3,3,1)	851	(1,58,3,1)	1048	(1,539,3,1)	1258	(1,689,3,1)	1462	(1,913,3,1)	1662	(1,285,3,1)	1851	(1,720,3,1)	2037	(1,210,3,1)
224	(1,83,3,1)	435	(1,122,3,1)	640	(1,35,3,1)	853	(1,142,3,1)	1051	(1,12,4,1)	1259	(1,168,3,1)	1464	(1,1155,3,1)	1664	(1,135,3,1)	1852	(1,46,3,1)	2038	(1,1498,3,1)
226	(1,169,3,1)	437	(1,318,3,1)	643	(1,412,4,1)	854	(1,138,3,1)	1053	(1,1044,3,1)	1261	(1,640,3,1)	1467	(1,782,3,1)	1667	(1,1146,3,1)	1853	(1,906,3,1)	2040	(1,77,3,1)
227	(1,182,3,1)	440	(1,297,3,1)	644	(1,236,3,1)	856	(1,691,3,1)	1056	(1,891,3,1)	1262	(1,285,3,1)	1469	(1,164,4,1)	1669	(1,106,3,1)	1856	(1,861,3,1)	2042	(1,417,3,1)
229	(1,80,3,1)	442	(1,187,4,1)	645	(1,50,3,1)	859	(1,178,3,1)	1059	(1,260,3,1)	1264	(1,283,3,1)	1472	(1,83,3,1)	1670	(1,472,3,1)	1858	(1,1099,3,1)	2043	(1,398,3,1)
230	(1,3,3,1)	443	(1,48,3,1)	648	(1,626,3,1)	863	(1,92,3,1)	1061	(1,268,3,1)	1267	(1,800,4,1)	1474	(1,829,3,1)	1672	(1,881,4,1)	1859	(1,4,3,1)	2045	(1,1554,3,1)
232	(1,133,3,1)	445	(1,88,3,1)	653	(1,478,3,1)	864	(1,39,3,1)	1064	(1,351,3,1)	1269	(1,907,4,1)	1475	(1,68,3,1)	1675	(1,970,4,1)	1861	(1,398,3,1)	2046	(1,240,3,1)
235	(1,26,3,1)	448	(1,119,3,1)	656	(1,295,3,1)	867	(1,48,3,1)	1066	(1,401,3,1)	1272	(1,383,3,1)	1477	(1,140,3,1)	1677	(1,852,3,1)	1864	(1,61,3,1)	2048	(1,83,3,1)

C

XOR Count Correlation Figures

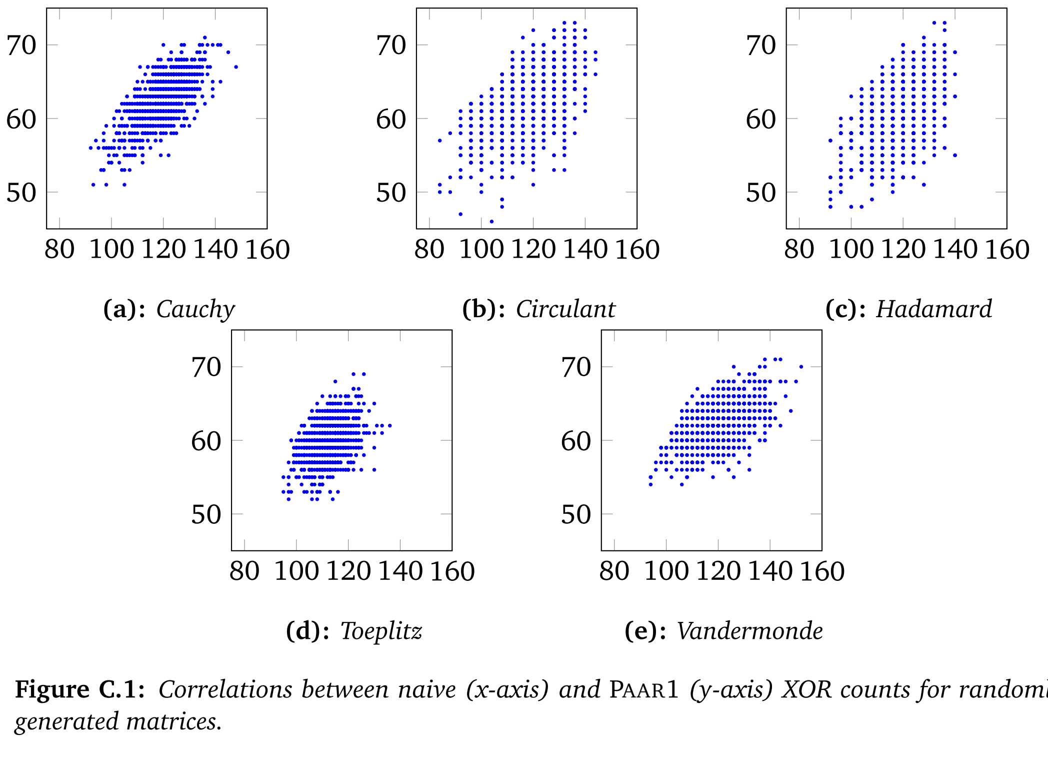
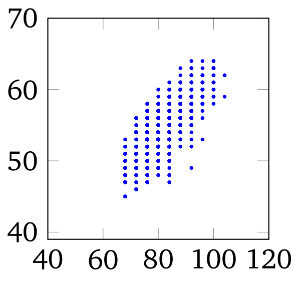
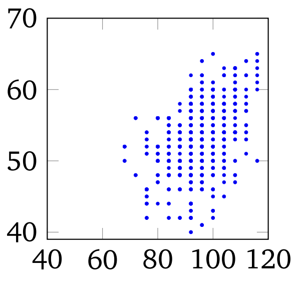


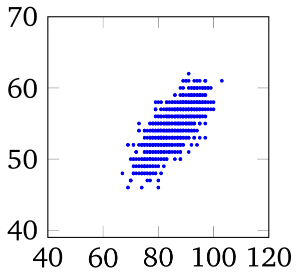
Figure C.1: *Correlations between naive (x-axis) and* Paar1 *(y-axis) XOR counts for randomly generated matrices.*



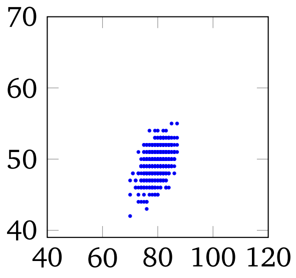
(a): *Circulant (enum)*



(b): *Hadamard (enum)*

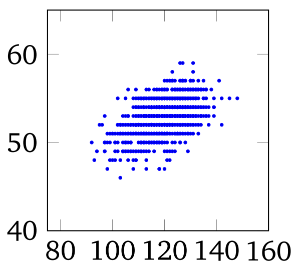


(c): *Toeplitz (enum)*

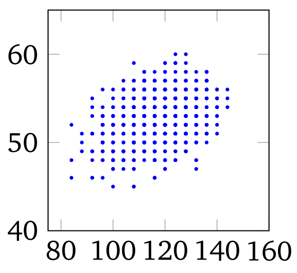


(d): *Arbitrary (enum)*

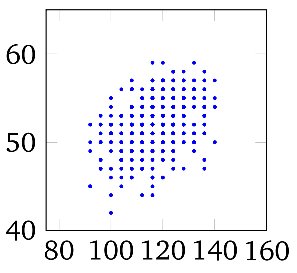
Figure C.2: *Correlations between naive (x-axis) and* PAAR1 *(y-axis) XOR counts for enumerated matrices.*



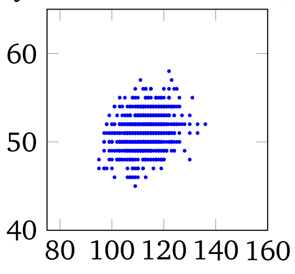
(a): *Cauchy*



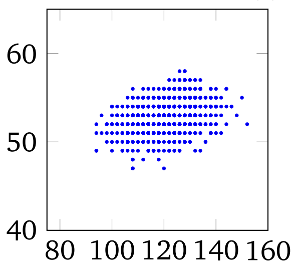
(b): *Circulant*



(c): *Hadamard*

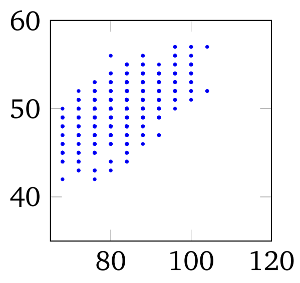


(d): *Toeplitz*

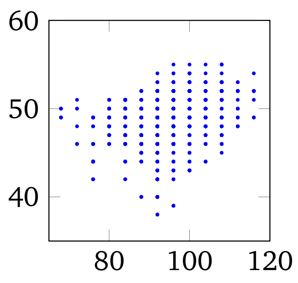


(e): *Vandermonde*

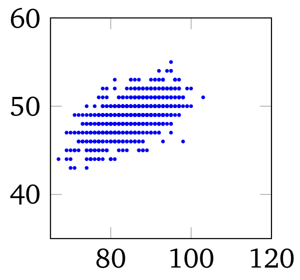
Figure C.3: *Correlations between naive (x-axis) and* BP *(y-axis) XOR counts for randomly generated matrices.*



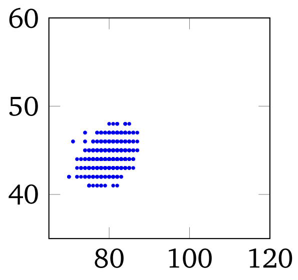
(a): *Circulant (enum)*



(b): *Hadamard (enum)*



(c): *Toeplitz (enum)*



(d): *Arbitrary (enum)*

Figure C.4: *Correlations between naive (x-axis) and* BP *(y-axis) XOR counts for enumerated matrices.*

D

Linear Straight-Line Programs for MDS Matrices

In Table 5.1, we presented the XOR counts for our new best results. The according matrices were given in Section 5.4.3, see Eqs. (5.1) to (5.7). In the following, the according linear straight-line programs are presented in Listings D.1 to D.3. Here, we restrict to $M_{4,4}, M_{8,4}, M^{i}_{8,4}$ (see Eqs. (5.1), (5.3) and (5.6)) as the other matrices are subfield constructions, i. e. the implementations can be derived straight forward from the underlying implementations. In the linear straight-line programs, x, y, t denote input, output, and temporary values, respectively.

Listing D.1: Straight-line program for the matrix $M_{4,4}$ from Eq. (5.1) that uses 36 XORs.

```
t0 = x0 + x5
t1 = x1 + x11
t2 = x8 + x13
t3 = x3 + x9
t4 = x7 + t2
y11 = x2 + t4
t6 = x12 + t3
y15 = x6 + t6
t8 = x15 + t0
y3 = x10 + t8
t10 = x4 + t1
y7 = x14 + t10
t12 = x3 + y7
y8 = t8 + t12
t14 = x11 + y15
y0 = t4 + t14
t16 = x7 + y3
y12 = t10 + t16
t18 = x15 + y11
y4 = t6 + t18
t20 = x0 + x13
y5 = t14 + t20
t22 = x0 + t6
y2 = y8 + t22
t24 = x1 + x12
y9 = t16 + t24
t26 = x4 + x9
y1 = t18 + t26
t28 = x4 + t4
y6 = y12 + t28
t30 = x5 + x8
y13 = t12 + t30
t32 = x8 + t10
```

```
y10 = y0 + t32
t34 = x12 + t8
y14 = y4 + t34
```

Listing D.2: Straight-line program for the matrix $M_{8,4}$ from Eq. (5.3) that uses 196 XORs.

```
t0 = x2 + x15
t1 = x3 + x14
t2 = x6 + x19
t3 = x7 + x18
t4 = x10 + x23
t5 = x11 + x22
t6 = x26 + x31
t7 = x27 + x30
t8 = x0 + x20
t9 = x4 + x28
t10 = x8 + x12
t11 = x16 + x24
t12 = x1 + x21
t13 = x5 + x29
t14 = x9 + x13
t15 = x17 + x25
t16 = t0 + t13
t17 = t1 + t15
t18 = t2 + t12
t19 = t3 + t14
t20 = t4 + t9
t21 = t5 + t11
t22 = t6 + t8
t23 = t7 + t10
t24 = t16 + t21
t25 = t17 + t20
t26 = t18 + t23
t27 = t19 + t22
t28 = x1 + x31
t29 = x3 + x29
t30 = x5 + x23
t31 = x7 + x21
t32 = x8 + t5
t33 = x9 + x19
t34 = x10 + t9
t35 = x11 + x17
t36 = x13 + x27
t37 = x15 + x25
t38 = x20 + t4
t39 = x22 + t11
t40 = x24 + t7
t41 = x26 + t8
t42 = x28 + t6
```

```
t43 = x30 + t10
t44 = x0 + t1
t45 = x0 + t26
t46 = x2 + x19
t47 = x2 + x29
t48 = x3 + x18
t49 = x4 + t3
t50 = x4 + t24
t51 = x6 + x15
t52 = x6 + x21
t53 = x7 + x14
t54 = x8 + t27
t55 = x9 + x18
t56 = x11 + t43
t57 = x12 + t0
t58 = x12 + t27
t59 = x14 + x25
t60 = x16 + t2
t61 = x16 + t25
t62 = x20 + t26
t63 = x23 + t41
t64 = x24 + t25
t65 = x27 + t39
t66 = x28 + t24
t67 = x31 + t34
t68 = t0 + t42
t69 = t1 + t40
t70 = t2 + t38
t71 = t3 + t32
t72 = t4 + t29
t73 = t5 + t37
t74 = t6 + t31
t75 = t7 + t33
t76 = t16 + t35
t77 = t17 + t30
t78 = t18 + t36
t79 = t19 + t28
y0 = t10 + t14 + t31 + t65 + t68
y1 = x29 + t57 + t64 + t74
y2 = x13 + t13 + t40 + t41 + t48 + t73
y3 = x23 + t6 + t59 + t62
y4 = t11 + t15 + t29 + t56 + t70
y5 = x21 + t54 + t60 + t72
y6 = x17 + t12 + t32 + t34 + t53 + t75
y7 = x31 + t4 + t55 + t66
y8 = x4 + t10 + t21 + t46 + t79
y9 = x5 + t28 + t61 + t71
y10 = t55 + t60 + t63 + t76
y11 = x3 + x10 + x17 + t3 + t45
```

```
y12 = x16 + t8 + t20 + t53 + t78
y13 = x17 + t36 + t50 + t70
y14 = t49 + t52 + t56 + t77
y15 = x5 + x15 + x22 + t2 + t58
y16 = t9 + t13 + t37 + t63 + t71
y17 = x9 + t49 + t62 + t73
y18 = x5 + t14 + t38 + t39 + t46 + t74
y19 = x27 + t5 + t52 + t64
y20 = x0 + t11 + t23 + t51 + t77
y21 = x1 + t30 + t58 + t69
y22 = t57 + t59 + t67 + t78
y23 = x7 + x13 + x26 + t1 + t50
y24 = t8 + t12 + t33 + t67 + t69
y25 = x25 + t44 + t66 + t75
y26 = x1 + t15 + t42 + t43 + t51 + t72
y27 = x11 + t7 + t47 + t54
y28 = x12 + t9 + t22 + t48 + t76
y29 = x13 + t35 + t45 + t68
y30 = t44 + t47 + t65 + t79
y31 = x1 + x19 + x30 + t0 + t61
```

Listing D.3: Straight-line program for the matrix $M^{i}_{8,4}$ from Eq. (5.6) that uses 212 XORs.

```
t0 = x0 + x14
t1 = x1 + x10
t2 = x11 + x21
t3 = x19 + x20
t4 = x15 + x22
t5 = x16 + x30
t6 = x17 + x31
t7 = x18 + x28
t8 = x2 + x29
t9 = x3 + x5
t10 = x4 + t1
t11 = x9 + x23
t12 = x7 + x24
t13 = x12 + t3
t14 = x25 + t4
t15 = x6 + x13
t16 = x8 + t0
t17 = x27 + t5
t18 = t2 + t8
t19 = x26 + t7
t20 = t10 + t17
t21 = x24 + t14
t22 = x14 + t7
t23 = x23 + t6
t24 = t6 + t11
```

```
t25 = t12 + t18
t26 = x13 + x17
t27 = x20 + t15
t28 = x26 + t16
t29 = x30 + t13
t30 = t2 + t12
t31 = x1 + t9
t32 = x2 + t16
t33 = x3 + x8
t34 = x4 + t5
t35 = x5 + t0
t36 = x9 + t20
t37 = x10 + x22
t38 = x17 + t15
t39 = x19 + t9
t40 = x21 + x29
t41 = x25 + x26
t42 = x25 + t24
t43 = x27 + t22
t44 = x28 + x29
t45 = x31 + t28
t46 = t6 + t21
t47 = t10 + t13
t48 = t11 + t19
t49 = x0 + x11
t50 = x3 + x15
t51 = x4 + t19
t52 = x5 + x8
t53 = x6 + x12
t54 = x6 + t29
t55 = x7 + x13
t56 = x7 + t0
t57 = x9 + t1
t58 = x10 + x18
t59 = x11 + t42
t60 = x12 + x16
t61 = x13 + x15
t62 = x16 + t13
t63 = x27 + t8
t64 = x28 + t18
t65 = x31 + t25
t66 = t3 + t7
t67 = t4 + t36
t68 = t9 + t27
t69 = t9 + t45
t70 = t14 + t22
t71 = t15 + t21
t72 = t19 + t57
t73 = t20 + t33
```

```
t74 = t23 + t37
t75 = t23 + t52
t76 = t24 + t30
t77 = t25 + t43
t78 = t26 + t48
t79 = t27 + t44
t80 = t30 + t32
t81 = t35 + t65
t82 = t38 + t50
t83 = t46 + t49
y0 = x0 + x18 + t18 + t41 + t47 + t75
y1 = t17 + t33 + t37 + t42 + t79
y2 = x29 + t5 + t28 + t58 + t76
y3 = t36 + t66 + t83
y4 = x7 + t6 + t35 + t40 + t67
y5 = x4 + t11 + t53 + t77
y6 = x8 + x10 + t9 + t14 + t29 + t44 + t55
y7 = t2 + t11 + t34 + t45 + t79
y8 = x6 + t3 + t58 + t63 + t69
y9 = x15 + t5 + t72 + t81
y10 = x19 + t8 + t53 + t73 + t83
y11 = x2 + x30 + t31 + t51 + t55 + t59
y12 = t41 + t54 + t56 + t74
y13 = t2 + t3 + t20 + t38 + t70
y14 = x14 + t4 + t13 + t25 + t51 + t75
y15 = x18 + t11 + t17 + t39 + t40 + t71
y16 = x23 + t20 + t39 + t41 + t61 + t64
y17 = x21 + t1 + t16 + t23 + t62 + t71
y18 = x1 + t14 + t78 + t80
y19 = x3 + t0 + t21 + t47 + t48 + t63
y20 = t16 + t31 + t40 + t46 + t54
y21 = t8 + t39 + t56 + t60 + t78
y22 = x29 + t10 + t29 + t43 + t82
y23 = t34 + t61 + t66 + t81
y24 = x25 + t1 + t3 + t26 + t77
y25 = x26 + t4 + t12 + t26 + t73
y26 = x5 + x18 + x27 + t32 + t59 + t60
y27 = x0 + x24 + t62 + t72 + t82
y28 = x6 + x16 + t31 + t70 + t76
y29 = x9 + t47 + t64 + t69
y30 = x12 + t18 + t67 + t68
y31 = t34 + t68 + t74 + t80
```

Bibliography

[Abd+12] Mohamed Ahmed Abdelraheem, Martin Ågren, Peter Beelen, and Gregor Leander. "On the Distribution of Linear Biases: Three Instructive Examples." In: *CRYPTO 2012*. Ed. by Reihaneh Safavi-Naini and Ran Canetti. Vol. 7417. LNCS. Springer, Heidelberg, Aug. 2012, pp. 50–67. DOI: 10.1007/978-3-642-32009-5_4.

[Abd13] Mohamed Ahmed Abdelraheem. "Estimating the Probabilities of Low-Weight Differential and Linear Approximations on PRESENT-Like Ciphers." In: *ICISC 12*. Ed. by Taekyoung Kwon, Mun-Kyu Lee, and Daesung Kwon. Vol. 7839. LNCS. Springer, Heidelberg, Nov. 2013, pp. 368–382. DOI: 10.1007/978-3-642-37682-5_26.

[AES01] *Advanced Encryption Standard (AES)*. National Institute of Standards and Technology (NIST), FIPS PUB 197, U.S. Department of Commerce. Nov. 2001.

[AF15] Daniel Augot and Matthieu Finiasz. "Direct Construction of Recursive MDS Diffusion Layers Using Shortened BCH Codes." In: *FSE 2014*. Ed. by Carlos Cid and Christian Rechberger. Vol. 8540. LNCS. Springer, Heidelberg, Mar. 2015, pp. 3–17. DOI: 10.1007/978-3-662-46706-0_1.

[Alb+14] Martin R. Albrecht, Benedikt Driessen, Elif Bilge Kavun, Gregor Leander, Christof Paar, and Tolga Yalçin. "Block Ciphers - Focus on the Linear Layer (feat. PRIDE)." In: *CRYPTO 2014, Part I*. Ed. by Juan A. Garay and Rosario Gennaro. Vol. 8616. LNCS. Springer, Heidelberg, Aug. 2014, pp. 57–76. DOI: 10.1007/978-3-662-44371-2_4.

[Alb+15] Martin R. Albrecht, Christian Rechberger, Thomas Schneider, Tyge Tiessen, and Michael Zohner. "Ciphers for MPC and FHE." In: *EUROCRYPT 2015, Part I*. Ed. by Elisabeth Oswald and Marc Fischlin. Vol. 9056. LNCS. Springer, Heidelberg, Apr. 2015, pp. 430–454. DOI: 10.1007/978-3-662-46800-5_17.

[And+14] Elena Andreeva, Begül Bilgin, Andrey Bogdanov, Atul Luykx, Florian Mendel, Bart Mennink, Nicky Mouha, Qingju Wang, and Kan Yasuda. *PRIMATEs v1.02*. Submission to the CAESAR competition. 2014.

[Ank+19] Ralph Ankele, Christoph Dobraunig, Jian Guo, Eran Lambooij, Gregor Leander, and Yosuke Todo. "Zero-Correlation Attacks on Tweakable Block Ciphers with Linear Tweakey Expansion." In: *IACR Trans. Symm. Cryptol.* 2019.1 (2019), pp. 192–235. ISSN: 2519-173X. DOI: 10.13154/tosc.v2019.i1.192-235.

[Ava17] Roberto Avanzi. "The QARMA Block Cipher Family." In: *IACR Trans. Symm. Cryptol.* 2017.1 (2017), pp. 4–44. ISSN: 2519-173X. DOI: 10.13154/tosc.v2017.i1.4-44.

[BAK98] Eli Biham, Ross J. Anderson, and Lars R. Knudsen. “Serpent: A New Block Cipher Proposal.” In: *FSE’98*. Ed. by Serge Vaudenay. Vol. 1372. LNCS. Springer, Heidelberg, Mar. 1998, pp. 222–238. DOI: 10.1007/3-540-69710-1_15.

[Ban+15] Subhadeep Banik, Andrey Bogdanov, Takanori Isobe, Kyoji Shibutani, Harunaga Hiwatari, Toru Akishita, and Francesco Regazzoni. “Midori: A Block Cipher for Low Energy.” In: *ASIACRYPT 2015, Part II*. Ed. by Tetsu Iwata and Jung Hee Cheon. Vol. 9453. LNCS. Springer, Heidelberg, Nov. 2015, pp. 411–436. DOI: 10.1007/978-3-662-48800-3_17.

[Bar+10] Paulo Barreto, Ventzislav Nikov, Svetla Nikova, Vincent Rijmen, and Elmar Tischhauser. “Whirlwind: a new cryptographic hash function.” In: *Designs, Codes and Cryptography* 56.2 (Aug. 2010), pp. 141–162. ISSN: 1573-7586. DOI: 10.1007/s10623-010-9391-y.

[BBK14] Alex Biryukov, Charles Bouillaguet, and Dmitry Khovratovich. “Cryptographic Schemes Based on the ASASA Structure: Black-Box, White-Box, and Public-Key (Extended Abstract).” In: *ASIACRYPT 2014, Part I*. Ed. by Palash Sarkar and Tetsu Iwata. Vol. 8873. LNCS. Springer, Heidelberg, Dec. 2014, pp. 63–84. DOI: 10.1007/978-3-662-45611-8_4.

[BBR16a] Subhadeep Banik, Andrey Bogdanov, and Francesco Regazzoni. *Atomic-AES v2.0*. Cryptology ePrint Archive, Report 2016/1005. http://eprint.iacr.org/2016/1005. 2016.

[BBR16b] Subhadeep Banik, Andrey Bogdanov, and Francesco Regazzoni. “Atomic-AES: A Compact Implementation of the AES Encryption/Decryption Core.” In: *INDOCRYPT 2016*. Ed. by Orr Dunkelman and Somitra Kumar Sanadhya. Vol. 10095. LNCS. Springer, Heidelberg, Dec. 2016, pp. 173–190. DOI: 10.1007/978-3-319-49890-4_10.

[BC13] Christina Boura and Anne Canteaut. “On the Influence of the Algebraic Degree of F^{-1} on the Algebraic Degree of $G \circ F$.” In: *IEEE Transactions on Information Theory* 59.1 (2013), pp. 691–702. DOI: 10.1109/TIT.2012.2214203. URL: http://dx.doi.org/10.1109/TIT.2012.2214203.

[BC14] Daniel J. Bernstein and Tung Chou. “Faster Binary-Field Multiplication and Faster Binary-Field MACs.” In: *SAC 2014*. Ed. by Antoine Joux and Amr M. Youssef. Vol. 8781. LNCS. Springer, Heidelberg, Aug. 2014, pp. 92–111. DOI: 10.1007/978-3-319-13051-4_6.

[BDK01] Eli Biham, Orr Dunkelman, and Nathan Keller. “The Rectangle Attack - Rectangling the Serpent.” In: *EUROCRYPT 2001*. Ed. by Birgit Pfitzmann. Vol. 2045. LNCS. Springer, Heidelberg, May 2001, pp. 340–357. DOI: 10.1007/3-540-44987-6_21.

[Bei+16] Christof Beierle, Jérémy Jean, Stefan Kölbl, Gregor Leander, Amir Moradi, Thomas Peyrin, Yu Sasaki, Pascal Sasdrich, and Siang Meng Sim. “The SKINNY Family of Block Ciphers and Its Low-Latency Variant MANTIS.” In: *CRYPTO 2016, Part II*. Ed. by Matthew Robshaw and Jonathan Katz. Vol. 9815. LNCS. Springer, Heidelberg, Aug. 2016, pp. 123–153. DOI: 10.1007/978-3-662-53008-5_5.

[Bei+17] Christof Beierle, Anne Canteaut, Gregor Leander, and Yann Rotella. “Proving Resistance Against Invariant Attacks: How to Choose the Round Constants.” In: *CRYPTO 2017, Part II*. Ed. by Jonathan Katz and Hovav Shacham. Vol. 10402. LNCS. Springer, Heidelberg, Aug. 2017, pp. 647–678. DOI: 10.1007/978-3-319-63715-0_22.

[Bei+19] Christof Beierle, Gregor Leander, Amir Moradi, and Shahram Rasoolzadeh. “CRAFT: Lightweight Tweakable Block Cipher with Efficient Protection Against DFA Attacks.” In: *IACR Trans. Symm. Cryptol.* 2019.1 (2019), pp. 5–45. ISSN: 2519-173X. DOI: 10.13154/tosc.v2019.i1.5-45.

[Bel+01] Mihir Bellare, Alexandra Boldyreva, Lars R. Knudsen, and Chanathip Namprempre. “Online Ciphers and the Hash-CBC Construction.” In: *CRYPTO 2001*. Ed. by Joe Kilian. Vol. 2139. LNCS. Springer, Heidelberg, Aug. 2001, pp. 292–309. DOI: 10.1007/3-540-44647-8_18.

[Ber+11] Guido Bertoni, Joan Daemen, Michaël Peeters, and GV Assche. “The Keccak reference.” In: *Submission to NIST (Round 3)* 13 (2011).

[Ber09] Daniel J. Bernstein. “Optimizing linear maps modulo 2.” In: *Workshop Record of SPEED-CC – Software Performance Enhancement for Encryption and Decryption and Cryptographic Compilers*. 2009, pp. 3–18.

[BFI19] Subhadeep Banik, Yuki Funabiki, and Takanori Isobe. “More Results on Shortest Linear Programs.” In: *Advances in Information and Computer Security*. 2019, pp. 109–128. ISBN: 978-3-030-26834-3.

[BFP17] Joan Boyar, Magnus Gausdal Find, and René Peralta. “Low-Depth, Low-Size Circuits for Cryptographic Applications.” BFA 2017. 2017.

[BI15] Andrey Bogdanov and Takanori Isobe. “White-Box Cryptography Revisited: Space-Hard Ciphers.” In: *ACM CCS 2015*. Ed. by Indrajit Ray, Ninghui Li, and Christopher Kruegel. ACM Press, Oct. 2015, pp. 1058–1069. DOI: 10.1145/2810103.2813699.

[Bih00] Eli Biham. “Cryptanalysis of Patarin’s 2-Round Public Key System with S Boxes (2R).” In: *EUROCRYPT 2000*. Ed. by Bart Preneel. Vol. 1807. LNCS. Springer, Heidelberg, May 2000, pp. 408–416. DOI: 10.1007/3-540-45539-6_28.

[Bih94] Eli Biham. “New Types of Cryptanalytic Attacks Using related Keys (Extended Abstract).” In: *EUROCRYPT’93*. Ed. by Tor Helleseth. Vol. 765. LNCS. Springer, Heidelberg, May 1994, pp. 398–409. DOI: 10.1007/3-540-48285-7_34.

[Bih95] Eli Biham. “On Matsui’s Linear Cryptanalysis.” In: *EUROCRYPT’94*. Ed. by Alfredo De Santis. Vol. 950. LNCS. Springer, Heidelberg, May 1995, pp. 341–355. DOI: 10.1007/BFb0053449.

[Bil+13] Begül Bilgin, Andrey Bogdanov, Miroslav Knežević, Florian Mendel, and Qingju Wang. “Fides: Lightweight Authenticated Cipher with Side-Channel Resistance for Constrained Hardware.” In: *CHES 2013*. Ed. by Guido Bertoni and Jean-Sébastien Coron. Vol. 8086. LNCS. Springer, Heidelberg, Aug. 2013, pp. 142–158. DOI: 10.1007/978-3-642-40349-1_9.

[Bir+04] Alex Biryukov, Christophe De Cannieére, Joseph Lano, Siddika Berna Ors, and Bart Preneel. *Security and Performance Analysis of ARIA*. Jan. 2004. URL: https://www.esat.kuleuven.be/cosic/publications/article-500.pdf.

[BIT16] Andrey Bogdanov, Takanori Isobe, and Elmar Tischhauser. "Towards Practical White-box Cryptography: Optimizing Efficiency and Space Hardness." In: *ASIACRYPT 2016, Part I*. Ed. by Jung Hee Cheon and Tsuyoshi Takagi. Vol. 10031. LNCS. Springer, Heidelberg, Dec. 2016, pp. 126–158. DOI: 10.1007/978-3-662-53887-6_5.

[BK15] Alex Biryukov and Dmitry Khovratovich. *Decomposition attack on SASASASAS*. Cryptology ePrint Archive, Report 2015/646. http://eprint.iacr.org/2015/646. 2015.

[BKL16] Christof Beierle, Thorsten Kranz, and Gregor Leander. "Lightweight Multiplication in GF(2^n) with Applications to MDS Matrices." In: *CRYPTO 2016, Part I*. Ed. by Matthew Robshaw and Jonathan Katz. Vol. 9814. LNCS. Springer, Heidelberg, Aug. 2016, pp. 625–653. DOI: 10.1007/978-3-662-53018-4_23.

[BKN09] Alex Biryukov, Dmitry Khovratovich, and Ivica Nikolić. "Distinguisher and Related-Key Attack on the Full AES-256." In: *CRYPTO 2009*. Ed. by Shai Halevi. Vol. 5677. LNCS. Springer, Heidelberg, Aug. 2009, pp. 231–249. DOI: 10.1007/978-3-642-03356-8_14.

[BKP16] Alex Biryukov, Dmitry Khovratovich, and Léo Perrin. "Multiset-Algebraic Cryptanalysis of Reduced Kuznyechik, Khazad, and secret SPNs." In: *IACR Trans. Symm. Cryptol.* 2016.2 (2016). http://tosc.iacr.org/index.php/ToSC/article/view/572, pp. 226–247. ISSN: 2519-173X. DOI: 10.13154/tosc.v2016.i2.226-247.

[BLP16] Alex Biryukov, Gaëtan Leurent, and Léo Perrin. "Cryptanalysis of Feistel Networks with Secret Round Functions." In: *SAC 2015*. Ed. by Orr Dunkelman and Liam Keliher. Vol. 9566. LNCS. Springer, Heidelberg, Aug. 2016, pp. 102–121. DOI: 10.1007/978-3-319-31301-6_6.

[BMP08] Joan Boyar, Philip Matthews, and René Peralta. "On the Shortest Linear Straight-Line Program for Computing Linear Forms." In: *MFCS 2008*. Vol. 5162. LNCS. 2008, pp. 168–179. DOI: 10.1007/978-3-540-85238-4_13.

[BMP13] Joan Boyar, Philip Matthews, and René Peralta. "Logic Minimization Techniques with Applications to Cryptology." In: *Journal of Cryptology* 26.2 (Apr. 2013), pp. 280–312. DOI: 10.1007/s00145-012-9124-7.

[BN16] Céline Blondeau and Kaisa Nyberg. "Improved Parameter Estimates for Correlation and Capacity Deviates in Linear Cryptanalysis." In: *IACR Trans. Symm. Cryptol.* 2016.2 (2016). http://tosc.iacr.org/index.php/ToSC/article/view/570, pp. 162–191. ISSN: 2519-173X. DOI: 10.13154/tosc.v2016.i2.162-191.

[BN17] Céline Blondeau and Kaisa Nyberg. "Joint Data and Key Distribution of Simple, Multiple, and Multidimensional Linear Cryptanalysis Test Statistic and Its Impact to Data Complexity." In: *Designs, Codes and Cryptography* 82.1 (2017), pp. 319–349. DOI: 10.1007/s10623-016-0268-6.

[Bog+07] Andrey Bogdanov, Lars R. Knudsen, Gregor Leander, Christof Paar, Axel Poschmann, Matthew J. B. Robshaw, Yannick Seurin, and C. Vikkelsoe. “PRESENT: An Ultra-Lightweight Block Cipher.” In: *CHES 2007*. Ed. by Pascal Paillier and Ingrid Verbauwhede. Vol. 4727. LNCS. Springer, Heidelberg, Sept. 2007, pp. 450–466. DOI: 10.1007/978-3-540-74735-2_31.

[Bor+12] Julia Borghoff, Anne Canteaut, Tim Güneysu, Elif Bilge Kavun, Miroslav Knežević, Lars R. Knudsen, Gregor Leander, Ventzislav Nikov, Christof Paar, Christian Rechberger, Peter Rombouts, Søren S. Thomsen, and Tolga Yalçin. “PRINCE - A Low-Latency Block Cipher for Pervasive Computing Applications - Extended Abstract.” In: *ASIACRYPT 2012*. Ed. by Xiaoyun Wang and Kazue Sako. Vol. 7658. LNCS. Springer, Heidelberg, Dec. 2012, pp. 208–225. DOI: 10.1007/978-3-642-34961-4_14.

[Bor+13] Julia Borghoff, Lars R. Knudsen, Gregor Leander, and Søren S. Thomsen. “Slender-Set Differential Cryptanalysis.” In: *Journal of Cryptology* 26.1 (Jan. 2013), pp. 11–38. DOI: 10.1007/s00145-011-9111-4.

[BP10] Joan Boyar and René Peralta. “A New Combinational Logic Minimization Technique with Applications to Cryptology.” In: *SEA 2010*. Vol. 6049. LNCS. 2010, pp. 178–189. DOI: 10.1007/978-3-642-13193-6_16.

[BP17] Alex Biryukov and Leo Perrin. *State of the Art in Lightweight Symmetric Cryptography*. Cryptology ePrint Archive, Report 2017/511. http://eprint.iacr.org/2017/511. 2017.

[BPU16] Alex Biryukov, Léo Perrin, and Aleksei Udovenko. “Reverse-Engineering the S-Box of Streebog, Kuznyechik and STRIBOBr1.” In: *EUROCRYPT 2016, Part I*. Ed. by Marc Fischlin and Jean-Sébastien Coron. Vol. 9665. LNCS. Springer, Heidelberg, May 2016, pp. 372–402. DOI: 10.1007/978-3-662-49890-3_15.

[BPW15] Céline Blondeau, Thomas Peyrin, and Lei Wang. “Known-Key Distinguisher on Full PRESENT.” In: *CRYPTO 2015, Part I*. Ed. by Rosario Gennaro and Matthew J. B. Robshaw. Vol. 9215. LNCS. Springer, Heidelberg, Aug. 2015, pp. 455–474. DOI: 10.1007/978-3-662-47989-6_22.

[BRa] Paulo Barreto and Vincent Rijmen. *The ANUBIS Block Cipher*. First Open NESSIE Workshop.

[BRb] Paulo Barreto and Vincent Rijmen. *The Khazad legacy-level Block Cipher*. First Open NESSIE Workshop.

[BRc] Paulo Barreto and Vincent Rijmen. *The* WHIRLPOOL *Hashing Function*. First Open NESSIE Workshop.

[BR05] Mihir Bellare and Phillip Rogaway. *Introduction to modern cryptography*. Lecture notes. Available online at http://web.cs.ucdavis.edu/~rogaway/classes/227/spring05/book/main.pdf. 2005.

[BR14] Andrey Bogdanov and Vincent Rijmen. “Linear hulls with correlation zero and linear cryptanalysis of block ciphers.” In: *Designs, Codes and Cryptography* 70.3 (2014), pp. 369–383. DOI: 10.1007/s10623-012-9697-z.

[BS01] Alex Biryukov and Adi Shamir. “Structural Cryptanalysis of SASAS.” In: *EUROCRYPT 2001*. Ed. by Birgit Pfitzmann. Vol. 2045. LNCS. Springer, Heidelberg, May 2001, pp. 394–405. DOI: 10.1007/3-540-44987-6_24.

[BS10] Alex Biryukov and Adi Shamir. “Structural Cryptanalysis of SASAS.” In: *Journal of Cryptology* 23.4 (Oct. 2010), pp. 505–518. DOI: 10.1007/s00145-010-9062-1.

[BS91] Eli Biham and Adi Shamir. “Differential Cryptanalysis of DES-like Cryptosystems.” In: *CRYPTO’90*. Ed. by Alfred J. Menezes and Scott A. Vanstone. Vol. 537. LNCS. Springer, Heidelberg, Aug. 1991, pp. 2–21. DOI: 10.1007/3-540-38424-3_1.

[BTV18] Andrey Bogdanov, Elmar Tischhauser, and Philip S. Vejre. “Multivariate Profiling of Hulls for Linear Cryptanalysis.” In: *IACR Trans. Symm. Cryptol.* 2018.1 (2018), pp. 101–125. ISSN: 2519-173X. DOI: 10.13154/tosc.v2018.i1.101-125.

[BW99] Alex Biryukov and David Wagner. “Slide Attacks.” In: *FSE’99*. Ed. by Lars R. Knudsen. Vol. 1636. LNCS. Springer, Heidelberg, Mar. 1999, pp. 245–259. DOI: 10.1007/3-540-48519-8_18.

[Can05] D. Canright. “A Very Compact S-Box for AES.” In: *CHES 2005*. Ed. by Josyula R. Rao and Berk Sunar. Vol. 3659. LNCS. Springer, Heidelberg, Aug. 2005, pp. 441–455. DOI: 10.1007/11545262_32.

[Can07] Christophe De Cannière. “Analysis and design of symmetric encryption algorithms.” PhD thesis. Katholieke Universiteit Leuven, 2007.

[Car07] Claude Carlet. “Boolean Functions for Cryptography and Error Correcting Codes.” In: *Boolean Methods and Models*. Ed. by Yves Crama and Peter Hammer. Cambridge University Press, 2007.

[Cho+02] S. Chow, P. Eisen, H. Johnson, and P. C. Van Oorschot. “A White-Box DES Implementation for DRM Applications.” In: *Proceedings of ACM CCS-9 Workshop DRM*. Springer, 2002, pp. 1–15.

[Cho+03] Stanley Chow, Philip A. Eisen, Harold Johnson, and Paul C. van Oorschot. “White-Box Cryptography and an AES Implementation.” In: *SAC 2002*. Ed. by Kaisa Nyberg and Howard M. Heys. Vol. 2595. LNCS. Springer, Heidelberg, Aug. 2003, pp. 250–270. DOI: 10.1007/3-540-36492-7_17.

[Cho+12] Jiali Choy, Huihui Yap, Khoongming Khoo, Jian Guo, Thomas Peyrin, Axel Poschmann, and Chik How Tan. “SPN-Hash: Improving the Provable Resistance against Differential Collision Attacks.” In: *AFRICACRYPT 12*. Ed. by Aikaterini Mitrokotsa and Serge Vaudenay. Vol. 7374. LNCS. Springer, Heidelberg, July 2012, pp. 270–286.

[Cho+16] Jihoon Cho, Kyu Young Choi, Orr Dunkelman, Nathan Keller, Dukjae Moon, and Aviya Vaidberg. “Hybrid WBC: Secure and Efficient White-Box Encryption Schemes.” In: *CANS 16*. Ed. by Sara Foresti and Giuseppe Persiano. Vol. 10052. LNCS. Springer, Heidelberg, Nov. 2016, pp. 749–754. DOI: 10.1007/978-3-319-48965-0_55.

[Cho10] Joo Yeon Cho. “Linear Cryptanalysis of Reduced-Round PRESENT.” In: *CT-RSA 2010*. Ed. by Josef Pieprzyk. Vol. 5985. LNCS. Springer, Heidelberg, Mar. 2010, pp. 302–317. DOI: 10.1007/978-3-642-11925-5_21.

[CMR05] Carlos Cid, Sean Murphy, and Matthew J. B. Robshaw. “Small Scale Variants of the AES.” In: *FSE 2005*. Ed. by Henri Gilbert and Helena Handschuh. Vol. 3557. LNCS. Springer, Heidelberg, Feb. 2005, pp. 145–162. DOI: 10.1007/11502760_10.

[Dae+00] Joan Daemen, Michaël Peeters, Gilles Van Assche, and Vincent Rijmen. *Nessie Proposal: NOEKEON*. http://gro.noekeon.org/Noekeon-spec.pdf. 2000.

[Dae95] Joan Daemen. “Cipher and Hash Function Design Strategies based on linear and differential cryptanalysis.” PhD thesis. Katholieke Universiteit Leuven, 1995.

[Del+14] Cécile Delerablée, Tancrède Lepoint, Pascal Paillier, and Matthieu Rivain. “White-Box Security Notions for Symmetric Encryption Schemes.” In: *SAC 2013*. Ed. by Tanja Lange, Kristin Lauter, and Petr Lisonek. Vol. 8282. LNCS. Springer, Heidelberg, Aug. 2014, pp. 247–264. DOI: 10.1007/978-3-662-43414-7_13.

[DES77] *Data Encryption Standard*. National Bureau of Standards, NBS FIPS PUB 46, U.S. Department of Commerce. Jan. 1977.

[DF04] David Steven Dummit and Richard M Foote. *Abstract algebra*. John Wiley and Sons, Inc., 2004.

[DGV95] Joan Daemen, René Govaerts, and Joos Vandewalle. “Correlation Matrices.” In: *FSE’94*. Ed. by Bart Preneel. Vol. 1008. LNCS. Springer, Heidelberg, Dec. 1995, pp. 275–285. DOI: 10.1007/3-540-60590-8_21.

[Din+15] Itai Dinur, Orr Dunkelman, Thorsten Kranz, and Gregor Leander. *Decomposing the ASASA Block Cipher Construction*. Cryptology ePrint Archive, Report 2015/507. http://eprint.iacr.org/2015/507. 2015.

[Din18] Itai Dinur. “An Improved Affine Equivalence Algorithm for Random Permutations.” In: *EUROCRYPT 2018, Part I*. Ed. by Jesper Buus Nielsen and Vincent Rijmen. Vol. 10820. LNCS. Springer, Heidelberg, Apr. 2018, pp. 413–442. DOI: 10.1007/978-3-319-78381-9_16.

[DKR97] Joan Daemen, Lars R. Knudsen, and Vincent Rijmen. “The Block Cipher Square.” In: *FSE’97*. Ed. by Eli Biham. Vol. 1267. LNCS. Springer, Heidelberg, Jan. 1997, pp. 149–165. DOI: 10.1007/BFb0052343.

[DL18] Sébastien Duval and Gaëtan Leurent. “MDS Matrices with Lightweight Circuits.” In: *IACR Trans. Symm. Cryptol.* 2018.2 (2018), pp. 48–78. ISSN: 2519-173X. DOI: 10.13154/tosc.v2018.i2.48-78.

[Dob+18] Christoph Dobraunig, Maria Eichlseder, Lorenzo Grassi, Virginie Lallemand, Gregor Leander, Eik List, Florian Mendel, and Christian Rechberger. “Rasta: A Cipher with Low ANDdepth and Few ANDs per Bit.” In: *CRYPTO 2018, Part I*. Ed. by Hovav Shacham and Alexandra Boldyreva. Vol. 10991. LNCS. Springer, Heidelberg, Aug. 2018, pp. 662–692. DOI: 10.1007/978-3-319-96884-1_22.

[DR01] Joan Daemen and Vincent Rijmen. “The Wide Trail Design Strategy.” In: *8th IMA International Conference on Cryptography and Coding*. Ed. by Bahram Honary. Vol. 2260. LNCS. Springer, Heidelberg, Dec. 2001, pp. 222–238.

[DR02] Joan Daemen and Vincent Rijmen. *The Design of Rijndael: AES - The Advanced Encryption Standard*. Information Security and Cryptography. Springer, 2002. ISBN: 3-540-42580-2. DOI: 10.1007/978-3-662-04722-4.

[DR05] Joan Daemen and Vincent Rijmen. *Probability distributions of Correlation and Differentials in Block Ciphers*. Cryptology ePrint Archive, Report 2005/212. http://eprint.iacr.org/2005/212. 2005.

[DR07] Joan Daemen and Vincent Rijmen. "Probability Distributions of Correlation and Differentials in Block Ciphers." In: *Journal of Mathematical Cryptology* 1.3 (2007), pp. 221–242. DOI: 10.1515/JMC.2007.011.

[DR11] Joan Daemen and Vincent Rijmen. "Correlation Analysis in $GF(2^n)$." In: *Advanced Linear Cryptanalysis of Block and Stream Ciphers. Cryptology and information security* (2011), pp. 115–131.

[DR98] Joan Daemen and Vincent Rijmen. *AES Proposal: Rjindael*. http://csrc.nist.gov/archive/aes/rijndael/Rijndael-ammended.pdf. 1998.

[FBR06] Décio Luiz Gazzoni Filho, Paulo S L M Barreto, and Vincent Rijmen. "The MAELSTROM-0 Hash Function." In: *Brazilian Symposium on Information and Computer Systems Security*. 2006.

[Fou+16] Pierre-Alain Fouque, Pierre Karpman, Paul Kirchner, and Brice Minaud. "Efficient and Provable White-Box Primitives." In: *ASIACRYPT 2016, Part I*. Ed. by Jung Hee Cheon and Tsuyoshi Takagi. Vol. 10031. LNCS. Springer, Heidelberg, Dec. 2016, pp. 159–188. DOI: 10.1007/978-3-662-53887-6_6.

[FS10] Carsten Fuhs and Peter Schneider-Kamp. "Synthesizing Shortest Linear Straight-Line Programs over GF(2) Using SAT." In: *SAT*. Vol. 6175. LNCS. Springer, 2010, pp. 71–84. DOI: 10.1007/978-3-642-14186-7_8.

[FS12] Carsten Fuhs and Peter Schneider-Kamp. "Optimizing the AES S-Box using SAT." In: *IWIL 2010. The 8th International Workshop on the Implementation of Logics*. Ed. by Geoff Sutcliffe, Stephan Schulz, and Eugenia Ternovska. Vol. 2. EPiC Series in Computing. EasyChair, 2012, pp. 64–70.

[Gau+] Praveen Gauravaram, Lars R. Knudsen, Krystian Matusiewicz, Florian Mendel, Christian Rechberger, Martin Schläffer, and Søren S. Thomsen. *Grøstl – a SHA-3 candidate*. Submitted to SHA-3.

[Gér+13] Benoît Gérard, Vincent Grosso, María Naya-Plasencia, and François-Xavier Standaert. "Block Ciphers That Are Easier to Mask: How Far Can We Go?" In: *CHES 2013*. Ed. by Guido Bertoni and Jean-Sébastien Coron. Vol. 8086. LNCS. Springer, Heidelberg, Aug. 2013, pp. 383–399. DOI: 10.1007/978-3-642-40349-1_22.

[Gil16] Henri Gilbert. "On White-Box Cryptography." Keynote talk at FSE 2016. 2016. URL: https://fse.rub.de/slides/wbc_fse2016_hg_2pp.pdf.

[Gou+19] Dahmun Goudarzi, Jérémy Jean, Stefan Kölbl, Thomas Peyrin, Matthieu Rivain, Yu Sasaki, and Siang Meng Sim. *Pyjamask v1.0*. Submission to the NIST Lightweight Cryptography Project. https://csrc.nist.gov/CSRC/media/Projects/Lightweight-Cryptography/documents/round-1/spec-doc/Pyjamask-spec.pdf. 2019.

[GPP11] Jian Guo, Thomas Peyrin, and Axel Poschmann. "The PHOTON Family of Lightweight Hash Functions." In: *CRYPTO 2011*. Ed. by Phillip Rogaway. Vol. 6841. LNCS. Springer, Heidelberg, Aug. 2011, pp. 222–239. DOI: 10.1007/978-3-642-22792-9_13.

[GPT15] Henri Gilbert, Jérôme Plût, and Joana Treger. "Key-Recovery Attack on the ASASA Cryptosystem with Expanding S-Boxes." In: *CRYPTO 2015, Part I*. Ed. by Rosario Gennaro and Matthew J. B. Robshaw. Vol. 9215. LNCS. Springer, Heidelberg, Aug. 2015, pp. 475–490. DOI: 10.1007/978-3-662-47989-6_23.

[GR13] Kishan Chand Gupta and Indranil Ghosh Ray. "On Constructions of Involutory MDS Matrices." In: *AFRICACRYPT 13*. Ed. by Amr Youssef, Abderrahmane Nitaj, and Aboul Ella Hassanien. Vol. 7918. LNCS. Springer, Heidelberg, June 2013, pp. 43–60. DOI: 10.1007/978-3-642-38553-7_3.

[GR14] Kishan Chand Gupta and Indranil Ghosh Ray. "Cryptographically significant MDS matrices based on circulant and circulant-like matrices for lightweight applications." In: *Cryptography and Communications* 7.2 (2014), pp. 257–287. ISSN: 1936-2455. DOI: 10.1007/s12095-014-0116-3. URL: http://dx.doi.org/10.1007/s12095-014-0116-3.

[Gra06] Erik W Grafarend. *Linear and Nonlinear Models: Fixed Effects, Random Effects, and Mixed Models*. Walter de Gruyter, 2006. ISBN: 3-110-16216-4.

[Gro+15] Vincent Grosso, Gaëtan Leurent, François-Xavier Standaert, and Kerem Varici. "LS-Designs: Bitslice Encryption for Efficient Masked Software Implementations." In: *FSE 2014*. Ed. by Carlos Cid and Christian Rechberger. Vol. 8540. LNCS. Springer, Heidelberg, Mar. 2015, pp. 18–37. DOI: 10.1007/978-3-662-46706-0_2.

[Gro96] Lov K. Grover. "A Fast Quantum Mechanical Algorithm for Database Search." In: *28th ACM STOC*. ACM Press, May 1996, pp. 212–219. DOI: 10.1145/237814.237866.

[Guo+11] Jian Guo, Thomas Peyrin, Axel Poschmann, and Matthew J. B. Robshaw. "The LED Block Cipher." In: *CHES 2011*. Ed. by Bart Preneel and Tsuyoshi Takagi. Vol. 6917. LNCS. Springer, Heidelberg, Sept. 2011, pp. 326–341. DOI: 10.1007/978-3-642-23951-9_22.

[Hua+15] Jialin Huang, Serge Vaudenay, Xuejia Lai, and Kaisa Nyberg. "Capacity and Data Complexity in Multidimensional Linear Attack." In: *CRYPTO 2015, Part I*. Ed. by Rosario Gennaro and Matthew J. B. Robshaw. Vol. 9215. LNCS. Springer, Heidelberg, Aug. 2015, pp. 141–160. DOI: 10.1007/978-3-662-47989-6_7.

[JA09] Jorge Nakahara Jr. and Élcio Abrahão. "A New Involutory MDS Matrix for the AES." In: *I. J. Network Security* 9.2 (2009), pp. 109–116. URL: http://ijns.femto.com.tw/contents/ijns-v9-n2/ijns-2009-v9-n2-p109-116.pdf.

[Jea+16] Jérémy Jean, Ivica Nikolić, Thomas Peyrin, and Yannick Seurin. *Deoxys v1.41*. Submission to the CAESAR competition. 2016.

[Jea+17a] Jérémy Jean, Amir Moradi, Thomas Peyrin, and Pascal Sasdrich. “Bit-Sliding: A Generic Technique for Bit-Serial Implementations of SPN-based Primitives - Applications to AES, PRESENT and SKINNY.” In: *CHES 2017*. Ed. by Wieland Fischer and Naofumi Homma. Vol. 10529. LNCS. Springer, Heidelberg, Sept. 2017, pp. 687–707. DOI: 10.1007/978-3-319-66787-4_33.

[Jea+17b] Jérémy Jean, Thomas Peyrin, Siang Meng Sim, and Jade Tourteaux. “Optimizing Implementations of Lightweight Building Blocks.” In: *IACR Trans. Symm. Cryptol.* 2017.4 (2017), pp. 130–168. ISSN: 2519-173X. DOI: 10.13154/tosc.v2017.i4.130-168.

[Jea17] Jérémy Jean. *TikZ for Cryptographers*. https://www.iacr.org/authors/tikz/. 2017.

[JNP14a] Jérémy Jean, Ivica Nikolić, and Thomas Peyrin. *Joltik*. Submission to the CAESAR competition. 2014.

[JNP14b] Jérémy Jean, Ivica Nikolić, and Thomas Peyrin. “Tweaks and Keys for Block Ciphers: The TWEAKEY Framework.” In: *ASIACRYPT 2014, Part II*. Ed. by Palash Sarkar and Tetsu Iwata. Vol. 8874. LNCS. Springer, Heidelberg, Dec. 2014, pp. 274–288. DOI: 10.1007/978-3-662-45608-8_15.

[JV04] Pascal Junod and Serge Vaudenay. “FOX: A New Family of Block Ciphers.” In: *SAC 2004*. Ed. by Helena Handschuh and Anwar Hasan. Vol. 3357. LNCS. Springer, Heidelberg, Aug. 2004, pp. 114–129. DOI: 10.1007/978-3-540-30564-4_8.

[Ker83] Auguste Kerckhoffs. “La cryptographie militaire.” In: *Journal de sciences militaires* IX (1883), pp. 5–38, 161–191.

[Kho+14] Khoongming Khoo, Thomas Peyrin, Axel York Poschmann, and Huihui Yap. “FOAM: Searching for Hardware-Optimal SPN Structures and Components with a Fair Comparison.” In: *CHES 2014*. Ed. by Lejla Batina and Matthew Robshaw. Vol. 8731. LNCS. Springer, Heidelberg, Sept. 2014, pp. 433–450. DOI: 10.1007/978-3-662-44709-3_24.

[KJJ99] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. “Differential Power Analysis.” In: *CRYPTO’99*. Ed. by Michael J. Wiener. Vol. 1666. LNCS. Springer, Heidelberg, Aug. 1999, pp. 388–397. DOI: 10.1007/3-540-48405-1_25.

[KK06] Tim Kerins and Klaus Kursawe. “A Cautionary Note on Weak Implementations of Block Ciphers.” In: *1st Benelux Workshop on Information and System Security (WISSec 2006)*. 2006.

[KKS01] John Kelsey, Tadayoshi Kohno, and Bruce Schneier. “Amplified Boomerang Attacks Against Reduced-Round MARS and Serpent.” In: *FSE 2000*. Ed. by Bruce Schneier. Vol. 1978. LNCS. Springer, Heidelberg, Apr. 2001, pp. 75–93. DOI: 10.1007/3-540-44706-7_6.

[KL14] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography, Second Edition*. 2nd. Chapman & Hall/CRC, 2014. ISBN: 1466570261.

[KLW17] Thorsten Kranz, Gregor Leander, and Friedrich Wiemer. “Linear Cryptanalysis: Key Schedules and Tweakable Block Ciphers.” In: *IACR Trans. Symm. Cryptol.* 2017.1 (2017), pp. 474–505. ISSN: 2519-173X. DOI: 10.13154/tosc.v2017.i1.474-505.

[Kna07] Anthony W Knapp. *Basic algebra*. Springer Science & Business Media, 2007.

[Knu95] Lars R. Knudsen. “Truncated and Higher Order Differentials.” In: *FSE’94*. Ed. by Bart Preneel. Vol. 1008. LNCS. Springer, Heidelberg, Dec. 1995, pp. 196–211. DOI: 10.1007/3-540-60590-8_16.

[Koc96] Paul C. Kocher. “Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems.” In: *CRYPTO’96*. Ed. by Neal Koblitz. Vol. 1109. LNCS. Springer, Heidelberg, Aug. 1996, pp. 104–113. DOI: 10.1007/3-540-68697-5_9.

[Köl19] Lukas Kölsch. “XOR-Counts and Lightweight Multiplication with Fixed Elements in Binary Finite Fields.” In: *EUROCRYPT 2019, Part I*. Ed. by Yuval Ishai and Vincent Rijmen. Vol. 11476. LNCS. Springer, Heidelberg, May 2019, pp. 285–312. DOI: 10.1007/978-3-030-17653-2_10.

[KR07] Lars R. Knudsen and Vincent Rijmen. “Known-Key Distinguishers for Some Block Ciphers.” In: *ASIACRYPT 2007*. Ed. by Kaoru Kurosawa. Vol. 4833. LNCS. Springer, Heidelberg, Dec. 2007, pp. 315–324. DOI: 10.1007/978-3-540-76900-2_19.

[KR11] Lars R. Knudsen and Matthew J. B. Robshaw. *The Block Cipher Companion*. Springer Publishing Company, Incorporated, 2011. ISBN: 3642173411.

[Kra+17] Thorsten Kranz, Gregor Leander, Ko Stoffelen, and Friedrich Wiemer. “Shorter Linear Straight-Line Programs for MDS Matrices.” In: *IACR Trans. Symm. Cryptol.* 2017.4 (2017), pp. 188–211. ISSN: 2519-173X. DOI: 10.13154/tosc.v2017.i4.188-211.

[KW02] Lars R. Knudsen and David Wagner. “Integral Cryptanalysis.” In: *FSE 2002*. Ed. by Joan Daemen and Vincent Rijmen. Vol. 2365. LNCS. Springer, Heidelberg, Feb. 2002, pp. 112–127. DOI: 10.1007/3-540-45661-9_9.

[Kwo+03] Daesung Kwon, Jaesung Kim, Sangwoo Park, Soo Hak Sung, Yaekwon Sohn, Jung Hwan Song, Yongjin Yeom, E-Joong Yoon, Sangjin Lee, Jaewon Lee, Seongtaek Chee, Daewan Han, and Jin Hong. “New Block Cipher: ARIA.” In: *ICISC*. Vol. 2971. LNCS. Springer, 2003, pp. 432–445. DOI: 10.1007/978-3-540-24691-6_32.

[Lai94] Xuejia Lai. “Higher Order Derivatives and Differential Cryptanalysis.” In: "*Symposium on Communication, Coding and Cryptography*", *in honor of James L. Massey on the occasion of his 60’th birthday*. 1994, pp. 227–233.

[Lea+11] Gregor Leander, Mohamed Ahmed Abdelraheem, Hoda AlKhzaimi, and Erik Zenner. “A Cryptanalysis of PRINTcipher: The Invariant Subspace Attack.” In: *CRYPTO 2011*. Ed. by Phillip Rogaway. Vol. 6841. LNCS. Springer, Heidelberg, Aug. 2011, pp. 206–221. DOI: 10.1007/978-3-642-22792-9_12.

[LF04] Jérôme Lacan and Jérôme Fimes. “Systematic MDS erasure codes based on Vandermonde matrices.” In: *IEEE Communications Letters* 8.9 (2004), pp. 570–572. DOI: 10.1109/LCOMM.2004.833807.

[Li+19] Shun Li, Siwei Sun, Chaoyun Li, Zihao Wei, and Lei Hu. "Constructing Low-latency Involutory MDS Matrices with Lightweight Circuits." In: *IACR Trans. Symm. Cryptol.* 2019.1 (2019), pp. 84–117. ISSN: 2519-173X. DOI: 10.13154/tosc.v2019.i1.84-117.

[LM17] Gregor Leander and Alexander May. "Grover Meets Simon - Quantumly Attacking the FX-construction." In: *ASIACRYPT 2017, Part II*. Ed. by Tsuyoshi Takagi and Thomas Peyrin. Vol. 10625. LNCS. Springer, Heidelberg, Dec. 2017, pp. 161–178. DOI: 10.1007/978-3-319-70697-9_6.

[LMM91] Xuejia Lai, James L. Massey, and Sean Murphy. "Markov Ciphers and Differential Cryptanalysis." In: *EUROCRYPT'91*. Ed. by Donald W. Davies. Vol. 547. LNCS. Springer, Heidelberg, Apr. 1991, pp. 17–38. DOI: 10.1007/3-540-46416-6_2.

[LMR15] Gregor Leander, Brice Minaud, and Sondre Rønjom. "A Generic Approach to Invariant Subspace Attacks: Cryptanalysis of Robin, iSCREAM and Zorro." In: *EUROCRYPT 2015, Part I*. Ed. by Elisabeth Oswald and Marc Fischlin. Vol. 9056. LNCS. Springer, Heidelberg, Apr. 2015, pp. 254–283. DOI: 10.1007/978-3-662-46800-5_11.

[LN97] Rudolf Lidl and Harald Niederreiter. *Finite Fields*. EBL-Schweitzer. Cambridge University Press, 1997. ISBN: 9780521392310.

[LP07] Gregor Leander and Axel Poschmann. "On the Classification of 4 Bit S-Boxes." In: *WAIFI'07*. Ed. by Claude Carlet and Berk Sunar. Vol. 4547. LNCS. Springer, Heidelberg, June 2007, pp. 159–176. DOI: 10.1007/978-3-540-73074-3_13.

[LRW02] Moses Liskov, Ronald L. Rivest, and David Wagner. "Tweakable Block Ciphers." In: *CRYPTO 2002*. Ed. by Moti Yung. Vol. 2442. LNCS. Springer, Heidelberg, Aug. 2002, pp. 31–46. DOI: 10.1007/3-540-45708-9_3.

[LS16] Meicheng Liu and Siang Meng Sim. "Lightweight MDS Generalized Circulant Matrices." In: *FSE 2016*. Ed. by Thomas Peyrin. Vol. 9783. LNCS. Springer, Heidelberg, Mar. 2016, pp. 101–120. DOI: 10.1007/978-3-662-52993-5_6.

[LW16] Yongqiang Li and Mingsheng Wang. "On the Construction of Lightweight Circulant Involutory MDS Matrices." In: *FSE 2016*. Ed. by Thomas Peyrin. Vol. 9783. LNCS. Springer, Heidelberg, Mar. 2016, pp. 121–139. DOI: 10.1007/978-3-662-52993-5_7.

[LW17] Chaoyun Li and Qingju Wang. "Design of Lightweight Linear Diffusion Layers from Near-MDS Matrices." In: *IACR Trans. Symm. Cryptol.* 2017.1 (2017), pp. 129–155. ISSN: 2519-173X. DOI: 10.13154/tosc.v2017.i1.129-155.

[Mat94] Mitsuru Matsui. "Linear Cryptanalysis Method for DES Cipher." In: *EUROCRYPT'93*. Ed. by Tor Helleseth. Vol. 765. LNCS. Springer, Heidelberg, May 1994, pp. 386–397. DOI: 10.1007/3-540-48285-7_33.

[Max19] Alexander Maximov. *AES MixColumn with 92 XOR gates*. Cryptology ePrint Archive, Report 2019/833. https://eprint.iacr.org/2019/833. 2019.

[ME19] Alexander Maximov and Patrik Ekdahl. “New Circuit Minimization Techniques for Smaller and Faster AES SBoxes.” In: *IACR TCHES* 2019.4 (2019). https://tches.iacr.org/index.php/TCHES/article/view/8346, pp. 91–125. ISSN: 2569-2925. DOI: 10.13154/tches.v2019.i4.91-125.

[Mes+19] Sihem Mesnager, Kwang Ho Kim, Dujin Jo, Junyop Choe, Munhyon Han, and Dok Nam Lee. “A proof of the Beierle–Kranz–Leander conjecture related to lightweight multiplication in $\mathbb{F}_{2^n}$.” In: *Designs, Codes and Cryptography* (2019). ISSN: 1573-7586. DOI: 10.1007/s10623-019-00665-2.

[Min+15] Brice Minaud, Patrick Derbez, Pierre-Alain Fouque, and Pierre Karpman. “Key-Recovery Attacks on ASASA.” In: *ASIACRYPT 2015, Part II*. Ed. by Tetsu Iwata and Jung Hee Cheon. Vol. 9453. LNCS. Springer, Heidelberg, Nov. 2015, pp. 3–27. DOI: 10.1007/978-3-662-48800-3_1.

[Min+18] Brice Minaud, Patrick Derbez, Pierre-Alain Fouque, and Pierre Karpman. “Key-Recovery Attacks on ASASA.” In: *Journal of Cryptology* 31.3 (July 2018), pp. 845–884. DOI: 10.1007/s00145-017-9272-x.

[MS77] F. J. MacWilliams and N. J. A. Sloane. *The Theory of Error Correcting Codes*. North Holland, 1977.

[Mur11] S. Murphy. “The Return of the Cryptographic Boomerang.” In: *IEEE Transactions on Information Theory* 57.4 (Apr. 2011), pp. 2517–2521. ISSN: 0018-9448. DOI: 10.1109/TIT.2011.2111091.

[MVO96] Alfred J. Menezes, Scott A. Vanstone, and Paul C. Van Oorschot. *Handbook of Applied Cryptography*. 1st. Boca Raton, FL, USA: CRC Press, Inc., 1996. ISBN: 0849385237.

[NIST18] NIST Computer Security Resource Center (CSRC). *Submission Requirements and Evaluation Criteria for the Lightweight Cryptography Standardization Process*. https://csrc.nist.gov/CSRC/media/Projects/Lightweight-Cryptography/documents/final-lwc-submission-requirements-august2018.pdf. 2018.

[Nyb01] Kaisa Nyberg. “Correlation theorems in cryptanalysis.” In: *Discrete Applied Mathematics* 111.1-2 (2001), pp. 177–188. DOI: 10.1016/S0166-218X(00)00351-6.

[Nyb15] Kaisa Nyberg. *Linear Cryptanalysis (Lecture Notes)*. SAC 2015 Summer School. Available at http://sacworkshop.org/SAC15/S3-linear-all.pdf. Aug. 2015.

[Nyb95] Kaisa Nyberg. “Linear Approximation of Block Ciphers (Rump Session).” In: *EUROCRYPT’94*. Ed. by Alfredo De Santis. Vol. 950. LNCS. Springer, Heidelberg, May 1995, pp. 439–444. DOI: 10.1007/BFb0053460.

[OCo94] Luke O’Connor. “On the Distribution of Characteristics in Bijective Mappings.” In: *EUROCRYPT’93*. Ed. by Tor Helleseth. Vol. 765. LNCS. Springer, Heidelberg, May 1994, pp. 360–370. DOI: 10.1007/3-540-48285-7_31.

[OCo95] Luke O’Connor. “On the Distribution of Characteristics in Bijective Mappings.” In: *Journal of Cryptology* 8.2 (Mar. 1995), pp. 67–86. DOI: 10.1007/BF00190756.

[Ohk09] Kenji Ohkuma. “Weak Keys of Reduced-Round PRESENT for Linear Cryptanalysis.” In: *SAC 2009*. Ed. by Michael J. Jacobson Jr., Vincent Rijmen, and Reihaneh Safavi-Naini. Vol. 5867. LNCS. Springer, Heidelberg, Aug. 2009, pp. 249–265. DOI: 10.1007/978-3-642-05445-7_16.

[Paa97] Christof Paar. “Optimized Arithmetic for Reed-Solomon Encoders.” In: *ISIT*. IEEE, 1997. DOI: 10.1109/ISIT.1997.613165.

[PG97] Jacques Patarin and Louis Goubin. “Asymmetric cryptography with S-Boxes.” In: *Information and Communication Security, First International Conference, ICICS’97, Beijing, China, November 11-14, 1997, Proceedings*. 1997, pp. 369–380. DOI: 10.1007/BFb0028492. URL: http://dx.doi.org/10.1007/BFb0028492.

[PP09] Christof Paar and Jan Pelzl. *Understanding Cryptography: A Textbook for Students and Practitioners*. 1st. Springer Publishing Company, Incorporated, 2009. ISBN: 3642041000.

[PRC12] Gilles Piret, Thomas Roche, and Claude Carlet. “PICARO - A Block Cipher Allowing Efficient Higher-Order Side-Channel Resistance.” In: *ACNS 12*. Ed. by Feng Bao, Pierangela Samarati, and Jianying Zhou. Vol. 7341. LNCS. Springer, Heidelberg, June 2012, pp. 311–328. DOI: 10.1007/978-3-642-31284-7_19.

[PU16] Léo Perrin and Aleksei Udovenko. “Algebraic Insights into the Secret Feistel Network.” In: *FSE 2016*. Ed. by Thomas Peyrin. Vol. 9783. LNCS. Springer, Heidelberg, Mar. 2016, pp. 378–398. DOI: 10.1007/978-3-662-52993-5_19.

[Qua17] Qualcomm Technologies, Inc. *Pointer Authentication on ARMv8.3*. https://www.qualcomm.com/media/documents/files/whitepaper-pointer-authentication-on-armv8-3.pdf. 2017.

[Rag19] Shashank Raghuraman. “Efficiency of Logic Minimization Techniques for Cryptographic Hardware Implementation.” MA thesis. Virginia Polytechnic Institute and State University, 2019.

[RTA18] Arash Reyhani-Masoleh, Mostafa Taha, and Doaa Ashmawy. “Smashing the Implementation Records of AES S-box.” In: *IACR TCHES* 2018.2 (2018). https://tches.iacr.org/index.php/TCHES/article/view/884, pp. 298–336. ISSN: 2569-2925. DOI: 10.13154/tches.v2018.i2.298-336.

[RZ11] Phillip Rogaway and Haibin Zhang. “Online Ciphers from Tweakable Blockciphers.” In: *CT-RSA 2011*. Ed. by Aggelos Kiayias. Vol. 6558. LNCS. Springer, Heidelberg, Feb. 2011, pp. 237–249. DOI: 10.1007/978-3-642-19074-2_16.

[Saj+12a] Mahdi Sajadieh, Mohammad Dakhilalian, Hamid Mala, and Behnaz Omoomi. “On construction of involutory MDS matrices from Vandermonde Matrices in GF(2^q).” In: *Designs, Codes and Cryptography* 64.3 (Sept. 2012), pp. 287–308. ISSN: 1573-7586. DOI: 10.1007/s10623-011-9578-x.

[Saj+12b] Mahdi Sajadieh, Mohammad Dakhilalian, Hamid Mala, and Pouyan Sepehrdad. “Recursive Diffusion Layers for Block Ciphers and Hash Functions.” In: *FSE 2012*. Ed. by Anne Canteaut. Vol. 7549. LNCS. Springer, Heidelberg, Mar. 2012, pp. 385–401. DOI: 10.1007/978-3-642-34047-5_22.

[Sat+01] Akashi Satoh, Sumio Morioka, Kohji Takano, and Seiji Munetoh. "A Compact Rijndael Hardware Architecture with S-Box Optimization." In: *ASIACRYPT 2001*. Ed. by Colin Boyd. Vol. 2248. LNCS. Springer, Heidelberg, Dec. 2001, pp. 239–254. DOI: 10.1007/3-540-45682-1_15.

[Sch+98] Bruce Schneier, John Kelsey, Doug Whiting, David Wagner, Chris Hall, and Niels Ferguson. *Twofish: A 128-Bit Block Cipher*. 1998.

[Seg55] Beniamino Segre. "Curve razionali normali ek-archi negli spazi finiti." In: *Annali di Matematica Pura ed Applicata* 39.1 (Dec. 1955), pp. 357–379. ISSN: 1618-1891. DOI: 10.1007/BF02410779.

[Sha49] Claude E. Shannon. "Communication theory of secrecy systems." In: *Bell Systems Technical Journal* 28.4 (1949), pp. 656–715.

[Shi+07] Taizo Shirai, Kyoji Shibutani, Toru Akishita, Shiho Moriai, and Tetsu Iwata. "The 128-Bit Blockcipher CLEFIA (Extended Abstract)." In: *FSE 2007*. Ed. by Alex Biryukov. Vol. 4593. LNCS. Springer, Heidelberg, Mar. 2007, pp. 181–195. DOI: 10.1007/978-3-540-74619-5_12.

[Shi+11] Kyoji Shibutani, Takanori Isobe, Harunaga Hiwatari, Atsushi Mitsuda, Toru Akishita, and Taizo Shirai. "Piccolo: An Ultra-Lightweight Blockcipher." In: *CHES 2011*. Ed. by Bart Preneel and Tsuyoshi Takagi. Vol. 6917. LNCS. Springer, Heidelberg, Sept. 2011, pp. 342–357. DOI: 10.1007/978-3-642-23951-9_23.

[Sil00] John R Silvester. "Determinants of block matrices." In: *The Mathematical Gazette* (2000), pp. 460–467.

[Sim+15] Siang Meng Sim, Khoongming Khoo, Frédérique E. Oggier, and Thomas Peyrin. "Lightweight MDS Involution Matrices." In: *FSE 2015*. Ed. by Gregor Leander. Vol. 9054. LNCS. Springer, Heidelberg, Mar. 2015, pp. 471–493. DOI: 10.1007/978-3-662-48116-5_23.

[SS16a] Sumanta Sarkar and Siang Meng Sim. "A Deeper Understanding of the XOR Count Distribution in the Context of Lightweight Cryptography." In: *AFRICACRYPT 16*. Ed. by David Pointcheval, Abderrahmane Nitaj, and Tajjeeddine Rachidi. Vol. 9646. LNCS. Springer, Heidelberg, Apr. 2016, pp. 167–182. DOI: 10.1007/978-3-319-31517-1_9.

[SS16b] Sumanta Sarkar and Habeeb Syed. "Lightweight Diffusion Layer: Importance of Toeplitz Matrices." In: *IACR Trans. Symm. Cryptol.* 2016.1 (2016). http://tosc.iacr.org/index.php/ToSC/article/view/537, pp. 95–113. ISSN: 2519-173X. DOI: 10.13154/tosc.v2016.i1.95-113.

[SS17] Sumanta Sarkar and Habeeb Syed. "Analysis of Toeplitz MDS Matrices." In: *ACISP 17, Part II*. Ed. by Josef Pieprzyk and Suriadi Suriadi. Vol. 10343. LNCS. Springer, Heidelberg, July 2017, pp. 3–18.

[Sto16] Ko Stoffelen. "Optimizing S-Box Implementations for Several Criteria Using SAT Solvers." In: *FSE 2016*. Ed. by Thomas Peyrin. Vol. 9783. LNCS. Springer, Heidelberg, Mar. 2016, pp. 140–160. DOI: 10.1007/978-3-662-52993-5_8.

[Swa62] Richard G. Swan. “Factorization of polynomials over finite fields.” In: *Pacific J. Math.* 12.3 (1962), pp. 1099–1106. URL: http://projecteuclid.org/euclid.pjm/1103036322.

[Tie+15] Tyge Tiessen, Lars R. Knudsen, Stefan Kölbl, and Martin M. Lauridsen. “Security of the AES with a Secret S-Box.” In: *FSE 2015*. Ed. by Gregor Leander. Vol. 9054. LNCS. Springer, Heidelberg, Mar. 2015, pp. 175–189. DOI: 10.1007/978-3-662-48116-5_9.

[TLS16] Yosuke Todo, Gregor Leander, and Yu Sasaki. “Nonlinear Invariant Attack - Practical Attack on Full SCREAM, iSCREAM, and Midori64.” In: *ASIACRYPT 2016, Part II*. Ed. by Jung Hee Cheon and Tsuyoshi Takagi. Vol. 10032. LNCS. Springer, Heidelberg, Dec. 2016, pp. 3–33. DOI: 10.1007/978-3-662-53890-6_1.

[TP19] Quan Quan Tan and Thomas Peyrin. “Improved Heuristics for Short Linear Programs.” In: *IACR TCHES* 2020.1 (2019). https://tches.iacr.org/index.php/TCHES/article/view/8398, pp. 203–230. ISSN: 2569-2925. DOI: 10.13154/tches.v2020.i1.203-230.

[VGE15] Roel Verdult, Flavio D. Garcia, and Baris Ege. “Dismantling Megamos Crypto: Wirelessly Lockpicking a Vehicle Immobilizer.” In: *Supplement to the Proceedings of 22nd USENIX Security Symposium (Supplement to USENIX Security 15)*. USENIX Association, 2015, pp. 703–718.

[VSP17] Andrea Visconti, Chiara Valentina Schiavo, and René Peralta. *Improved upper bounds for the expected circuit complexity of dense systems of linear equations over GF(2)*. Cryptology ePrint Archive, Report 2017/194. http://eprint.iacr.org/2017/194. 2017.

[Wag99] David Wagner. “The Boomerang Attack.” In: *FSE’99*. Ed. by Lars R. Knudsen. Vol. 1636. LNCS. Springer, Heidelberg, Mar. 1999, pp. 156–170. DOI: 10.1007/3-540-48519-8_12.

[War94] William P. Wardlaw. “Matrix Representation of Finite Fields.” In: *Mathematics Magazine* 67.4 (1994), pp. 289–293. ISSN: 0025570X, 19300980.

[Wat+02] Dai Watanabe, Soichi Furuya, Hirotaka Yoshida, Kazuo Takaragi, and Bart Preneel. “A New Keystream Generator MUGI.” In: *FSE 2002*. Ed. by Joan Daemen and Vincent Rijmen. Vol. 2365. LNCS. Springer, Heidelberg, Feb. 2002, pp. 179–194. DOI: 10.1007/3-540-45661-9_14.

[WWW13] Shengbao Wu, Mingsheng Wang, and Wenling Wu. “Recursive Diffusion Layers for (Lightweight) Block Ciphers and Hash Functions.” In: *SAC 2012*. Ed. by Lars R. Knudsen and Huapeng Wu. Vol. 7707. LNCS. Springer, Heidelberg, Aug. 2013, pp. 355–371. DOI: 10.1007/978-3-642-35999-6_23.

[Wys09] Brecht Wyseur. “White-Box Cryptography.” PhD thesis. Katholieke Universiteit Leuven, 2009.

[Wys12] Brecht Wyseur. “White-Box Cryptography: Hiding Keys in Software.” In: *MISC HS 5 magazine*. 2012, pp. 65–72. URL: http://www.whiteboxcrypto.com/files/2012_misc.pdf.

[XZL14] Hong Xu, Yonghui Zheng, and Xuejia Lai. “Construction of perfect diffusion layers from linear feedback shift registers.” In: *IET Information Security* 9.2 (2014), pp. 127–135.

[Zha+16] Ruoxin Zhao, Baofeng Wu, Rui Zhang, and Qian Zhang. *Designing Optimal Implementations of Linear Layers (Full Version)*. Cryptology ePrint Archive, Report 2016/1118. http://eprint.iacr.org/2016/1118. 2016.

[ZWS17] Lijing Zhou, Licheng Wang, and Yiru Sun. *On the Construction of Lightweight Orthogonal MDS Matrices*. Cryptology ePrint Archive, Report 2017/371. http://eprint.iacr.org/2017/371. 2017.