PhD-FSTC-2017-24
The Faculty of Sciences, Technology and Communication

DISSERTATION

Defence held on 25/04/2017 in Belval

to obtain the degree of

DOCTEUR DE L’UNIVERSITÉ DU LUXEMBOURG

EN INFORMATIQUE

by

Léo Paul PERRIN
Born on the 20th of November 1990 in Lons-le-Saunier (France)

# CRYPTANALYSIS, REVERSE-ENGINEERING AND DESIGN OF SYMMETRIC CRYPTOGRAPHIC ALGORITHMS

## Dissertation defence committee

DR ALEX BIRYUKOV, dissertation supervisor
*Professor, Université du Luxembourg*

Dr Henri Gilbert
*HDR, Agence Nationale pour la Sécurité des Systèmes d'Information*

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

Dr Gregor Leander
*Professor, Ruhr-Universität Bochum*

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

To skipjacks, sponges, pollocks and whales;
to grasshoppers, butterflies and skippers.

# Abstract

In this thesis, I present the research I did with my co-authors on several aspects of symmetric cryptography from May 2013 to December 2016, that is, when I was a PhD student at the university of Luxembourg under the supervision of Alex Biryukov. My research has spanned three different areas of symmetric cryptography.

In Part I of this thesis, I present my work on *lightweight cryptography*. This field of study investigates the cryptographic algorithms that are suitable for very constrained devices with little computing power such as RFID tags and small embedded processors such as those used in sensor networks. Many such algorithms have been proposed recently, as evidenced by the survey I co-authored on this topic. I present this survey along with attacks against three of those algorithms, namely Gluon, Prince and Twine. I also introduce a new lightweight block cipher called Sparx which was designed using a new method to justify its security: the *Long Trail Strategy*.

Part II is devoted to *S-Box reverse-engineering*, a field of study investigating the methods recovering the hidden structure or the design criteria used to build an S-Box. I co-invented several such methods: a statistical analysis of the differential and linear properties which was applied successfully to the S-Box of the NSA block cipher Skipjack, a structural attack against Feistel networks called the *yoyo game* and the *TU-decomposition*. This last technique allowed us to decompose the S-Box of the last Russian standard block cipher and hash function as well as the only known solution to the *APN problem*, a long-standing open question in mathematics.

Finally, Part III presents a unifying view of several fields of symmetric cryptography by interpreting them as *purposefully hard*. Indeed, several cryptographic algorithms are designed so as to maximize the code size, RAM consumption or time taken by their implementations. By providing a unique framework describing all such design goals, we could design modes of operations for building any symmetric primitive with any form of hardness by combining secure cryptographic building blocks with simple functions with the desired form of hardness called *plugs*. Alex Biryukov and I also showed that it is possible to build plugs with an *asymmetric hardness* whereby the knowledge of a secret key allows the privileged user to bypass the hardness of the primitive.

# Acknowledgements

This section is my attempt at expressing my gratitude to all those who helped me through my four years as a PhD student, academically or otherwise.

First of all, I thank my supervisor Alex Biryukov. For accepting me as his student and for giving me the opportunity to work in his group of course, but also for his advice and guidance during my studies. He always pushed me to improve our results as far as possible and my work would not be half as meaningful without these encouragements.

I also thank Jean-Sébastien Coron, Henri Gilbert, Gregor Leander and Volker Müller for accepting to be in my jury; meaning they agreed to both go through the 380-odd pages of this thesis and to come all the way to Belval which, for those living outside Luxembourg, is no small journey!

None of the work presented in this thesis would have been possible without my co-authors who have all my gratitude: Alex Biryukov, Anne Canteaut, Patrick Derbez, Daniel Dinu, Sébastien Duval, Johann Großschädl, Dmitry Khovratovich, Yann Le Corre, Gaëtan Leurent, Aleksei Udovenko, and Vesselin Velichkov.

Beyond our work together, I also thank my colleagues for fun and interesting conversations around a coffee or a beer: Patrick Derbez, Daniel Dinu, Daniel Fehrer, Johann Großschädl, Dmitry Khovratovich, Yann Le Corre, Zhe Liu, Ivan Pustogarov, Arnab Roy, Sergei Tikhomirov, Aleksei Udovenko, Praveen Vadnala, Vesselin Velichkov and Srinivas Vivek.

I was fortunate to have had prior experience in research before the start of my PhD. I thank Céline Blondeau, my master thesis advisor, for teaching me how to do research, write papers and advising me to apply for this PhD position. She also introduced me to Boolean functions, an area in which I am still working. If it were not for her, I would probably not have done an academic career!

I thank Anne Canteaut for inviting me and welcoming me for a week in her group at Inria as well as for helping me with a grant application. Her knowledge in Boolean functions and the discussions we had during conferences and workshops contributed to several of my papers. I am looking forward to joining her group in a couple of months!

Unfortunately, a PhD thesis does not simply involve doing research and teaching. Thus, I thank Fabienne Schmitz for her help with administrative matters. My work was made possible by the *Fond National de la Recherche (FNR)* through the Core project Acrypt (ID C12-15-4009992). Many thanks to them. More generally, I also thank the university of Luxembourg and indeed the country of Luxembourg for providing such a stimulating work environment.

I thank the chairs of Esc'2015, the chairs of the Dagsthul seminar 16021 on symmetric cryptography, and the chairs of Esc'2017 for inviting me to their workshops. The opportunity to present my work to some of the best cryptographers lead to very fruitful discussions and collaborations.

I thank Daniel Dinu and Virginie Lallemand for checking parts of my thesis. I also give a special thank you to Brian Shaft for proof-reading all of this manuscript and finding a much too high number of typos! Virginie also helped me solve a crucial question: to whom/what should this thesis be dedicated?

As far back as I can remember, I always wanted to work in science. I thank my family and in particular my parents for always being supportive of my unquenchable curiosity. I also thank them for being present on the day of my defence despite the distance.

I thank Brian and Zezinha for Sundays that took my mind off of work through interesting conversations and delicious food, and the members of my martial arts club that took my mind off of work through physical exercise.

Finally, my work relied heavily on several open source projects whose developers have all my gratitude. In particular, I thank the authors of Sage [Dev16], LaTeX, Ubuntu and Emacs.

# Summary

# List of Figures

# List of Tables

# List of Algorithms

# Notations and Abbreviations

| | |
|---|---|
| $\oplus$ | Exclusive or, also called “XOR” |
| $\mathbb{F}_2$ | The set $\{0, 1\}$ |
| $\mathbb{F}_2^n$ | The set of the bitstrings of length $n$ |
| $\mathbb{F}_{2^n}$ | The finite field of size $2^n$ |
| $\mathrm{hw}(x)$ | The Hamming weight of the bitstring $x$ |
| $a \cdot b$ | The scalar product of the two elements $a$ and $b$ of $\mathbb{F}_2^n$ |
| $\lvert E \rvert$ | The number of elements in a set $E$ |
| $\#E$ | The number of elements in a set $E$ |
| $\Pr[\omega]$ | The probability of an event $\omega$ |
| $a \xleftarrow{\$} E$ | The fact that $a$ is drawn uniformly at random from a set $E$ |
| $\mathbb{Z}/n\mathbb{Z}$ | The ring of the integers modulo $n$ |

| | |
|---|---|
| ANF | Algebraic Normal Form |
| DDT | Difference Distribution Table |
| LAT | Linear Approximation Table |
| LFSR | Linear Feedback Shift Register |
| LSB | Least Significant Bit |
| LUT | Look-Up Table |
| MSB | Most Significant Bit |
| NLFSR | Non-Linear Feedback Shift Register |
| S-Box | Substitution-box (see Definition 8.0.1) |

| | |
|---|---|
| ASIC | Application-Specific Integrated Circuit |
| FSB | Federal Security Service (Russian spying agency) |
| IoT | Internet of Things |
| NSA | National Security Agency (American spying agency) |
| PoW | Proof-of-Work |
| RFC | “Request For Comments” |
| SCA | Side Channel Attack |

Chapter 1

# Introduction

In its broadest definition, *cryptography* is the science studying algorithms to ensure that information is protected from attackers. These attackers may try to read a message they are not supposed to have access to, in which case cryptographic algorithms must provide *privacy*. If they try to modify the message, these algorithms must ensure its *integrity*. Finally, if attackers try to impersonate one of the actors, algorithms must *authenticate* the rightful sender and receiver. *Cryptology* encompasses the work of both the cryptographers designing such algorithms and the cryptanalysts trying to find flaws in them. This field of research can be thought of as an arms race.

## 1.1 Modern Cryptography

While cryptography is an ancient topic of research, the algorithms used until the XXth century were not designed to resist the attacks of cryptanalysts with significant computing power and relied on simple principles. The most famous example is the Caesar cipher which encrypts a text by replacing every letter by the one 3 steps above it in the alphabet, so that *A* becomes *D*, *B* becomes *E*, etc.

More recently, an even more famous use of cryptography was the Enigma machine used, among others, by the armed forces of Nazi Germany during WW-II. The encryption provided by this device was broken by several teams of cryptanalysts. First, Polish mathematicians lead by Rejewski reverse-engineered the details of this machine and found the first attacks against it. Later, Alan Turing and the cryptanalysts of Bletchley Park (UK) built upon this work and found cryptanalyses against more recent versions of Enigma. In fact, the machines used to break Enigma as well as other electro-mechanical encrypting devices of the time like the Japanese *Purple* or the German *Lorentz machine* were the prototypes of the first ever computers.

Nowadays, if cryptography is still a tool of crucial importance for the military, it is in fact used more often in a civlian context. Indeed, as the amount of information shared is reaching new peaks every years thanks to the internet, the ability to control who can access it has an importance which would be hard to overstate.

A formal description of the aim of modern cryptography is given in Section 1.1.1 while the types of attacks that must be prevented are detailed in Section 1.1.2 (p. 3).

### 1.1.1 Purpose

Formally, the aim of modern cryptography is threefold. As is traditional, I will use a hypothetical communication between Alice and Bob to explain them. Eve is the *attacker*[1], that is, someone trying to interact malevolently with the communication between Alice and Bob. Note that neither Alice, Bob nor Eve have to be physical people; instead they may be servers or institutions.

- *Confidentiality.* It must be impossible for Eve to obtain any meaningful information from the communication she intercepts. For example, if Eve works for the intelligence agency of a repressive government, she should not be able to figure out what a journalist, Alice, is discussing with her source, Bob.
- *Authentication.* It must be impossible for Eve to send a message to Bob and convince him that Alice sent it. If Alice receives a message from her friend Bob suggesting to pick her up by car, it is crucial to ensure that this message was indeed sent by Bob and not by a thief, Eve.
- *Integrity.* It must be impossible for Eve to tamper with a message. If Alice is sending a message to her bank (Bob) to wire some money to a friend, then it must be impossible for a thief (Eve) to change the message to have her be the recipient of the money instead.

More recently, the following two goals have emerged.

- *Anonymity.* It must be impossible for Eve to figure out who Alice and Bob are. To re-use the example where Alice is a journalist, it is crucial that Bob can hide his identity even if Eve is capable, like spying agencies are, of intercepting vast amount of internet communications. The aim here is to protect *meta-data*: rather than the content of the conversation, what is protected is the identity of its actors.
- *Proof-of-Work* It must be impossible to solve a problem without investing a tunable amount of computation into its resolution. Such objects are used for example to build crypto-currencies such as Bitcoin [Nak08].

These last two goals require public-key cryptography — which will be defined below — or higher level protocols, both of which are out of the scope of this thesis. However, in practice, they must combine those with secret-key cryptography.

The distinction between public- and private-key cryptography is straight-forward. In *secret-key* cryptography, the key used to operate on the plaintext and on the ciphertext are the same. In *public-key* cryptography, these are different. Public-key cryptography is easier to use in practice: the key for encrypting plaintexts (called *public key*) can be freely distributed while only the decryption key (called *private key*) is kept secret. In contrast, secret-key algorithms require that the key is only known by the entities exchanging messages, which leads to complex key management problems.

[1] I voluntarily chose specific and realistic examples of attackers in this introduction. To quote Phillip Rogaway's essay, *The Moral Character of Cryptography* [Rog15]: "When we try to explain [cryptography] with cartoons and cute narratives, I don't think we make our contributions easier to understand. [...] Stop with the cutesy pictures. Take adversaries seriously."

Unfortunately, public-key algorithms such as RSA are orders of magnitude slower than secret-key algorithms. Therefore, in practice, a combination of both is most most commonly used. In a nutshell, public-key algorithms are used by Alice and Bob to agree on a secret key which they then use to secure their communications.

### 1.1.2 Kerckhoffs' Principle and Attack Models

*Kerckhoffs' law* is a strong guideline that must be followed when designing a cryptosystem. It is the second of a series of six requirements for a cryptosystem to be secure which was published by a cryptographer, August Kerckhoffs, in an article published in 1883 in the "Journal des Sciences Militaires" [Journal of Military Sciences] [Ker83]:

> 2. Il faut qu[e le système] n'exige pas le secret, et qu'il puisse sans inconvénient tomber entre les mains de l'ennemi ;

which can be translated into the following.

> 2. It must be that [the system] does not require secrecy and does not cause problems should it be stolen by the enemy ;

While almost 134 years old, this principle is still as important today. Indeed, many cryptosystems have been designed, in particular in the industry, that relied on the secrecy of the algorithm itself to be secure. And yet, the "Kindle Cipher" used by Amazon for a DRM scheme was reverse-engineered and then broken [BLR13]. Similarly,[2] the Keeloq algorithm used among other things for the remote car keys of many manufacturers has been attacked successfully by multiple teams [EKM+08, CBW08, ABM+12].

Conversely, the security of all cryptographic algorithms published in the open literature, be they academic papers or standards, rely exclusively on the secrecy of the key to provide security. Indeed, attackers are free to read the specification of the algorithms online. Cryptographers are then free to study these algorithms and assess their quality, thus allowing us to discard bad algorithms and to increase our confidence in the good ones.

In this context, an *attack* against a cryptosystem is a method which recovers some information about the plaintext and/or about key. These attacks can be sorted according to the information which the attackers are considered to have access to.

- *Known Plaintext.* In a *known plaintext* attack, the attacker has access to several pairs consisting of a plaintext and the corresponding ciphertext. However, it is impossible to choose which plaintexts are encrypted.

- *Chosen Plaintext.* If the attacker is allowed to first chose which plaintexts are encrypted and then receive all the corresponding ciphertexts, the attack is a *chosen plaintext* attack. It is a more contrived setting than the known plaintext case.

- *Related-Key.* In this setting, the attacker can encrypt some plaintexts (either chosen or not) under two different keys $k$ and $k'$ which, while unknown, are linked by a specific relation such as, for example, $k \oplus k' = c$ for some known constant $c$. It is a more contrived setting than the known and chosen plaintext cases.

[2] A list of such industry-designed ciphers of questionable quality is provided in Section 2.2.1 (p. 34).

All these attack settings are in the *black-box model*; that is, the attacker cannot have access to the internal state of the encryption function. Only its input, output and key (in the related-key setting and only to some extent) can be controlled. However, this assumption is not always true. For example, a *side-channel attack* is a technique that can recover some information about the internal state of the primitive by, for example, studying the power consumption of a device performing the encryption. The corresponding model is called *gray box* because the adversary has partial access to the internal state.

The most extreme case is that of *white-box* cryptography in which the adversary is has full access to the encryption algorithm, including its key. In this context, the security target is different. It is obviously impossible to prevent the adversary from decrypting the ciphertext of their choosing. However, it may be possible to prevent her from recovering a short description of the encryption algorithm. This topic is more thoroughly discussed in Chapter 15 (p. 303).

## 1.2 Symmetric Cryptography

Symmetric cryptography deals with primitives that use a unique key for all their operations — if they use a key at all. The algorithms that fall in this category are described in Section 1.2.1. This stands in contrast to *public key* algorithm which use one key to encrypt and another to decrypt. A quick overview of the different attacks that can target those is provided in Section 1.2.2.

### 1.2.1 Basic Symmetric Algorithms

Symmetric algorithms can provide confidentiality, integrity and authentication. The first can be provided using either a *block cipher* or a *stream cipher*, which are described in Sections 1.2.1.1 (p. 4) and 1.2.1.2 (p. 6) respectively. Integrity and authentication are obtained using a *Message Authentication Code (MAC)* which is usually built using either a block cipher or a *hash function*. The latter is presented in Section 1.2.1.3 (p. 7) and the MAC is described in Section 1.2.1.4 (p. 8). Finally, *authenticated ciphers* which provide confidentiality, integrity and authentication at once are mentioned in Section 1.2.1.5 (p. 8).

For each algorithm, we briefly present the most common ways of building them.

#### 1.2.1.1 Block Ciphers

A block cipher is a family of permutations operating on blocks of $n$ bits which is indexed by a key of $k$ bits. Usually $n = 64$ or $n = 128$, the latter offering better security against generic attacks but the former allowing a smaller memory use more suitable for constrained devices, as discussed in Chapter 2.1.2.

The value of $k$ determines the resilience of the cipher to the most basic attack: brute-force. It consists simply in enumerating all possible keys until the correct one is found. Hence, a large enough value of $k$ is needed. While the first public block cipher, the DES, had $k = 56$ bits of key material,[3] modern ciphers such as the AES use at least $k = 128$ and often $k = 256$.

[3] This value was chosen after discussion with the National Security Agency (NSA), an American government agency. However, it was considered too small by Hellman, Diffie, Merkle and others as early as 1976 [HMS+76].

Since block ciphers operate only on blocks of fixed size, they must be combined with *modes of operations* to encrypt plaintexts of arbitrary size.

In practice, block ciphers are built by iterating a simple transformation called *round* multiple times. The *round functions* must depend on the key. They can be built using different well-known structures described below: a Substitution-Permutation network (SPN), a Feistel network or a Lai-Massey structure. A high level view of each of them is provided in Figures 1.1, 1.2 and 1.3 respectively.

**Substitution-Permutation Network (SPN).** These structures alternate a linear and a non-linear layer. The non-linear layer usually consists of the parallel application of smaller permutations operating on a subset of the internal state called *S-Boxes*, as can be seen in Figure 1.1. These components are the topic of an extensive study in Part II of this thesis. The diffusion layer consists of a linear operation mixing the output of the different S-Boxes. The round key can simply be XORed into the internal state of the cipher.

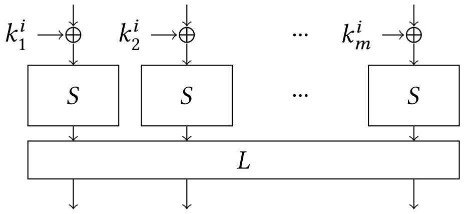


Figure 1.1: An SPN round.

The most prominent SPN-structured cipher is the current NIST standard called the *Advanced Encryption Standard (AES)* [DR02]. The block cipher PRINCE analyzed in Chapter 4 (p. 73) is also an SPN. The SPARX cipher introduced in Chapter 7 (p. 117) is a variant of an SPN with bigger S-Boxes and a simpler linear layer.

Given the code book of an SPN, it is possible to recover the S-Boxes and the linear layers used even if their full descriptions are secret. Such structural attacks are discussed in Chapter 11 (p. 207).

**Feistel Network.** This structure was first introduced by Horst Feistel during the design of the Lucifer block cipher at IBM. Eventually, this line of research lead to the design of the most prominent example of a Feistel network: the DES [U.S99]. The round function of a Feistel network uses a keyed Feistel function $F_k$ and maps an internal state $(x_L, x_R)$, for $x_L$ and $x_R$ in $\mathbb{F}_2^{n/2}$, to a value $(y_L, y_R)$ defined as:

$$\begin{cases} y_L = x_R \\ y_R = x_L \oplus F_k(x_R) \end{cases},$$

as summarized in Figure 1.2. Such round functions are easy to invert, meaning that implementing both encryption and decryption is not much more expensive than implementing only encryption. It is not necessary that $F_k$ is a permutation. In fact, it is not a permutation for the DES. It is also possible to replace the XOR by a modular addition.

There are many variants of this structure. In an *unbalanced Feistel network*, the two branches are of different sizes. In a *Misty-like structure*, which is named after

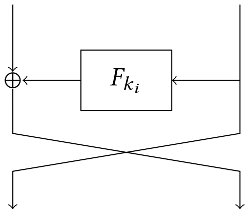


Figure 1.2: A Feistel round.

the Misty block cipher [Mat97]), the Feistel function is applied on a branch rather than on a copy of a branch. In this case, the function has to be a permutation. This is further generalized in Skipjack [U.S98] where the round function uses four 16-bit branches where only two of them undergo a Misty-like transformation, as detailed in Section 9.2.1 (p. 168). Finally, a *generalized Feistel network* as introduced by [Nyb96] consists of $2m$ branches grouped into $m$ groups of two branches. Each group goes through a regular Feistel round and then the $2m$ branches are shuffled. For example, the block cipher Twine studied in Chapter 5 (p. 89) is a generalized Feistel network.

Chapter 10 (p. 181) is devoted to structural attacks against Feistel networks, that is, attacks that work regardless of the definition of the Feistel functions.

**Lai-Massey.** The Lai-Massey structure was first used in the design of the Idea block cipher [LM91]. This structures splits the state into two halves $x_L$ and $x_R$. Then, $y = f(x_L \oplus x_R)$ is computed for some function $f$ — it does not have to be a permutation. Finally, the initial values are replaced as follows: $x_L \leftarrow x_L \oplus y$ and $x_R \leftarrow x_R \oplus y$. This type of round is invertible because $x_L \oplus x_R$ is left unchanged by this computation, meaning that the value of $f(x_L \oplus x_R)$ can be evaluated from both the input and the output of the round. Some function has to be applied on the branches in between the round to break some simple patterns. In Figure 1.3, this role is played by the $\sigma$ function which may or may not be linear.

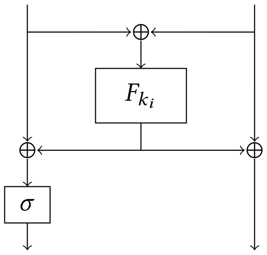


Figure 1.3: A Lai-Massey round.

#### 1.2.1.2 Stream Ciphers

A *stream cipher* simulates a *One-Time Pad* (otp), an encryption algorithm which consists simply in xoring the plaintext with a key of the same length. This cipher is perfectly secure in the sense that a ciphertext reveals no information whatsoever about the plaintext apart from its length. However, this security collapses if the key is re-used, which makes key management too difficult in practice. To remedy this issue, stream ciphers generate a pseudo-random stream of bits called *keystream* from a shorter secret key. While this construction looses the security proof of the OTP, it is far easier to use in practice.

The *keystream* generated must be such that it is impossible, given parts of it, to recover either the key used to initialize the stream cipher or other parts of the keystream. Decryption is identical to decryption. Except for some specific designs, stream ciphers only provide privacy, not authentication or integrity.

Historically, many stream ciphers have been built using Linear Feedback Shift Registers (LFSR) and Non-Linear Feedback Shift Registers (NLFSR). Such stream ciphers are particularly attractive where a hardware implementation is needed as illustrated for instance by the stream cipher Trivium [Can06].

It is also common to build stream ciphers out of block ciphers using the counter mode, that is, by encrypting an increasing counter with a block cipher.

#### 1.2.1.3 Hash Functions

A *hash function* maps a string of arbitrarily length to a string of fixed length. A *cryptographic* hash function must additionally satisfy the following properties.

- *Collision resistance*: impossible to find a pair of messages $(x, x')$ such that $H(x) = H(x')$ in time less than $2^{n/2}$.

- *Preimage resistance*: given $d$, impossible to find a message $x$ such that $H(x) = d$ in time less than $2^n$.

- *Second preimage resistance*: given $d$ and $x$ such that $H(x) = d$, impossible to find a message $x' \neq x$ such that $H(x') = d$ in time less than $2^n$ (Merkle-Damgård) or $2^{n/2}$ (sponge construction).

Such a function can be used for different purposes. The usual example is document integrity verification. If the hash of a downloaded document matches the hash computed using the original document, then the two documents are very likely to be identical. Hash functions can also be used as building blocks of higher level cryptographic constructions. For example, HMAC [KBC97] is a message authentication code (see Section 1.2.1.4 (p. 8)) built using a hash function. The two most prominent high level structures for hash functions are the Merkle-Damgård construction and the more recent sponge construction.

**Merkle-Damgård.** This structure relies on a compression function $C : (x, y) \rightarrow z$ where $x$ and $z$ are $n$-bit long and $y$ is $m$-bit long. This compression function must be collision resistant, meaning that finding $(x, y)$ and $(x', y')$ such that $C(x, y) = C(x', y')$ must be difficult. Such compression functions can be built from block ciphers. Hashing then consists of decomposing the message into blocks of $m$ bits, initializing an $n$-bit variable $h_0$ with a fixed value and then computing $h_{i+1} = C(h_i, m_i)$. The last value $h_\ell$ is the digest of the message $m$. To prevent simple attacks, it is necessary to pad the message e.g. by appending its length to it.

The most prominent Merkle-Damgård-based hash functions are those of the SHA-2 family [U.S15a].

**Sponge.** The sponge construction [BDPVA07] is characterized by its *rate* $r$, its *capacity* $c$ and its update function $g$. It is based on an internal state $x$ of size $r+c$ where, at each round, $r$ bits of the padded message are XORed. Then the sponge alternates the application of $g$ function with the message injection until the message has been entirely *absorbed*. The digest is then *squeezed* by extracting $r$ bits of the internal state and applying the update function to the internal state again. This is repeated as many times as necessary to obtain a digest of desired length. A representation of a sponge is given in Figure 1.4. The sponge-based hash function is indifferentiable from a random oracle in the random-function model up to $2^{c/2}$ queries to $g$ [BDPV08]. If $g$ is a permutation, the sponge is a *permutation sponge* or P-sponge. If $g$ is not a permutation, the sponge is called *transformative sponge* or T-sponge.

The most prominent example of sponge construction is Keccak [BDPA15] which won the NIST SHA-3 competition. Therefore, the SHA-3 hash function is based on this algorithm [U.S15b].

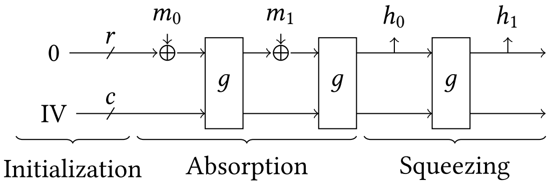


Figure 1.4: A sponge-based hash function.

#### 1.2.1.4 Message Authentication Codes

A message authentication code (MAC) is a family of keyed functions $F_k$ mapping an arbitrarily long plaintext to a *tag* of fixed length. A MAC $F_k$ should be such that it is computationally infeasible for an attacker to construct a plaintext $p$ and a tag $\tau$ such that $\tau = F_k(p)$, unless $k$ is known.

MACs can be built in many ways, including using block ciphers and sponge constructions.

#### 1.2.1.5 Authenticated Ciphers

An authenticated cipher encrypts a plaintext into a ciphertext combined with a tag. It provides at once privacy, integrity and authentication. It can be built by combining a block cipher and a MAC with an appropriate mode of operation. It can also be constructed using a sponge.

### 1.2.2 What is an Attack?

The aim of cryptanalysts is to find *attacks* against these algorithms. But what do we call an attack? And what is the purpose behind them?

First, I describe the two components of most attacks presented in the literature: *distinguishers* which correspond to properties of an algorithm that differentiate it from a random one and *key recoveries* which identify some information about the key used using such a distinguisher. These are described respectively in Sections 1.2.2.1 (p. 9) and 1.2.2.2 (p. 10). In each case, examples are provided.

The goal behind these attacks is two fold. They may of course be used to actually attack algorithms as they are used in practice. However, more often than not, they are used as an academic tool to assess the security offered by an algorithm. This second duality is explored in Section 1.2.2.3 (p. 11).

#### 1.2.2.1 Distinguisher

A block cipher is supposed to be indistinguishable from a random permutation. That is, given a permutation picked uniformly at random from the set of all permutations (Pseudo-Random Permutation, PRP) and a block cipher operating with an unknown key, an adversary must not be capable of successfully telling which is which with a probability higher than 1/2.

A property allowing the attacker to successfully make the difference between a PRP and a given block cipher is called a *distinguisher*. There are several well-known methods for building such objects.

**Differential Distinguisher.** For any pair of non-zero $n$-bit elements $(\delta_{\text{in}}, \delta_{\text{out}})$, the probability

$$\Pr[P(x \oplus \delta_{\text{in}}) \oplus P(x) = \delta_{\text{out}}]$$

has an expected value of $2^{-n}$ if $P$ is picked uniformly at random from the set of all permutation and $x$ is picked uniformly at random from the set of all $n$-bit blocks.

If this probability is higher than $2^{-n}$ when $P$ is replaced by a block cipher $E_k$ with a key picked uniformly at random, then we have a differential distinguisher. The pair $(\delta_{\text{in}} \rightsquigarrow \delta_{\text{out}})$ is called a differential. By identifying differentials $(\delta_i \rightsquigarrow \delta_{i+1})$ with high probability $p_i$ for the round function of a block cipher, we can create a high probability *differential trail* $\delta_0 \rightsquigarrow \delta_1 \rightsquigarrow \dots \rightsquigarrow \delta_r$ covering $r$ rounds. We typically assume that the round keys are independent,[4] and thus that the probability of $\delta_i \rightsquigarrow \delta_{i+1}$ and $\delta_j \rightsquigarrow \delta_{j+1}$ are independent for $i \neq j$. In this case, the differential $(\delta_0 \rightsquigarrow \delta_r)$ holds with probability at least $\prod_{i=0}^{i-1} p_i$. The attacks presented in Chapters 3 (p. 55), 4 (p. 73) and 5 (p. 89) as well as Section 10.3 (p. 186) can be seen as particular cases of differential attacks.

This principle can be generalized. For example, it may be possible to identify a class of patterns $(\delta^i_{\text{in}} \rightsquigarrow \delta^i_{\text{out}})$ for $i \leq t$ such that some bits of $\delta^i_{\text{in}}$ and/or $\delta_{\text{out}}$ are equal to 0 for all $i$, the other bits taking different values. Such a pattern is called a *truncated differential*. It was introduced by Knudsen in [Knu95] and Section 5.3 (p. 93) presents such distinguishers for the lightweight block ciphers Twine [SMMK13], LBlock [WZ11] and LBlock-s [ZWW$^+$14].

A differential $(\delta_{\text{in}} \rightsquigarrow \delta_{\text{out}})$ with probability zero can also be used as a distinguisher. This is leveraged in *impossible differential attacks*. One of the first of these cryptanalyses targeted the block cipher Skipjack [BBS05, U.S98][5] because of its poor diffusion.

Finally, it is possible to use two short differential trails covering $r$ and $r'$ rounds respectively to attack $r + r'$ rounds of a primitive. If the first differential trail has probability $p$ and the second has probability $p'$, then we can find two right pairs for

[4] This assumption is called the *Markov assumption*. It is not true in practice because all round keys are derived from a comparatively small master key, meaning that there must be some dependency between them. Still, this assumption yields results that can be experimentally verified, so that it is usually a safe bet in practice.

[5] Skipjack itself is described in more details in Section 9.2.1 (p. 168).

each with probability about $p^2p'^2$ using a *boomerang attack* as first introduced by Wagner in [Wag99].

**Linear Distinguisher.** Let $a \cdot b$ denote the scalar product of two vectors $a$ and $b$ of $\{0,1\}^n$, so that $a \cdot b = \bigoplus_{i=0}^{n-1} a_i b_i$. If $P$ is a PRP, the probability

$$\Pr[a \cdot x \oplus b \cdot P(x) = 0]$$

is close to $1/2$ for any pair of non-zero elements $(a,b)$. This probability is the probability of the *linear approximation* $(a \rightsquigarrow b)$.

The quantity $\epsilon(a,b) = \Pr[a \cdot x \oplus b \cdot P(x) = 0] - 1/2$ is the *bias* of a linear approximation. If it is too high, the permutation can be distinguished from a PRP. In order to identify a linear approximation with a high bias, a linear trail can be constructed in a fashion analogous to a differential trail, as was done in the first linear attack against the DES [Mat94].

Much like in the differential case, several linear trails can be used at the same time to form a so-called *linear hull*. On the other hand, a linear approximation with a bias equal to 0 can be used in a so-called *zero-correlation attack*.

**Integral and Zero-Sum Distinguishers.** Let $f_k : \mathbb{F}_2^m \rightarrow \mathbb{F}_2^n$ be a keyed function. It could for example correspond to some output bits of a block cipher. The sum of $f_k(x)$ for all $x$ in some subset $\mathcal{V}$ of $\mathbb{F}_2^m$ may contain some artifacts because of the structure of $f_k$. In particular, if we can identify a vector space $\mathcal{V}$ strictly included in $\mathbb{F}_2^m$ such that, for any $k$,

$$\bigoplus_{x \in \mathcal{V}} f_k(x) = 0,$$

then we have identified a *zero-sum distinguisher*. A zero-sum distinguisher can correspond to an *integral distinguisher* which is obtained by identifying a set $\mathcal{V}$ of dimension $v$ such that the quantity

$$\text{Preimage}(z) \quad = \quad \#\{x \in \mathcal{V}, f_k(x) = z\}$$

is constant for all $k$ and all $z$. That is, all elements in the range of $f_k$ restricted to $\mathcal{V}$ have exactly $2^{n-v}$ preimages.

Zero-sum distinguishers can be found using the algebraic degree of a function and are particularly useful to attack secret SPN structures, as explained in Chapter 11 (p. 207). Integral distinguishers are related to zero-correlation distinguishers, as first shown in [BLNW12]. This relation is exploited in Section 12.2.1 (p. 228) to identify visual patterns in a graphical representation of the linear properties of an S-Box.

#### 1.2.2.2 Key Recovery

The methods presented in the previous section merely allow an attacker to figure out whether a black box is a given keyed algorithm or an idealised object picked uniformly at random. However, such properties can be used in a *key recovery* attack to actually identify some bits of the key used.

The general idea to attack a block cipher with $r$ rounds is to use a distinguisher on $r-1$ rounds, brute-force the subkey used in the last round and, for each guess, (partially) decrypt the ciphertext to know the (partial) internal state of the cipher at round $r-1$. If a subkey is such that the internal state at round $r-1$ is coherent with the distinguisher, then it is probably the right one. If not, it can be discarded. In practice though, this analysis is much more sophisticated and involves non-trivial statistical consideration, as can be seen in the multiple truncated differential attack presented in Section 5.3 (p. 93).

Alternatively, the key can be recovered by solving a system of equations. This resolution can for example be done with the help of a SAT-solver, as shown in Section 4.2 (p. 76). In this case, an "equation" — in fact a CNF formula — involving variables corresponding to plaintext, ciphertext and key bits is generated in such a way that the only solutions correspond to valid encryptions. By fixing the variables corresponding to the input and output to their values obtained via known plaintexts, we can solve a system where the only unknowns are the key bits. In practice, the resolution of such a system is impractical but there are heuristics based on differential distinguishers allowing a far more efficient resolution, as shown in Section 4.2 (p. 76) and in [DEKM17]. Linear systems can also be derived and solved, as in the structural attacks against SPNs presented in Chapter 11 (p. 207).

#### 1.2.2.3 Security Analysis Versus Practical Attack

The literature in symmetric cryptography consists mostly in the presentation of algorithms, the theoretical analysis of their components and *attacks*. The term "attack" may be misleading for readers who are not familiar with cryptography in the sense that these attacks are usually wildly impractical because of their tremendous complexity in terms of time, memory needed or amount of plaintext/ciphertext pairs available.

The aim of such attacks is not really to lay out a strategy which can be used by an attacker to recover the secret key. Rather, they illustrate structural observations. For example, in Section 5 (p. 89), I present a truncated differential attack against Twine. It is far too inefficient to be used by anyone but it shows that the strange behavior of the linear layer of this cipher allows a cryptanalyst to find better attacks than those found by the designers of the block cipher who overlooked this property. Therefore, these attacks should not be thought of as paving the way towards practical cryptanalysis but as security analyses studying whether or not the designers of an algorithm considered all attack vectors.

For example, although the attacks against the full-round lightweight block ciphers Klein [GNL11] and Picaro [PRC12] of Lallemand *et al.* [LN15, CLNP16] have too high a complexity to be practical. Nevertheless, the fact that attacks targeting full-round primitives exist is sufficient to compromise the claims of the designers of these algorithms. More generally, why use an algorithm vulnerable to as yet impractical attacks when there are other algorithms which can withstand them.

In accordance with the famous maxim "attacks only get better", an impractical attack submitted at a given time may later be improved. In this case, the mere security analysis is gradually improved into a practical attack. The successive attacks against the hash function Sha-1 are a good example of this.

## 1.3 Thesis Outline

In this thesis, I present the work I have done with my co-authors during my studies at the university of Luxembourg, under the supervision of Alex Biryukov. This section is a detailed summary of its content.

### 1.3.1 Part I - On Symmetric Lightweight Cryptography

Part of my work has been devoted to *lightweight cryptography*, that is, cryptography intended to be used on very constrained devices (such as RFID tags) and microcontrollers of the type used for the Internet of Things. It is presented in Part I of this thesis. The main result of this part is presented in its last two chapters.

**Main Results of Part I.** *First, the field of lightweight cryptography should be split into two distinct areas: ultra-lightweight cryptography — aimed at the most constrained devices — and pervasive cryptography — targeting more sophisticated IoT devices connected to global networks such as the internet.*

*Second, it is possible to build an ARX-based primitive which is provably secure against single trail differential and linear cryptanalysis.*

#### 1.3.1.1 Overview of Lightweight Cryptography

**Chapter 2.** To better understand this field of study, I made a thorough analysis of the literature on this topic. It encompasses both academic papers and standard specifications. A part of this review has already been made available online [BP14a]. A summary of the lightweight algorithms published published by academics is provided in Figure 1.5.

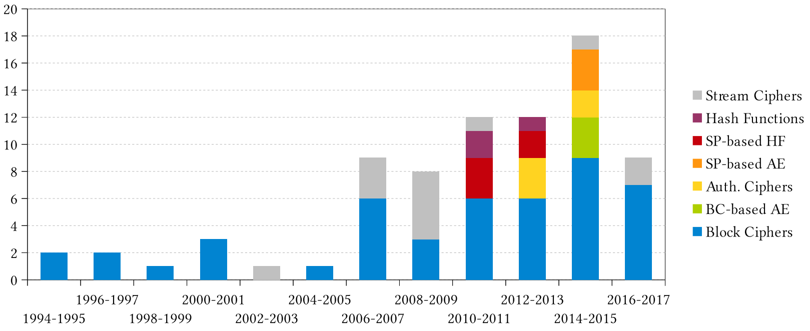


Figure 1.5: Lightweight algorithms published by academics. If an authenticated cipher is based on a block cipher, it is listed as a BC-based AE. Similarly, algorithms based on sponges are shown as such.

Informed by this survey, I argue that the field of lightweight cryptography is in fact too large and that it should be split into two distinct areas. The first, *ultra-lightweight cryptography*, would deal specifically with the lightest hardware platforms such as passive RFID tags. The second, *pervasive cryptography*, concerns algorithms intended to run on "things" connected to a global network such as the internet and equipped with a low-power micro-processor or a micro-controller.

The content of this chapter is joint work with Alex Biryukov and Daniel Dinu. It was also greatly enhanced by the feedback from the audience of ESC'17 where a first version of its content was presented.

#### 1.3.1.2 Cryptanalysis of Lightweight Algorithms

**Chapter 3** The family of sponge-based lightweight hash function Gluon was introduced by Berger *et al.* in [BDM+12]. It is the only sponge I am aware of where the transformation is not a bijection. First, we found a simple model quantifying the loss of information incurred by the iteration of a fixed non-bijective function $g$. It depends on the statistical distribution of the number of solution $x$ of $f(x \oplus y) \oplus f(y)$ taken over all $y$ which corresponds to its Collision Probability Spectrum (cps). The shrinkage of $g^i(\mathcal{S})$ as $i$ increases is illustrated in Figure 1.6.

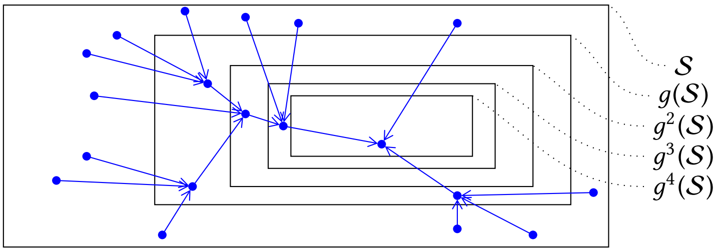


Figure 1.6: Collision trees and output shrinkage of an iterative non-injective function $g$ operating on $\mathcal{S}$.

For sponge-based hash functions, this shrinkage can be exploited in a second preimage attack targeting messages ending with multiple repetitions of an identical block when the rate of the sponge is low. In the specific case of Gluon-64, we devised an algorithm relying on a sat-solver to estimate its cps which turns out to be far worse than that of a random function. This property, combined with the low rate used, makes Gluon-64 vulnerable to the generic second preimage search we found.

The content of this chapter is based on [PK15], which I co-wrote with Dmitry Khovratovich.

**Chapter 4** Prince is a lightweight block cipher presented in [BCG+12] which is optimised for low-latency. Its designers challenged cryptographers to find practical against reduced-round versions of this cipher. I found several such attack using a sat-solver for key recovery. The first simply encodes known plaintext-ciphertext pairs into a cnf formula and solves for the key. A significant speed up is provided by exploiting zero differences and their probability 1 propagation. Then, an attack against 6 rounds is presented which combines a differential attack with a key recovery performed with the sat-solver. This differential attack is peculiar in that it does not use pairs of plaintexts but *families* of pairs built in such a way that all pairs in the same family must follow the differential trail if one does.

Some specific properties of the cycle structure of Prince are also discussed. Those are caused by the very specific structure of Prince.

The content of this chapter was published along with attacks by Patrick Derbez in [DP15].

**Chapter 5** The lightweight block ciphers Twine-80 and Twine-128 introduced in [SMMK13] use the same round function based on a generalized Feistel network. To improve diffusion, the designers used a sophisticated permutation of the branches instead of a simpler and more classical rotation.

This improved layer leads to an unexpected behavior of the diffusion whereby the 16 branches can be divided into two groups of 8 branches. During 3 out of 4 rounds, these sets only interact with themselves. However, during the fourth round, no information is exchanged within each set; instead, information flows from one set to the other. At the end of the fourth round, the two sets are swapped and will undergo a similar separation for the next 3 rounds.

We deduced high probability truncated differentials from this observation. Those can be used to attack a round-reduced version of Twine and to identify high probability differentials by clustering all differential trails that "fit" in those truncated trails.

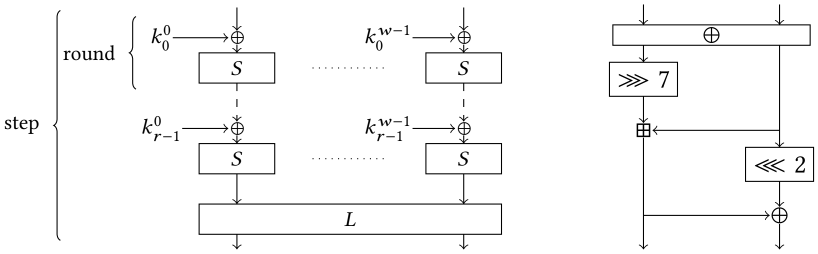
The content of this chapter is based on a joint work with Alex Biryukov and Patrick Derbez [BDP15].

#### 1.3.1.3 Designing a New Family of Lightweight Block Ciphers

The following two chapters present a design strategy and a family of lightweight block cipher that implement it which I co-designed with Alex Biryukov, Daniel Dinu, Johann Großschädl, Aleksei Udovenko and Vesselin Velichkov. Those results were first presented in [DPU+16a].

**Chapter 6.** It is possible to design a block cipher in such a way that its resilience against single trail differential and linear attacks can be proved. This property was one of the main selling points of Rijndael which eventually won the Aes competition. Unfortunately, the strategy used by its designer cannot be adapted to design arx-based ciphers, that is, ones that can be implemented using only modular Addition, Rotation and xor.

We have found a new method to design (possibly arx-based) block ciphers in such a way that it is provably secure against single trail differential and linear attacks. We call it the *Long Trail Strategy*. It works by combining two elements. The first is a transformation consisting of the parallel application of $r$ rounds of a large but rather weak S-Box which is secure after $2r$ rounds. The second is a linear layer which merely copies some of its inputs to its outputs. Combining the parallel calls to the S-Box with a linear layer application yields a *step*, as shown in Figure 1.7a. The encryption then consists of several such steps. Because of this structure, the input of half of the S-Boxes effectively go through $2r$ rounds of the simple S-Box. If the S-Box is chosen well, this ensures a high level of security.



(a) A cipher structure for the long trail strategy.

(b) The Speckey transformation $S$.

Figure 1.7: The components of the block ciphers of the Sparx family.

This behavior can be exploited to formally prove the security of such a block cipher against single trail differential and linear cryptanalysis.

We also found another structure we called Lax combining a modular addition with arbitrary linear functions. Block ciphers built using it can be proved secure against single trail differential attacks using a simple mathematical argument but security against linear attacks remains an open problem.

**Chapter 7.** Using the long trail strategy and the structure summarized in Figure 1.7a, we designed a family of lightweight block ciphers called Sparx. There are three instances, Sparx-64/128, Sparx-128/128 and Sparx-128/256 where Sparx-n/k encrypts $n$-bit blocks with a $k$-bit key. The transformation $S$ used, Speckey, is described in Figure 1.7b.

Because of their construction, the Sparx ciphers are the first arx-based block ciphers designed to be provably secure against single trail differential and linear attacks. Furthermore, their implementations on micro-controllers are quite efficient and compare favourably with the state of the art.

### 1.3.2 Part II - On S-Box Reverse-Engineering

With my co-authors, we pioneered the field of S-Box reverse-engineering; that is, the study of the methods that can be used to recover the hidden design criteria or the hidden structure used to build an S-Box using only its look-up table. This part describes the tools we developed for this purpose and the results we obtained using them. While our aim was S-Box reverse-engineering, these methods have applications beyond this area. They can be used to attack some white-box constructions or to derive new mathematical results.

Unlike in the previous part, each chapter does not correspond exactly to one of the papers I co-authored. Instead, all of the following papers each contributed to a several chapters.

- [BP15]
- [BLP16]
- [BPU16]
- [PUB16]
- [PU16]
- [BKP17]
- [PU17]
- [CDP17]

The content of this part is based on joint works with, in alphabetic order: Alex Biryukov, Anne Canteaut, Sébastien Duval, Dmitry Khovratovich, Gaëtan Leurent and Aleksei Udovenko.

**Main Results of Part II.** *It is possible to decompose all S-Boxes with a structure previously used in the literature. The corresponding techniques can be successfully applied to the S-Box of Kuznyechik and to the apn permutation of Dillon et al.. Besides, the design process of the S-Box of Skipjack can be partially recovered.*

#### 1.3.2.1 Mathematical Tools for Analyzing S-Boxes

**Chapter 8.** In order to study S-Box, it is necessary to first define what they are and what properties they have. This chapter presents their differential, linear and algebraic properties. In particular, the two tables which play a crucial role in the next chapters are defined: the Difference Distribution Table (ddt) and the Linear Approximations Table (lat).

Then, I performed a thorough analysis of the literature where I listed all S-Boxes I could find. These were sorted according to their structures. While some are built like small block ciphers, others rely on mathematical constructions or even simpler heuristic generation algorithms. The proportions of those methods for algorithms using 8-bit S-Boxes are summarized in Figure 1.8.

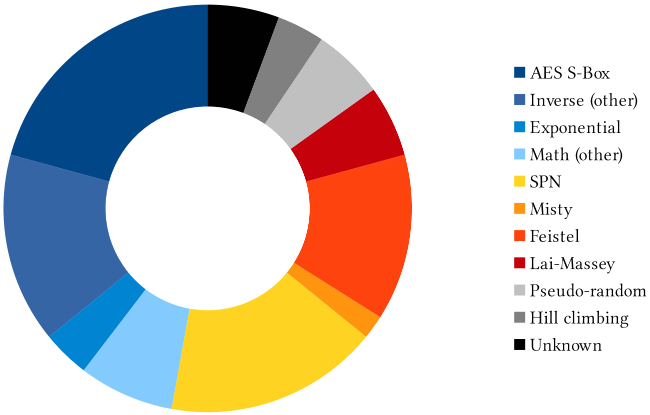


Figure 1.8: Types of structure used to build 8-bit S-Boxes by 53 different algorithms from the literature.

**Chapter 9.** Given the look-up table of an S-Box, how can we know if it were generated at random? It turns out that a careful analysis of the coefficients of its DDT and LAT can often answer this question. In particular, it is shown that the S-Box of the block cipher Skipjack, which was designed by the NSA in the late 80's, has non-random linear properties. Though unimpressive, they are better than what would be expected from a random S-Box. In fact, I show that the probability that this S-Box was picked uniformly at random is upper-bounded by $2^{-54}$.

Other specific DDT/LAT artifacts are explored, such as the similarity of the lines in the LAT of an exponential function and a pattern in the position of the coefficient 4 in the DDT of a permutation affine-equivalent to the multiplicative inverse.

Finally, the analysis used to quantify how "non-random" the S-Box of Skipjack is is applied to all the S-Boxes listed in the previous chapter.

#### 1.3.2.2 Structural Attacks

**Chapter 10.** An S-Box may have a Feistel structure. In this case, it is necessary to attack Feistel networks where the Feistel functions are unknown and do not have any specific structure. The chapter starts with a summary of all structural attacks against Feistel networks. Then, it describes the two I found with my co-authors.

The first is the Yoyo game and its improvement using cycles. It works by iterating the encryption of a plaintext, the addition of a difference to the ciphertext, the decryption of the new ciphertext, the addition of a difference to the new plaintext, etc. The cycle structure of such construction can be leveraged to efficiently identify large groups of pairs of plaintexts such that their encryptions all follow the same known differential trail. This pattern is then used to recover the complete look-up table of the first and last Feistel function of 5-round Feistel networks.

The second is a distinguisher which can identify a Feistel network as such. Consider a $2n$-bit Feistel network encrypting blocks using $r$ rounds and Feistel functions of algebraic degree $d$. Roughly speaking, if

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

then some terms cannot be present in its algebraic normal form. This property can be used as a distinguisher but, if a 4-round Feistel network is composed with affine mappings, these absent terms can be used to mount an attack recovering this mappings.

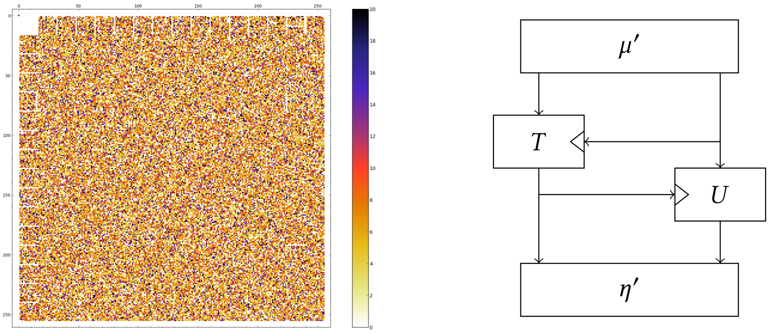
**Chapter 11.** The other large class of block cipher structure is the Substitution-Permutation Network (SPN). Several attacks have already targeted SPN with secret S-Box and affine layers, such as the one against SASAS of Biryukov and Shamir [BS01, BS10] or the ones against ASASA of Dunkelman *et al.* [DDKL15] and Minaud *et al.* [MDFK15].

First, these are briefly recalled. All those attacks rely on the fact that the algebraic degree of the composition of two S-Box layers with an affine transformation is low. By applying a theorem of Boura *et al.* bounding the algebraic degree of a permutation composed with an S-Box layer, we showed that the same attacks could be applied to a higher number of rounds provided that the S-Box size is low enough compared to the block size. In particular, we show that ASASA cannot be secure and that if $n$ is the block size and $m$ the S-Box size, then about $2\log_{m-1}(n)$ rounds are necessary to achieve full degree.

**Chapter 12.** The structural attacks described in Chapters 10 and 11 can only be applied to Feistel networks and SPNs. Any other structure is safe.

To mitigate this problem, we can look at a the *Pollock representation* of the DDT and the LAT, that is, a picture where each pixel corresponds to a table coefficient whose color depends on the absolute value of the coefficient. For example, Figure 1.9a shows the Pollock representation of one of the S-Boxes of the block cipher Clefia.

Several visual patterns that can be expected depending on the properties of the S-Box are described. In particular, the relation between integral and zero-correlation distinguishers mean that those are easy to see — literally.



(a) The LAT of the S-Box $S_0$ of Clefia.

(b) The TU-decomposition.

Figure 1.9: A new type of distinguisher and the decomposition it yields.

The main decomposition method presented in this thesis is the *TU-decomposition*. It is described in Main Theorem 1 which is reproduced below.

**Theorem.** *Let $S : \mathbb{F}_2^n \to \mathbb{F}_2^n$ be an n-bit S-Box with* LAT $\mathcal{L}$ *and let $\eta$ and $\mu$ be linear permutations such that $\mathcal{L}'$ defined by $\mathcal{L}'[a,b] = [\mu(a), \eta(b)] = 0$ verifies $\mathcal{L}'[i,j] = 0$ for all $i < 2^{n/2}$ and $j < 2^{n/2}$. Then $S$ can be written as*

$$S = (\eta^t)^{-1} \circ R_U \circ L_T \circ \mu^t ,$$

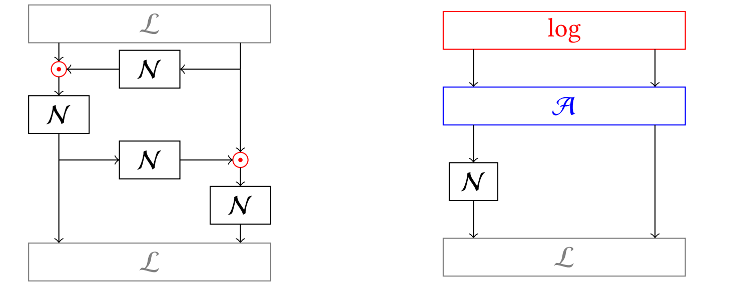
*where $L_T(x||y) = T_y(x)||y$ and $R_U(x||y) = x||U_x(y)$. This corresponds to the structure in Figure 1.9b, where $\mu' = \mu^t$ and $\eta' = (\eta^t)^{-1}$.*

The fact that $\mathcal{L}'[i,j] = 0$ for all $i < 2^{n/2}$ and $j < 2^{n/2}$ corresponds to the presence of a white square in the upper left corner of the Pollock representation of the table, as is the case in Figure 1.9a.

#### 1.3.2.3 Applications of the TU-Decomposition

**Chapter 13.** The latest Russian standards in symmetric cryptography are the 128-bit block cipher Kuznyechik and the hash function Streebog. They use the same 8-bit S-Box $\pi$ but their designers did not disclose their design method. This target was suggested by Oleksandr Kazymyrov who I warmly thank for this pointer.

Using faint patterns in the Pollock representation of its LAT, we can identify a TU-decomposition of this permutation. Its $T$ and $U$ components can be further decomposed into components consisting of multiplying the output of two non-linear function, the multiplication being performed in a finite field of size $2^4$. The overall structure of this decomposition is given in Figure 1.10a, where $\mathcal{L}$ and $\mathcal{N}$ denote respectively some linear and non-linear functions and $\odot$ represents a finite field multiplication.



(a) Feistel-like decomposition.

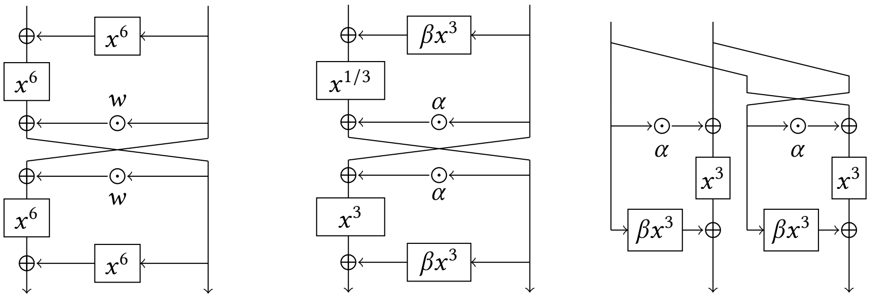
(b) Logarithm-based decomposition.

Figure 1.10: Simplified views of our two decompositions of $\pi$.

The patterns visually identified in the Pollock representation of the LAT of $\pi$ are related to patterns expected in the LAT of exponential permutation such as the one used by the latest Belarussian standard, BelT. As a consequence, we looked for a decomposition of $\pi$ involving such an object. We found several which share the same overall structure described in Figure 1.10b, where $\mathcal{L}$ and $\mathcal{N}$ denote respectively some linear and non-linear functions, log is a discrete logarithm and $\mathcal{A}$ a permutation comprising a couple of arithmetic operations. While these decompositions are not elegant, they highlight an unexpected proximity between $\pi$ and exponential permutations such as the one used in BelT.

**Chapter 14.** A function whose DDT only contains 0 and 2 is called Almost Perfect Non-linear (APN). Whether APN permutations of $\mathbb{F}_{2^n}$ exist for $n$ even is still an open problem, except for $n = 6$ where a unique one was found by Dillon *et al.*, denoted $S_0$.

This S-Box can be decomposed using a TU-decomposition. The resulting $T$ and $U$ components can be expressed as simple combinations of the multiplicative inverse of $\mathbb{F}_{2^3}$, namely $x \mapsto x^6$, and multiplication by a generator $w$ of the multiplicative group of the finite field such that $\text{Tr}(w) = 0$. This decomposition is presented in Figure 1.11a and in Main Theorem 2 which is reproduced below.



(a) A permutation affine equivalent to $S_0$.

(b) Open (bijective) generalized butterfly $\mathsf{H}_{\alpha,\beta}$.

(c) Closed (non-bijective) generalized butterfly $\mathsf{V}_{\alpha,\beta}$.

Figure 1.11: Butterfly structures; where ⊙ denotes finite field multiplications.

**Theorem.** *There exist linear bijections $A$ and $B$ such that the APN 6-bit permutation of Dillon et al. is equal to*

$$x \mapsto= B(S_I(A(x) \oplus 9) \oplus 4,$$

*where $S_I(\ell||r)$ is the concatenation of two bivariate polynomials of $\mathbb{F}_{2^3}$ denoted $S^L_I(\ell,r)$ and $S^R_I(\ell,r)$ and which are equal to*

$$\begin{cases} S^R_I(\ell||r) = (r^6+\ell)^6 + 2 \odot r, \\ S^L_I(\ell||r) = \left(r + 2 \odot S^R_I(\ell,r)\right)^6 + S^R_I(\ell,r)^6 \ . \end{cases}$$

*This structure is summarized in Figure 1.11a.*

This structure can be generalized by considering other elements instead of $w$, by multiplying the output of the Feistel functions by a constant and by increasing the block size. The resulting object are called *generalized butterflies*. The permutation of Dillon *et al.* is a particular case of an *open* generalized butterfly, as described in Figure 1.11b. There are also *closed* generalized butterflies which are quadratic non-bijective functions shown in Figure 1.11c. An open and a closed generalized butterfly are *CCZ-equivalent*, a form of equivalence which preserves the differential and linear properties. By studying the simpler quadratic case of the closed generalized butterflies, we were able to show that open generalized butterflies have the best known differential and non-linear properties over fields of size $2^{4k+2}$. Unfortunately, we also showed that, for $k > 1$, no such APN permutation exists.

### 1.3.3 Part III - On Purposefully Hard Cryptography

This last part deals with cryptographic primitives which are, by design, inefficient. Although Alex Biryukov and I worked on two papers on this topic, the algorithm presented in the second is in the process of being patented at the time of writing. Thus, this part contains only one chapter based on the first of those which is currently under submission.

**Chapter 15.** In many different contexts, it may be desirable to evaluate a primitive with a high complexity. For example, the brute-force of low-entropy passwords can be slowed down by feeding such low-entropy passwords through a time consuming key stretching function.

The efficiency of a primitive is measured along three axes: time, memory consumption and code size. In fact, each of these three quantities are arbitrarily increased in some contexts: a slow primitive is good for key stretching, a memory consuming primitive is good to prevent attackers from speeding up their computation with dedicated ASICs and one with a large code size cannot be duplicated easily, which is precisely the goal of modern white-box block ciphers.

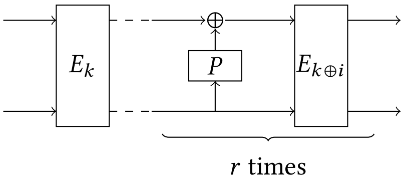


Figure 1.12: The HBC block cipher mode introduced in Section 15.3.2 (p. 316).

We proposed a unifying model which allows the design of a primitive with any of these forms of hardness. It works by combining a secure cryptographic primitive with a *plug*, a small function with the desired form of hardness. Essentially, the cryptographic security reduces to that of the primitive used and the hardness is reduced to that of the plug. For example, it is possible to build a secure time-hard block cipher by using the HBC mode summarized in Figure 1.12 with AES-128 as $E_k$ and a time consuming plug $P$. This method is generalized to sponges, which in turn allows the construction of any symmetric algorithm with any form of hardness.

We also noticed that the hardness may be asymmetric. Let $f$ be a hard function. It is *asymmetrically hard* if it exists a secret key $K$ such that $f_K$ is functionally equivalent to $f$ but does not have its hardness. If an asymmetrically hard plug is used in our constructions, we can then devise a symmetric primitive with any form of asymmetric hardness.

We provide plugs for (asymmetric) time-hardness and (asymmetric) code-hardness. However, only assymmetric memory-hardness is not known to be possible at the time of writing. We illustrate our framework by building an asymmetrically time-hard block cipher called SKIPPER and a code-hard hash function called WHALE.

## 1.4 Publications

The majority of the work done throughout my PhD studies has been published at a conference or in a journal. These papers are listed below.

### Journal Papers

1. Alex Biryukov, Dmitry Khovratovich, and Léo Perrin. Multiset-algebraic cryptanalysis of reduced Kuznyechik, Khazad, and secret SPNs. *IACR Transactions on Symmetric Cryptology*, 2016(2):226–247, 2017

2. Léo Perrin and Aleksei Udovenko. Exponential S-boxes: a link between the S-boxes of BelT and Kuznyechik/Streebog. *IACR Transactions on Symmetric Cryptology*, 2016(2):99–124, 2017

3. Anne Canteaut, Sébastien Duval, and Léo Perrin. A generalisation of Dillon's APN permutation with the best known differential and nonlinear properties for all fields of size $2^{4k+2}$. *IEEE Transactions on Information Theory*, (to appear), 2017

4. Dumitru-Daniel Dinu, Yann Le Corre, Dmitry Khovratovich, Léo Perrin, Johann Großschädl, and Alex Biryukov. Triathlon of lightweight block ciphers for the internet of things. *Journal of Cryptology Engineering*, page To appear, 2017

### Conference Papers

5. Léo Perrin and Dmitry Khovratovich. Collision spectrum, entropy loss, T-sponges, and cryptanalysis of GLUON-64. In Carlos Cid and Christian Rechberger, editors, *Fast Software Encryption – FSE 2014*, volume 8540 of *Lecture Notes in Computer Science*, pages 82–103. Springer, Heidelberg, March 2015

6. Alex Biryukov, Patrick Derbez, and Léo Perrin. Differential analysis and meet-in-the-middle attack against round-reduced TWINE. In Gregor Leander, editor, *Fast Software Encryption – FSE 2015*, volume 9054 of *Lecture Notes in Computer Science*, pages 3–27. Springer, Heidelberg, March 2015

7. Patrick Derbez and Léo Perrin. Meet-in-the-middle attacks and structural analysis of round-reduced PRINCE. In Gregor Leander, editor, *Fast Software Encryption – FSE 2015*, volume 9054 of *Lecture Notes in Computer Science*, pages 190–216. Springer, Heidelberg, March 2015

   This paper was invited to the *Journal of Cryptology* (reviews pending).

8. Alex Biryukov, Gaëtan Leurent, and Léo Perrin. Cryptanalysis of Feistel networks with secret round functions. In Orr Dunkelman and Liam Keliher, editors, *Selected Areas in Cryptography – SAC 2015*, volume 9566 of *Lecture Notes in Computer Science*, pages 102–121, Cham, 2016. Springer International Publishing

9. Alex Biryukov and Léo Perrin. On reverse-engineering S-boxes with hidden design criteria or structure. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *Advances in Cryptology – CRYPTO 2015, Part I*, volume 9215 of *Lecture Notes in Computer Science*, pages 116–140. Springer, Heidelberg, August 2015

10. Léo Perrin and Aleksei Udovenko. Algebraic insights into the secret feistel network. In Thomas Peyrin, editor, *Fast Software Encryption – FSE 2016*, volume 9783 of *Lecture Notes in Computer Science*, pages 378–398. Springer, Heidelberg, March 2016

11. Alex Biryukov, Léo Perrin, and Aleksei Udovenko. Reverse-engineering the S-box of streebog, kuznyechik and STRIBOBr1. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology – EUROCRYPT 2016, Part I*, volume 9665 of *Lecture Notes in Computer Science*, pages 372–402. Springer, Heidelberg, May 2016

12. Léo Perrin, Aleksei Udovenko, and Alex Biryukov. Cryptanalysis of a theorem: Decomposing the only known solution to the big APN problem. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology – CRYPTO 2016, Part II*, volume 9815 of *Lecture Notes in Computer Science*, pages 93–122. Springer, Heidelberg, August 2016

13. Daniel Dinu, Léo Perrin, Aleksei Udovenko, Vesselin Velichkov, Johann Großschädl, and Alex Biryukov. Design strategies for ARX with provable bounds: Sparx and LAX. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *Advances in Cryptology – ASIACRYPT 2016, Part I*, volume 10031 of *Lecture Notes in Computer Science*, pages 484–513. Springer, Heidelberg, December 2016

### Paper Under Submission

14. "Symmetrically and Asymmetrically Hard Cryptography", Alex Biryukov and Léo Perrin. Already available on `eprint` [BP17].

**Seminars and Invited Talks** I also gave several invited talks to present my results. These are listed below.

1. "On reverse-engineering S-boxes with hidden design criteria or structure", Early Symmetric Crypto (ESC), Clervaux, Luxembourg, January 2015.
2. "S-Box Reverse-Engineering: Recovering Design Criteria, Hidden Structures and New Boolean Function Result", Dagstuhl Seminar 16021 (Symmetric Cryptography), Dagstuhl, Germany, January 2016.
3. "Cryptanalysis of a Theorem: Decomposing the Only Known Solution to the Big APN Problem", INRIA, Paris, France, June 2016.
4. "Rétro-ingénierie de boîtes-S", *Séminaire Codage, Cryptologie, Algorithmes (CCA)*, Paris, France, July 2016.
5. "On the Scope of Lightweight Cryptography", Early Symmetric Crypto (ESC), Clervaux, Luxembourg, January 2017.
6. "On S-Box Reverse-Engineering", *CryptoAction Symposium II*, Amsterdam, Netherlands, March 2017.

Part I

# On Symmetric Lightweight Cryptography

# Contents of this Part

Chapter 2

# A Survey of Lightweight Symmetric Cryptography

Lightweight cryptography has been one of the "hot topics" in symmetric cryptography for the past few years. This chapter starts with an overview of this topic in Section 2.1 (p. 29). What is the aim of lightweight cryptography? What makes an algorithm lightweight or not? Section 2.2 (p. 34) presents a list of the lightweight algorithms published as of the time of writing which is as complete as possible. Some trends in the design of lightweight algorithms are identified in Section 2.3 (p. 43). Then, I present a survey of the algorithms which have actually been standardized in Section 2.4 (p. 47). Finally, in Section 2.5 (p. 50), I argue that lighweight cryptography has been spread too thin by including widely different requirements and that it should be divided into two better defined areas: *ultra-lightweight* and *IoT* cryptography.

## 2.1 Overview of Lightweight Cryptography

First, I describe the overall purpose of lightweight cryptography in Section 2.1.1 (p. 29). The design constraints it implies are described in Section 2.1.2 (p. 30). Finally, I discuss the need for dedicated algorithms in Section 2.1.3 (p. 32).

### 2.1.1 General Aim

The Internet of Things (IoT) is one of the foremost buzzwords in computer science and information technology at the time of writing. It is a very broad term describing the fact that, in the near future, the internet will be used more and more to connect devices to one another other rather than to connect people together.

Some of these devices use powerful processors and can be expected to use the same cryptographic algorithms as standard desktop PCs. However, many of them use extremely low power micro-controllers which can only afford to devote a small fraction of their computing power to security.

A common example of such use is that of sensor networks. Such networks are intended to connect vast amounts of very simple sensors to a central hub. These sensors would run on batteries and/or generate their own energy using for example solar panels. Cryptographic algorithms must be used on the messages sent by the sensors to their hub in order to secure them and, in particular, to ensure their authenticity and integrity. However, because of the very low energy available, the cryptographic algorithms have to be as “small” as possible.

Similarly, RFID (Radio-Frequency IDentification) chips are used to identify devices, animals — and even people. In order to prevent an eavesdropper from learning the identification associated to a chip, this information has to be encrypted. Even better, such RFID tags can be used in challenge response protocols. Because of the very small number of logical gates that can be used in such devices and because of the very little energy available, specially designed algorithms are necessary.

### 2.1.2 Design Criteria

The metrics usually optimized are the memory consumption, the implementation size and the speed or throughput of the primitive. However, the specifics of the comparison depend on whether hardware or software implementations are considered.

**Hardware Case.** If the primitive is implemented in hardware, the memory consumption and the implementation size are lumped together into its gate area which is termed in Gate Equivalents (GE). It quantifies how physically large a circuit implementing the primitive is. The throughput is measured in bytes per second and corresponds to the amount of plaintext processed per time unit.

The exact measures depend on the exact type of circuit considered, e.g. the frequency at which it is clocked or the area of each gate. Furthermore, the tools used to simulate those circuits do not give the same results and are usually both proprietary and expensive. Therefore, a fair comparison of the different algorithms proposed regarding their hardware implementation is very difficult. In fact, when comparing their new algorithm with existing ones, designers are usually forced to design their own implementations of preexisting ones too.

Memory is usually the most expensive part of the implementation of a lightweight primitive. In most cases, implementations work by storing the full internal state and key state and then perform one round in $c$ clock cycle, e.g. one round per clock cycle.[1] As a consequence, it is preferable to operate on small blocks using a small key. For some applications, 80-bit keys are preferred over 128-bit ones!

However, some space can be saved by hard-coding or “burning” the keys into the device. That is, instead of using read/write memory to store the key, use read-only structures. In order for this method to be viable, the key schedule must build the round keys using only simple operations taking as input the bits of the master key. In particular, no key state can be operated upon. This strategy has been used by several algorithms, both block and stream ciphers, as shown in Section 2.3.2.2 (p. 45).

**Software Case.** Primitives be implemented in software instead, typically for use on micro-controllers. In this case, the relevant metrics are the RAM consumption, the code size and the throughput of the primitive measured in bytes per CPU cycle. The FELICS framework allows a relevant comparison of these quantities across algorithms and across different implementations of a given algorithm. This framework was presented in [DBG+15] which was later accepted in the Journal of Cryptographic

[1] It is a bit more complicated in the case of a serial implementation. They only update a small part of the state at each round — typically the size of an S-Box — but, due to the simplicity of the logic involved, they allow a far higher clock frequency.

Engineering [DLCK+17]. The name Felics stands for “Fair Evaluation of LIghtweight Cryptographic Systems”.

Felics takes as input the implementation of a block or stream cipher and outputs the corresponding code size, RAM consumption and time taken to perform a given task. These quantities are obtained for three different micro-controllers: an 8-bit AVR, a 16-bit MSP and a 32-bit ARM. The tasks investigated correspond to different scenarios such as the encryption of a 128-bit block in counter-mode. The information extracted is then summarized into a single quantity called Figure of Merit (FoM) (see [DLCK+17] for the exact definition). The lower this FoM, the better. It can be used to rank block ciphers, as shown in Table 2.1 where the block and key sizes are in bits, the code size and maximum RAM consumption are in bytes and the time is in number of CPU cycles.

| General info | | | AVR (8-bit) | | | MSP (16-bit) | | | ARM (32-bit) | | | FoM |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Name | block | key | Code | RAM | Time | Code | RAM | Time | Code | RAM | Time | |
| Chaskey | 128 | 128 | 770 | 84 | 1597 | 490 | 86 | 1351 | 178 | 80 | 614 | 4.7 |
| Speck | 64 | 96 | 448 | 53 | 2829 | 328 | 48 | 1959 | 256 | 56 | 1003 | 4.8 |
| Speck | 64 | 128 | 452 | 53 | 2917 | 332 | 48 | 2013 | 276 | 60 | 972 | 4.9 |
| Chaskey-LTS | 128 | 128 | 770 | 84 | 2413 | 492 | 86 | 2064 | 178 | 80 | 790 | 5.4 |
| Simon | 64 | 96 | 600 | 57 | 4269 | 460 | 56 | 2905 | 416 | 64 | 1335 | 6.6 |
| Simon | 64 | 128 | 608 | 57 | 4445 | 468 | 56 | 3015 | 388 | 64 | 1453 | 6.8 |
| Lea | 128 | 128 | 906 | 80 | 4023 | 722 | 78 | 2814 | 520 | 112 | 1171 | 7.6 |
| Rectangle | 64 | 128 | 602 | 56 | 4381 | 480 | 54 | 2651 | 452 | 76 | 2432 | 8.1 |
| Rectangle | 64 | 80 | 606 | 56 | 4433 | 480 | 54 | 2651 | 452 | 76 | 2432 | 8.1 |
| Sparx | 64 | 128 | 662 | 51 | 4397 | 580 | 52 | 2261 | 654 | 72 | 2338 | 8.3 |
| Sparx | 128 | 128 | 1184 | 74 | 5478 | 1036 | 72 | 3057 | 1468 | 104 | 2935 | 12.4 |
| RC5-20 | 64 | 128 | 1068 | 63 | 8812 | 532 | 60 | 15925 | 372 | 64 | 1919 | 13.5 |
| Aes | 128 | 128 | 1246 | 81 | 3408 | 1170 | 80 | 4497 | 1348 | 124 | 4044 | 14.1 |
| Hight | 64 | 128 | 636 | 56 | 6231 | 636 | 52 | 7117 | 670 | 100 | 5532 | 14.8 |
| Fantomas | 128 | 128 | 1712 | 76 | 9689 | 1920 | 78 | 3602 | 2184 | 184 | 4550 | 18.8 |
| Robin | 128 | 128 | 2530 | 108 | 7813 | 1942 | 80 | 4913 | 2188 | 184 | 6250 | 22.0 |

Table 2.1: The current best Felics results for scenario 2: counter mode encryption of 128 bits.

The three quantities measured are not independent. For example, loading information from the RAM into CPU registers is a costly operation and so is its inverse. Therefore, limiting the number of such operations leads to a decrease in both RAM consumption and time complexity.

**Side-Channel Attack Resilience.** Side-channel attacks (SCAs) use some special knowledge about the implementation of a cipher to break its security. For example, observing the power consumption of an encryption can leak information about the Hamming weight of the output of an S-Box.

Such attacks demand that the cryptanalyst has physical access to the device attacked. However, this requirement is particularly easy to fulfill in the context of the IoT: a desktop computer can be expected to be reasonably hard to physically interact with because it is in a locked room but a sensor measuring traffic in an open street may not enjoy such protection.

As a consequence, lightweight algorithms are often built in such a way as to decrease the vulnerability of their implementation to such attacks. This can be done through the use of inherently less leaky operations or by simplifying the use of a masked implementation. These topics are further discussed in Section 2.3.1 (p. 43).

**Other Implementation Criteria.** More recently, other criteria have emerged for the design of lightweight algorithms. For example, energy and power efficiency are at the core of the design of the Midori block cipher published about a year ago [BBI+15]. Another criteria is latency, that is, the time taken to perform a given operation. There are contexts in which a low-latency is crucial, for example for memory encryption. This particular requirement requires specific design choices as illustrated by the lightweight block ciphers Prince [BCG+12] and Mantis [BJK+16].

**Common Trade-Offs.** To accommodate these constraints, most lightweight algorithms are designed to use smaller internal states and smaller key sizes. Indeed, while a 128-bit block and at least 128-bit key was demanded from the Aes candidates, most lightweight block ciphers use only 64-bit blocks. This smaller size leads to a smaller memory footprint in both software and hardware. It also means that the algorithm is better suited for processing smaller messages.

Nevertheless, the small block size can be a problem as the security of some modes of operation such as cbc erodes very quickly when the number of $n$-bit blocks encrypted approaches $2^{n/2}$, as exploited for example in [BL16]. As a consequence, dedicated modes of operation such as [LPTY16] have been proposed to mitigate these issues. Furthermore, key sizes are often as small as 80 bits which offers little security margin against brute-force search. Such an attack is likely to be infeasible nowadays for all but the most powerful state sponsored adversaries, but how long will this last? As pointed out in [Mou15], time-memory-data tradeoff can become an issue if the key size is too small, especially in the multi-key setting.

In the case of lightweight block ciphers, it is also common for the components used to be involutions so as to decrease the cost of the implementation of decryption. This can be done by using non-linear involutions as S-Boxes or by using a Feistel structure. On the other hand, this issue can be mitigated through the use of modes of operations that do not require block cipher decryption. For example, if a block cipher is used in counter mode, the area/rom which would be needed to store the description of the inverse block cipher can be saved.

Finally, due to the importance of their performance, lightweight algorithms often have thinner security margins. For example, Ketje [BDP+16] is a sponge-based authenticated cipher where the sponge transformation uses only one round. That is, it does not even provide full diffusion. This obvious issue is offset by the use of a non-repeating nonce.

### 2.1.3 Are Dedicated Algorithms Needed?

As lightweightness is mostly a property of the implementation of an algorithm, we can wonder if dedicated algorithms are actually needed. Would it not be sufficient to use lightweight *implementations* of regular algorithms?

It is often possible. For example, many implementers have worked on optimizing the implementation of the Aes with some success in both hardware [BJM+14, BBR16, UMHA16] and software [SS16]. Even if an efficient implementation of the Aes is impossible using the instructions available on a micro-controller, those devices are sometimes shipped with a hardware acceleration module for this task, effectively adding a new set of instructions dedicated entirely to a quick evaluation of this block cipher.

In this context, lightweight symmetric algorithms may seem unnecessary. However, block cipher hardware acceleration has its limitations. As summarized for example in Table 1 of [OC16] (which is reproduced in Table 2.2), the hardware-accelerated encryptions used by many devices are vulnerable to various forms of side-channel attacks. These attacks do not only target these devices "in a vacuum". For example, Philips light bulbs using the Zigbee protocol to communicate have been recently shown to be insecure [ROSW16]. One of the key components of this attack is a subversion of the update mechanism of the light bulb. Updates are normally authenticated with an AES-based MAC using a secret key which is constant across all devices. Ronen *et al.* recovered this key via an SCA and were therefore able to push malicious updates to these devices.

| Product | Cipher |
|---|---|
| DESFire, MF3ICD40 | 3-DES |
| DS2432, DS28E01 | SHA-1 |
| Microchip HCSXXX | Keeloq |
| ProASIC3 | AES |
| Spartan-6 | AES |
| Stratix II | AES |
| Stratix III | AES |
| Virtex-II | 3-DES |
| Virtext-4, Virtex-5 | AES |
| XMEGA | AES |
| Yubikey | AES |

Table 2.2: Several micro-processors whose hardware accelerated cryptography is vulnerable to SCA (reproduced from [OC16]).

Still, there are cases where side-channel attacks are not really relevant. For example, a yubikey[2] is supposed to be always carried by its owner, so that studying its power consumption is not practical for the adversary. However, if attackers can easily access devices with the secret key they are after, e.g. in the case of a wireless sensor network, such a weakness is not acceptable. This problem could be mitigated by using protected implementations such as masked ones.

On micro-controllers without hardware support for cryptographic functions, their assembly implementation must be as small and as fast as possible. In these cases, the AES is decently fast, especially on 8-bit micro-controllers where it is in fact one of the fastest. However, its implementation requires storing at least the full look-up-table of its 8-bit S-Box, meaning that its code size cannot be as small as that of dedicated algorithms.

All in all, while the AES is a decent lightweight block cipher, its large S-Box, large block size and inherent vulnerability to SCA caused by its look-up-based S-Box make it a suboptimal choice in many cases.

[2] A yubikey is a commercial USB drive designed to store cryptographic keys securely and use them in authentication protocols.

Another case where dedicated lightweight algorithms are needed is for hashing. Indeed, standard hash functions need large amounts of memory to store both their internal states — 1600 bits in the case of SHA-3 — and the block they are operating on — 512 bits in the case of the SHA-2 family. These memory requirements significantly hinder performance on lightweight platforms and justify the need for dedicated lightweight hash functions.

## 2.2 Symmetric Lightweight Algorithms

Several very distinct actors are involved in the field of lightweight cryptography. In fact, they are the same that discuss “regular” cryptography: academia, industry, standardizing bodies and government agencies — including spying agencies.

The industry is supposed to be the implementer of those algorithms, designing or choosing the best one for their purpose. Unfortunately, until the 2000’s and the spread of the AES, many of the algorithms used had been designed in-house with little regard to what is considered best practice. The corresponding algorithms are described in Section 2.2.1 (p. 34).

Academics have published dozens of symmetric cryptographic algorithms claiming to be lightweight. Those are listed in Section 2.2.5 (p. 39). Said publications always contain a description of the cryptanalysis attempts by the authors of the algorithm.

This creates a significant contrast with the algorithms proposed by government agencies: even their public algorithms have been designed in a secret way. Furthermore, these agencies are more often than not also in charge of spying, (see the American NSA and the Russian FSB), meaning that they have contradictory incentives. On the one hand, it is their task to ensure the security of their own citizens which may imply designing strong encryption. On the other hand, as evidenced by the first crypto wars and the current ongoing debate surrounding the alleged fear of some law enforcement agencies of “going dark”, they may also seek to purposefully weaken these standards. What might have been discounted as mere conspiracy theory a few years ago is now an established fact: the Snowden documents show that the NSA has a budget dedicated to the subversion of cryptographic standards and has pushed for the standardization of the Dual EC pseudo-random number generator [BLN15]. My co-authors and I have also shown, as explained in Chapter 13 (p. 245), that the latest FSB designs share an S-Box with a hidden structure. Lightweight algorithms designed by such government agencies are listed in Section 2.2.6 (p. 42).

### 2.2.1 Algorithms from the Industry

Many lightweight algorithms used by industrial products are surprisingly weak. Many of those algorithms were designed in the 80’s or early 90’s, a time during which cipher design had to accommodate for the stringent American export laws which forbid selling devices with overly strong cryptography. Still, like modern lightweight algorithms, those were intended to run on devices with little computing power devoted to encryption.

The algorithms in this section were, at least at first, intended to be kept secret. They were published through leaks or reverse-engineering (except for the Kindle cipher). I list them below. They are stream ciphers unless their name is followed by a “†” mark, in which case they are block ciphers, or a “‡” mark corresponding to MACS.

- A5/1
- A5/2
- A5-GMR-1
- A5-GMR-2
- Cmea †
- Crypto-1
- CryptoMem.
- Cryptomeria †
- Csa-BC †
- Csa-SC
- Css
- Dsc
- Dst40 †
- E0
- Hitag2
- iClass
- Keeloq †
- Megamos
- Oryx
- PC-1
- RC4
- SecurID ‡
- SecureMem.

As already noted by Kerckhoffs in 1883 [Ker83] and as recalled in Section 1.1.2 (p. 3), security through obscurity is no security. It should be assumed that the attacker has access to the algorithm and the security should be based on the secrecy of the key. This principle was not followed by many of the algorithms below and, unsurprisingly, those collapsed as soon as their specification was leaked or reverse-engineered.

### 2.2.2 Industry-Designed Stream Ciphers

**A5/1.** The exact design date of this algorithm is unclear but a first approximation of its inner workings was published in 1994 [And94]. It generates a keystream from a 22-bit IV along with a 64-bit key using three different lfsrs whose lengths add up to 64 bits. Practical attacks have been implemented using time-memory trade-offs exploiting the fact that the update function of the internal state is not bijective [Gol97, BSW01]. The most time efficient of those needs only $2^{24}$ simple steps provided that a significant (but practical) pre-computation was performed. Furthermore, 10 bits of the key were always set to 0 in many implementations. The 2G gsm protocol still uses this algorithm.

**A5/2.** A cipher somewhat similar similar to A5/1 but even weaker was intended to be used in countries targeted by American export restrictions. It is called A5/2. It is vulnerable to ciphertexts only attacks with complexity $2^{16}$ using redundancy introduced by error correcting codes. It requires a one-time pre-computation of practical complexity. Unfortunately, interoperability imposed the implementation of this algorithm on devices supposed to run A5/1 instead, thus making downgrade attacks possible [BBK03, BBK08].

**A5-GMR-1 and A5-GMR-2.** Satellite phones have their own protocols and, therefore, use their own cryptographic algorithms. The two algorithms used, A5-GMR-1 and A5-GMR-2, were reverse-engineered by Driessen *et al.* in [DHW$^+$12]. Those are very different from one another but both are easily attacked.

- A5-GMR-1 is a variant of A5/2 with an internal state consisting in 4 lfsrs with a total size of 82 bits. Those are clocked irregularly, much like in A5/2. It can be attacked using only known ciphertexts by inverting $2^{21}$ triangular matrices of size 532×532, which requires roughly $2^{21} \times 532^2/2 \approx 2^{38.1}$ simple operations. A significant but practical pre-computation step is necessary.

- A5-GMR-2 is a byte oriented stream cipher with a much more sophisticated structure based on 3 different components denoted $\mathcal{F}$, $\mathcal{G}$ and $\mathcal{H}$ by Driessen *et al.*. Surprisingly, $\mathcal{H}$ uses the S-Box $S_2$ and $S_6$ of the Des. A practical attack with very low data and time complexity is presented in [LLLS14]. It requires guessing at most 32 bits using only 1 frame of 15 bytes for an average complexity of $2^{28}$.

**Atmel Ciphers.** The stream ciphers used by the SecureMemory, CryptoMemory and CryptoRF families of products from Atmel are similar to one another. They are proprietary algorithms which were reverse-engineered and attacked by Garcia *et al.* in [GvRVWS10]. Other more powerful attacks were later proposed by Biryukov *et al.* [BKZ11] breaking the cipher of SecureMemory in time $2^{29.8}$ using 1 frame and the cipher of CryptoMemory in time $2^{50}$ using 30 frames and about 530 Mb of memory. The ciphers rely on 3 NLFSRS with a total size of a bit more than 100 bits. The attacks found by both Garcia *et al.* and Biryukov *et al.* were successfully implemented.

**Crypto-1.** It is a stream cipher used by the *Mifare classic* line of smartcards of NXP. It was reverse-engineered by Nohl *et al.* in [NESP08] and was subsequently attacked by many teams [CNO08, Gol13] with a time complexity as low as $2^{32}$. It has been used at least since 1998 but the exact date of its design is unclear. It is based on a 48-bit LFSR combined with several non-linear Boolean functions.

**Content Scrambling System (Css).** In order to implement Digital Rights Managements (DRMS), the content of DVD discs is encrypted. This encryption used to be performed with a stream cipher called Css. It uses two 17- and 25-bit long LFSRS to generate two 8-bit words in parallel. These are afterwards added modulo $2^8$ to obtain a byte of keystream. However, unlike in most stream ciphers, this key stream is not simply XORed with the plaintext. Instead, the plaintext first goes through an 8-bit bijective S-Box whose result is added to the keystream to obtain the ciphertext. This operation is sometimes called the *mangling step*. A full description is available in [BD04] and in [PMA07]. Several powerful attacks target the protocol using this stream cipher. However, given its key length of 40 bits, the cipher alone is vulnerable to a brute-force search of time complexity $2^{40}$.

**Common Scrambling Algorithm (Csa-SC).** The Common Scrambling Algorithm is used to secure digital television broadcast. It cascades two ciphers, as described in [WW05]. The first is a block cipher which we call Csa-BC and which is described below. The second is a stream cipher which we call Csa-SC. The stream cipher is based on two FSRS consisting of twenty 4-bit cells each and a combiner with memory. The feedback function of the registers involves, among other things, several $5 \times 2$ S-Boxes. The combiner uses addition modulo $2^4$ to extract 2 bits of keystream from its internal state and the two shift registers at each clock cycle. In [WW05], several undesirable properties are presented. For example, the keystream often has very short cycles. It is also possible to recover the secret key by solving about $2^{28}$ systems of 60 linear equations with 40 unknowns which must take at most $2^28 \times 60^3 \approx 2^{45.7}$.

**Dsc.** The Dect[3] Standard Cipher, usually abbreviated into Dsc, is a stream cipher used to encrypt the communications of cordless phones. First, attacks targeting the protocol using it and its flawed implementation were presented in [LST+09]. It was subsequently reverse-engineered and its attackers found practical attacks requiring

[3] Dect stands for “Digital Enhanced Cordless Telecommunications”.

only about $2^{15}$ samples of keystream and $2^{34}$ trial encryptions which take a couple of hours on a standard computer to recover the key [NTW10]. It is described by the authors of this paper as being "an asynchronous stream cipher with low gate complexity that takes a 64-bit secret key and a 35-bit initialization vector." Its structure, based on irregularly clocked LFSRS, is reminiscent of that of A5/1.

**E0.** The privacy of the Bluetooth protocol is now based on the AES but it used to rely on a custom stream cipher called E0. Its 128-bit internal state is divided into 4 LFSRS and its filter function has its own 2-bit memory. A description of E0 can be found in the papers presenting attacks against it such as [FL01, LV04, LMV05]. Lu *et al.* found an attack which recovers the secret key using the first 24 bits of $2^{23.8}$ frames and with $2^{38}$ computations.

**Hitag2 ; Megamos.** These stream ciphers are used in the *car immobilizers* implemented by different car manufacturers. These devices prevent a car engine from starting unless a specific transponder is close to them. While initially kept secret, the first was published by Wiener[4] and the second was reverse-engineered by Verdult *et al.* [VGE13]. They are both stream ciphers with a small internal state of 48 and 57 bits respectively. These small sizes and other weaknesses in the ciphers themselves and in the protocols using them lead to practical attacks against the devices relying on these algorithms for security. For example, it is possible to attack a car key using Hitag2 using 1 min of communication between the key and the car and about $2^{35}$ encryptions. The secret key of Megamos can be recovered in time $2^{48}$ but more powerful attacks are possible using the key update mecanism of the devices using it.

**iClass.** Formally, iClass is family of smartcards introduced in 2002. The stream cipher it uses was reverse-engineered and attacked by Garcia *et al.* in [GdKGV14]. It has a 40-bit internal state. The cryptanalysts who reverse-engineered it presented attacks against this cipher in the same paper. By recording $2^{22}$ authentication attempts, the key can be recovered using $2^{40}$ trial encryptions.

**Kindle Cipher (PC1).** This stream cipher was first published on Usenet by Alexander Pukall in 1997, meaning that this algorithm was not technically designed in the industry. However, it was not designed by academics and Amazon used it at least up until 2012 for the DRM scheme "protecting" its e-book using the MOBI file format. It uses a 128-bit key and a separate 24-bit internal state updated using different operations, including modular multiplications. The keystream is generated byte by byte. It has been broken by Biryukov *et al.* [BLR13] using e.g. $2^{20}$ known plaintexts and a time of $2^{31}$. Even practical known-ciphertexts attacks are possible in some contexts.

**Oryx.** While A5/1 "secures" GSM communications in Europe, the stream cipher Oryx was chosen by the Telecom Industry Association Standard (TIA) to secure phone communications in north America. A description of the algorithm can be found in [WSD+99] where practical attacks are presented. It uses a 96-bit key, a 96-bit internal state consisting of three 32-bit LFSRS, and an 8-bit S-Box which changes every time. It is possible to attack it in time $2^{16}$ using 25 bytes of known plaintext.

[4] While this first publication is mentioned for example by Verdult *et al.* in [VGB12], I was not able to find a copy of it. Nevertheless, the specification of Hitag2 can be found in [VGB12].

**RC4.** First designed by Ron Rivest in 1987, this stream cipher was intended to remain a trade secret of the Rsa company. However, it was leaked to the cypherpunk mailing list in 1994 [Nob94] and turned out to be a remarkably simple algorithm. Unfortunately it has several issues, in particular when its first outputs are not discarded. The attack from [ABP+13] was successfully implemented: using between $2^{28}$ and $2^{32}$ encryptions of the same message, it is possible to recover it using biases in the keystream.

The now deprecated WEP protocol for wireless communications used it for encryption. It lead to practical attacks implemented e.g. by the `aircrack`[5] tool allowing an attacker to recover the password protecting WiFi access. It uses a 256-byte internal state containing all numbers in $\{0, ..., 255\}$ which is updated using a very simple rule each time an 8-bit output is generated. It supports all key sizes between 40 and 2048 bits, although it usually uses 128-bit keys.

### 2.2.3 Industry-Designed Block Ciphers

**Cmea** This block cipher was used by the TIA to secure the transmission of phone numbers across telephone lines. A good description of this algorithm is provided in [WSK97] which, incidentally, describes an attack against the full cipher. It encrypts a block of an arbitrary number of bytes — although in practice those were usually 2 to 6 bytes long — using a 64-bit key. It is vulnerable to a known plaintext attacks requiring only 40–80 blocks of data and taking a time between $2^{24}$ and $2^{32}$ encryptions.

The extreme weakness of this cipher might indicate that it was designed with little care, and yet its designers included a hidden structure in its S-Box. It is exhibited in Section 12.3.2.

**Cryptomeria.** It is a block cipher nicknamed "C2" in the literature. It shares the same structure as the Des: it encrypts 64-bit blocks using a 56-bit key and uses a 32-bit Feistel function. It works by mixing in a 32-bit subkey with a modular addition, then use one 8-bit S-Box call followed by a 32-bit linear permutation. The S-Box is secret, so an S-Box recovery attack has been proposed [BKLM09]. The same paper presents a key recovery with time complexity $2^{48}$. This algorithm was intended from the start to be used by "things", namely DVD players (in which case it can be seen as a successor of Css) and some SD cards. In total, 10 rounds are used; which means that only 10 S-Box calls are needed to encrypt one 64-bit block compared to, say, the 160 calls needed to encrypt one 128-bit block using Aes-128.

**Common Scrambling Algorithm (Csa-BC).** The Common Scrambling Algorithm uses a stream cipher (described above) and a block cipher which we call Csa-BC. It encrypts a 64-bit block using a 64-bit key. Its structure is reminiscent of a generalized Feistel network using eight 8-bit branches. The Feistel functions are based on a unique random-looking 8-bit S-Box $B$ and a variant defined as $\sigma \circ B$, where $\sigma$ is a simple bit permutation. An encryption consists in 56 rounds. A full specification is given in [WW05]. To the best of our knowledge, there is no attack other than brute-force against this cipher.

[5] Its successor, `aircrack-ng`, is described on its official website `http://www.aircrack-ng.org/`.

**Dst40.** This algorithm was reverse-engineered from partial information disclosed in a patent and from a physical device implementing this block cipher [BGS+05]. It was used by RFID transponders sold by Texas Instrument. They were used in car immobilizers and for electronic payment. The cipher itself encrypts a 40-bit block with a 40-bit key using 200 rounds of an unbalanced Feistel network. The Feistel function maps 38 bits of internal state and a 40-bit subkey to a 2-bit output by nesting several Boolean functions. Due to its key size of 40 bits, a brute-force search is practical.

**Keeloq.** It is a so-called "code-hopping encoder". A US patent was filed in 1993 and eventually granted in 1996 [BSK96] but it was designed earlier, around 1985 [Lea14]. Using modern terminology, it is a 32-bit block cipher which uses a 64-bit key. It was first kept secret but its specification was leaked in 2006. Using this information, several teams presented practical attacks against devices using this algorithm [IKD+08]. For example, the key be recovered using $2^{16}$ known plaintexts and $2^{44.5}$ encryptions. Far more powerful side-channel attacks have also been proposed against commercial implementations of the cipher [EKM+08]. This ciphers was still in use when these attacks were found, 20 years after its design.

### 2.2.4 Industry-Designed Macs

**SecurID mac.** A SecurID is small hardware token used for authentication and designed by SDTI (which was later bought by Rsa Security). It displays a 6 digit number which changes every minute. It is based on a 64-bit MAC described for example in [BLP04]. This paper also presents attacks against the algorithm. The details of the MAC were initially kept secret but were eventually leaked which lead to the attacks of Biryukov *et al.*. These were later sped up by Contini *et al.* [CY04] to obtain a time complexity of about $2^{44}$ MAC computations.

### 2.2.5 A Semi-Exhaustive List of Public Algorithms

Throughout the last 25 years and especially since 2011, a lot of algorithms intended to be lightweight have been published in cryptography- and security-related conference proceedings and journals. Those are listed in this section.

The algorithms in these lists have either been advertised as lightweight in their specification, have a very small implementation or have been standardized as such. Figure 2.1 provides an overview of all lightweight symmetric algorithms published by academics, sorted by publication date and by type.

#### 2.2.5.1 Stream ciphers

The eSTREAM competition was held in 2008 to choose two portfolios of stream ciphers. The first type of algorithms fits the so-called *software profile*, meaning that they were aimed at software efficiency. The second category was the *hardware profile*. Further, some of them use an internal state so small that they can be considered to be lightweight stream ciphers.

However, lightweight stream ciphers were proposed outside the framework of this competition. For example, Snow 3G corresponds to a simple modification of the academic-designed Snow 2.0 tailored for specific industrial needs: it is used in the 3GPP communication standard.

All such lightweight stream ciphers I am aware of are listed below.

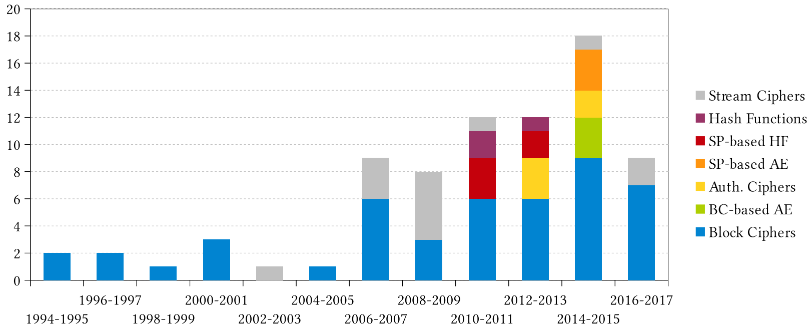


Figure 2.1: Lightweight algorithms published by academics. If an authenticated cipher is based on a block cipher, it is listed as a BC-based AE. Similarly, algorithms based on sponges are shown as such.

- A2U2 [DRL11]
- Chacha20 [Ber08a]
- Enocoro [WIK+08]
- F-FCSR-H/16 [ABL+09]
- Grain [HJM07]
- Lizard [HKM17]
- Mickey [BD08]
- Plantlet [MAM17]
- Salsa20 [Ber08b]
- Snow 2.0 [EJ03]
- Snow 3G [ETS06a]
- Sprout [AM15]
- Trivium [Can06]

#### 2.2.5.2 Block ciphers

Block ciphers are the most common choice for designers trying to build a lightweight symmetric algorithm. All those designed and published by academics I am aware of are listed below, sorted by date of publication. Block ciphers followed by a dagger "†" were published as part of a higher level construction such as an authenticated encryption scheme submitted to the Caesar competition[6] or a MAC. Similarly, tweakable block ciphers are marked with a double dagger "‡".

- 3-Way [DGV94]
- RC5 [Riv95]
- Misty1 [Mat97]
- Xtea [NW97]
- Aes [DR98]
- Bksq [DR00]
- Khazad [BR00b]
- Noekeon [DPVAR00]
- Iceberg [SPR+04]
- Hight [HSH+06]
- mCrypton [LK06]
- Sea [SPGQ06]
- Clefia [SSA+07]
- Deslx [LPPS07]
- Present [BKL+07]
- Mibs [ISSK09]
- Katan/Ktantan [CDK09]
- Gost revisited[7] [PLW10]
- PrintCipher [KLPR10]
- Epcbc [YKPH11]
- Klein [GNL11]
- LBlock [WZ11]
- Led [GPPR11]
- Piccolo [SIH+11]
- Picaro [PRC12]
- Prince [BCG+12]
- ITUbee [KDH13]

[6] The project Caesar (Competition for Authenticated Encryption: Security, Applicability, and Robustness) aims at identifying the best authenticated ciphers. All submissions are listed on the following web page: http://competitions.cr.yp.to/caesar-submissions.html.

[7] The "Gost cipher" has been a public Russian standard since 1994 when it was published in the standard GOST 28147-89. It is thus referred to as "GOST" in most of the literature, although the more recent block cipher Kuznyechik described in Section 13.1.1 (p. 246) is also such a GOST standard. In GOST 28147-89, the S-Boxes are not specified. The version of this algorithm described in [PLW10] is technically covered by this standard, but a specific set of S-Boxes is provided. It should not be mistaken for Magma [Fed15], another modern update of the old Gost cipher aiming at lightweightness. Magma has been designed by the Russian fsb and is described in Sections 2.2.6 (p. 42).

- Twine [SMMK13]
- Zorro [GGNS13]
- Chaskey[8] [MMH+14] †
- Pride [ADK+14]
- Joltik [JNP14a] † ‡
- Lea [HLK+14]
- iScream [GLS+14] † ‡
- LBlock-s [ZWW+14] †
- Scream [GLS+14] † ‡
- Lilliput [BFMT15]
- Rectangle [ZBL+15]
- Fantomas [GLSV15]
- Robin [GLSV15][9]
- Midori [BBI+15]
- Simeck [YZS+15]
- RoadRunneR [BŞ16]
- Fly [KG16]
- Mantis [BJK+16] ‡
- Skinny [BJK+16] ‡
- Sparx [DPU+16a]
- Mysterion [JSV17]
- Qarma [Ava17] ‡

#### 2.2.5.3 Hash functions

It is more difficult to implement a lightweight hash function than a lightweight block cipher. Indeed, they usually require a much larger internal state which is reasonable on a desktop computer but would have a prohibitive cost on a lightweight device. For example, Sha-3 uses a 1600-bit internal state which dwarfs the 64-bit block of most lightweight block ciphers.

And yet, since a collision in the internal state leads to a collision in the final digest, it has to have at least a size corresponding to the desired security level.

As an answer to this problem, several designers chose the use of a sponge construction with a very small rate. Indeed, the internal state of a sponge is divided into two distinct parts:

- the $r$-bit rate decides how fast the plaintext is processed and how fast the final digest is produced, and
- the $c$-bit capacity determines the security level as for example a birthday collision search succeeds with a time complexity of roughly $2^{c/2}$ independently of the digest size.

For example, using a capacity of 128 bits along with a rate of 8 bits, as done in some versions of the hash functions listed below, minimizes the memory footprint at the cost of a slower data processing.

All lightweight hash functions I am aware of are listed below. Sponge-based ones are marked with a dedicated symbol: "※".

- Armadillo [BDN+10]
- Blake2s/b [ANWOW13]
- Gluon [BDM+12] ※
- Lesamnta-LW [HIK+11]
- Photon [GPP11] ※
- Quark [AHMN10] ※
- SipHash [AB12]
- Spongent [BKL+11] ※
- Spn-Hash [CYK+12] ※

#### 2.2.5.4 Dedicated Authenticated Encryption Schemes

Following the call for submissions of the Caesar competition, several lightweight authenticated encryption schemes were proposed. Some of them rely on dedicated block ciphers used with specific modes, in which case their underlying block cipher is in the list above. However, other algorithms based on sponge transformation or stream cipher-like construction were also proposed. These are listed below. As with hash functions, sponge-based algorithms are marked with the symbol "※".

[8] While Chaskey is a sponge-based mac, its authors proposed the used of the *Chaskey Cipher*, an Even-Mansour construction using the sponge transformation of Chaskey as a public permutation.

[9] To prevent the invariant subspace attack targeting this cipher, a variant called Robin* with different round constants was proposed in [JSV17].

- Acorn [Wu16]
- Ale [BMR$^{+}$14]
- Asc-1 [JK12]
- Ascon [DEMS16] ※
- Fides [BBK$^{+}$13]
- Hummingbird-2 [ESSS12]
- Ketje [BDP$^{+}$16] ※
- Lac [ZWW$^{+}$14]
- Norx32 [AJN16] ※
- Helix [FWS$^{+}$03]
- Sablier [ZSX$^{+}$14]

### 2.2.6 Algorithms from Government Agencies

Governmental agencies have published their own lightweight ciphers. The publication is often done via national standards. These ciphers are usually targeting local usage but, due to the interconnection of the markets and the corresponding standardizing efforts, these can end up being used outside of their expected zone of influence.

For example, Simon and Speck are two block ciphers designed by the American National Security Agency (NSA) which were disclosed on `eprint.iacr.org` [BSS$^{+}$13]. However, they might in the end be used outside of the USA as the NSA is lobbying ISO/IEC to include them as part of the standard for "lightweight cryptography", as mentioned in Section 2.4.1 (p. 48). The designers of these algorithms were invited to present these algorithms to the Design Automation Conference (DAC) of 2015 but their paper [BTCS$^{+}$15] provides little insight into their process. In particular, it discloses no information regarding their security analysis.

The Skipjack block cipher is in a similar position. The rationale behind its design is not known. The only public information comes from the report written by external cryptographers after two days spent at the NSA headquarters [BDK$^{+}$93]. Their comments and what my co-authors and I were able to reverse-engineer from the specification are discussed in Section 9.2 (p. 168).

While some of the design criteria used to build the S-Boxes of the Des [U.S99] have eventually been published [Cop94], the exact generation process remains a mystery.

Finally, the same can be said of the Russian lightweight block cipher Magma. It is specified as a part of the latest Russian standard for block ciphers, Gost R 34.12–2015 [Fed15], which both describes this algorithm and the heavier Kuznyechik described and thoroughly investigated in Chapter 13 (p. 245). Again, the rationale behind these designs is not known. Their specification merely discloses the algorithms, not their design process and especially not the best cryptanalysis of their authors.

The stream cipher Zuc was designed by the Data Assurance and Communication Security Research Center (DACAS) of the Chinese Academy of Science. It was published directly as part of the 3GPP standard [ETS11]. This addition was caused by the demand from the Chinese government to use Chinese algorithms when operating in China. Zuc should in theory not be used while in other countries. Unlike in the cases of the Russian and American algorithms, some information is provided regarding its design. In particular, several modifications were made necessary by external cryptanalysis which are described in [ETS11]. Nevertheless, the cryptanalysis (hopefully) performed by its original designers is, to the best of my knowledge, still secret.

The public lightweight ciphers designed by government agencies are listed below.

- Des [U.S99]
- Magma [Fed15]
- Simon [BSS$^{+}$13]
- Skipjack [U.S98]
- Speck [BSS$^{+}$13]
- Zuc [ETS11] †

## 2.3 Trends in Lightweight Design

Lightweightness can be seen as a set of specific design constraints. These are tackled differently by different algorithms but some trends emerge when we look at the evolution of lightweight block ciphers. These are particularly visible on two fronts: the choice of the non-linear operations and the key schedule which are described respectively in Section 2.3.1 (p. 43) and in 2.3.2 (p. 44). In Section 2.3.3 (p. 46), the fact that fewer bad ciphers seem to be in use now than 15 years ago is discussed.

### 2.3.1 Non-Linear Operations

Non-linearity is a necessary property of any cryptographic primitive. It can be provided by S-Boxes, a type of function studied thoroughly in Part II (p. 133), or through the use of non-linear arithmetic operations. S-Box-based algorithms can further be divided into two categories. The first implements them using Look-Up Tables (LUT) and the second relies on bit-sliced implementations. As for arithmetic operations, only modular additions are considered here, that is, primitives following the ARX paradigm. Although other operations are sometimes used, such as modular multiplication in the block cipher IDEA [LM91] and the PC1 stream cipher, these are extremely uncommon.

#### 2.3.1.1 LUT-based

*LUT-based* algorithms use S-Boxes which are intended to be implemented using look-up tables in software. Such functions are useful as they can offer (near) optimal cryptographic properties using only one operation. However, their implementation requires storing all possible outputs which, for an 8-bit S-Box such as the one used by the AES, has a significant cost. Furthermore, the table look-up is the operation leaking the most information, as shown in [BDG16].

S-Boxes intended to be implemented using LUTs in software usually correspond to a simple electronic circuit which can be efficiently implemented in hardware, such as the 4-bit S-Boxes used by Piccolo, PRESENT and SKINNY.

#### 2.3.1.2 Bit-slice-based

*Bit-slice-based* algorithms also use S-Boxes but, in this case, the S-Box is supposed to be implemented in a bit-sliced fashion: no table look-ups are required to evaluate the S-Box layer. Instead, some bitwise operations such as AND and XOR are performed on words of $w$ bits, thus evaluating the S-Box in parallel $w$ times.

S-Boxes implemented in this fashion are typically designed for this purpose. Thus, they require only a limited number of logical operations: 4-bit ones usually need only 4 ANDs during their evaluation which makes their software implementation particularly easy to mask. At the same time, they allow simple security argument based for example on the wide trail strategy. A simple bit-sliced implementation is also related — but is not equivalent to — a small area for a hardware implementation, meaning that such algorithms can be expected to perform well in hardware as well.

Because of these properties, bit-sliced S-Boxes are a popular choice for the design of lightweight algorithms, especially during the last 4 years. For example, all the algorithms in the following list use such components.

- 3-Way
- Ascon
- Fantomas
- Fly
- iScream
- Ketje
- Mysterion
- Noekeon
- Pride
- Rectangle
- RoadRunneR
- Scream

#### 2.3.1.3 Arx-based

*Arx-based* algorithms rely on modular addition to provide non-linearity while word-wise rotations and xor provide diffusion, hence the name: Addition, Rotation, xor.

The bits of highest weight in the output of a modular addition are highly non-linear functions due to the propagation of the carry. However, the lower weight bits retain a simple dependency. Furthermore, some differentials and linear approximations have probability 1, meaning that the structure of the linear part must be studied carefully. In fact, as far as I know, our proposal Sparx presented in Chapter 7 (p. 117) is the only arx-based primitive designed to be provably secure against differential and linear attacks.

Modular addition is fairly expensive to implement in hardware, especially if the size of the words is larger as this significantly increases the length of the critical path. On the other hand, it is extremely cheap in software. Not only does it consist in one operation, it also uses fewer or no additional registers as it can often be performed in place using the “+=” operator.

As a consequence, arx-based ciphers are among the best performers for micro-controllers identified using Felics. Some arx-based block and stream ciphers are listed below.

- Chacha20
- Chaskey
- Hight
- Lea
- RC5
- Salsa20
- Sparx
- Speck
- Xtea

### 2.3.2 Key Schedule

The key schedule is the area where lightweight algorithms differ the most from their non-lightweight counterparts. Indeed, for algorithms intended to run on standard computers, it is fine to have a complex key schedule as it would typically be run only once, the corresponding subkeys being subsequently stored. For lightweight algorithms, the incurred cost in terms of ram or gate area is unacceptable. Furthermore, it is common for lightweight algorithms to dismiss resilience against related key attacks, a design decision which authorizes the use of much simpler key schedules.

Different attitudes regarding related-key attacks are discussed in Section 2.3.2.1 (p. 45). Popular strategies for building simple key schedules are described in Sections 2.3.2.2 (p. 45) and 2.3.2.3 (p. 46).

Some recent proposals provide a tweak in addition to the secret key [LRW02]. It is a public parameter which enables the use of more sophisticated modes of operation. In fact, most of these algorithms are parts of authenticated ciphers submitted to the Caesar competition. However, an overwhelming majority of lightweight block ciphers do not provide this functionality.

#### 2.3.2.1 On Related-Key Attacks

Some cipher designers claim resilience against related-key attacks while some other algorithms are trivially vulnerable against such attacks. For example, the block cipher Prince, whose round reduced versions are attacked in Chapter 4 (p. 73), has a very simple related-key distinguisher because of its $\alpha$-reflection. On the other hand, other algorithms explicitly give resilience against related-key as a design criteria.

Preventing related-key attacks is a more conservative choice. Whatever the setting, from a security standpoint, being protected against such adversaries can only be an advantage. And yet this resilience has a cost since it implies the use of more rounds and/or more complex key schedules which lead to a performance degradation particularly unwelcome in the lightweight setting. Furthermore, for devices using a unique factory-defined key throughout their lifetimes, the probability of finding two devices with the appropriate key relation is small enough that it is of no practical concern. Similarly, if the protocols using the ciphers are properly implemented, related-key attacks should not be possible.

The following algorithms were explicitly *not* designed to prevent related-key attacks. Still, the approach used for Noekeon and Fly is a bit more subtle. Indeed, for cases where related-key attacks might be of concern, the authors provide a modified key schedule. While normally the master key is simply XORed in the state, as for the ciphers in Section 2.3.2.2 (p. 45), the related-key protected version imposes that the master key first go through several rounds of the round function so as to break any pattern relating the keys.

- Fantomas
- Fly
- Mysterion
- Noekeon
- Pride
- Prince
- Zorro
- ...

Some designers prefer to make the most conservative choice by providing related-key security. Some of the corresponding algorithms are listed below. These usually employ a more complex key-schedule but, since they remain lightweight ciphers, those can be evaluated “on the fly” cheaply. It means that the subkeys are obtained by extracting bits from a key state which is updated in every round, just like the internal state of the block cipher. However, this update function is kept simple to limit the performance overhead.

- Epcbc
- LBlock
- Sea
- Simon
- Skinny
- Sparx
- Twine
- ...

#### 2.3.2.2 Even-Mansour and “Selecting” Key Schedules

It is popular for lightweight algorithms to use a key schedule which merely selects different bits of the master key in each round for use as subkey material along with some round constants. If the master key of a block cipher is simply XORed to the internal state during each round along with a round constant, the key schedule can be seen as a variant of the Even-Mansour construction [EM97]. Of course, it is possible to use said construction directly, as is the case for the Chaskey cipher. It is also possible to use different chunks of the master key during encryption. For example, as detailed in Section 9.2.1 (p. 168), Skipjack uses a 32-bit subset of its 80-bit master key in each round. The subkeys therefore repeat themselves every 5 rounds. I call such a key schedule a *selecting* key schedule since it merely selects some bits of the master key for use as subkeys.

The main advantage of such methods is that they require very little logic to compute the round keys. Furthermore, they have no need for a key state getting updated at each round which would be particularly expensive in hardware. This observation is what lead the designers of the stream cipher Sprout, followed later by those of Lizard and Plantlet, to fix the content of one of their registers to be the master key without modifying it. In fact, the paper introducing Plantlet [MAM17] provides a detailed analysis of the way a key stored in non-volatile memory can be accessed and its impact on both performance and algorithm design. For example, it is better to access master key bits sequentially, like in Skipjack and in Led, than to use master key bits that are far apart to build a given round key.

Below, I list all ciphers using the master key in such a way that no key state needs to be maintained. It encompasses the (iterated) Even-Mansour construction, the "selecting" key schedules and the stream ciphers that do not modify their key register. Stream ciphers are indicated by the "†" symbol.

- 3-Way
- Chaskey
- Des
- Fantomas
- Fly
- Gost revisited
- Hight
- iScream
- ITUbee
- Ktantan
- Led
- Lizard †
- Magma
- Midori
- Mysterion
- Noekeon
- Piccolo
- Plantlet
- Pride
- Prince
- RoadRunneR
- Robin
- Scream
- Skipjack
- Sprout †
- Xtea
- Zorro

The impact of such a key schedule in terms of gate area in hardware is extensively discussed in the recent paper [MAM17] which introduced Plantlet.

#### 2.3.2.3 Round Function Based

A simple strategy to have a substantial key schedule while minimizing its cost is to reuse significant parts of the round function to update the key state. The whole round function can be used, as in Speck, or only parts of it, as in Sparx. Several block ciphers using this principle are listed below. For Fly and Noekeon, only the key schedule protecting against related-key attacks is concerned.

- Epcbc
- Fly
- Noekeon
- Sea
- Simeck
- Sparx
- Speck

### 2.3.3 No More Non-Standard Ciphers?

So far, I have discussed trends regarding algorithm design. But there is another trend at a higher level: questionable ciphers such as those listed in Section 2.2.1 (p. 34) are being phased out. Nowadays, the prevalence of the Aes means that using algorithms such as A5/1 would be unacceptable. Not only new standards are concerned: previously existing standards such as Bluetooth have been amended to move away from their previous *ad hoc* solutions (here, the E0 stream cipher) and towards more common choices (for Bluetooth, the Aes). The reason behind this change is probably two-fold.

First, the lessons from the attacks targeting proprietary algorithms have likely been learned. Thus, once these standards had to be replaced by more modern ones, the cryptography used was updated at the same time.

Second, the Aes likely played a significant role. The fact that it performs decently — even if not optimally — on a wide variety of platforms, means that it a priori constitutes a satisfactory choice in most situations. As explained in Section 2.1.3 (p. 32), the situation is unfortunately a bit more complicated but using this algorithm is nevertheless a significant step up from what was done before. As a consequence of this versatility, the Aes has been formally standardized for use in most areas, as explained in Section 2.4 (p. 47). Still, the time it took for the algorithms from Section 2.2.1 (p. 34) to be phased out shows the importance of getting an algorithm choice right from the start.

## 2.4 Lightweight Cryptography in the Wild

Several standards and libraries are aimed at use cases overlapping with those of lightweight cryptography. These are listed in this section and summarized in Table 2.3. Section 2.4.1 (p. 48) deals with iso/iec standards, Section 2.4.2 (p. 49) with regional cryptographic ones, Section 2.4.3 (p. 49) with general purpose communication protocols run by low power devices and, finally, Section 2.4.4 (p. 49) describes several libraries specifically aimed at the IoT.

| Type | Name | Lightweight algorithms standardized |
|---|---|---|
| iso/iec | 29167 | Aes-128, Present-80, Grain-128A |
| | 29192-2 | Present, Clefia |
| | 29192-3 | Enocoro, Trivium |
| | 29192-5 | Photon, Lesamnta-LW, Spongent |
| | 18033-3 | Aes, Misty1, Hight |
| | 18033-4 | Snow 2.0 |
| Regional | FIPS 185 (usa) | Skipjack (now deprecated [BR15]) |
| | FIPS 197 (usa) | Aes |
| | Nessie (EU) | Aes, Misty1 |
| | eStream portfolio (EU) | Grain, Trivium, Salsa20, Mickey |
| | gost R 34.12–2015 (Russia) | Magma |
| Protocols | GSM | A5/1, A5/2, A5/3 (Kasumi) |
| | 3G | Snow 3G, Zuc, Aes, Kasumi |
| | Bluetooth | E0, Aes |
| | Bluetooth smart | Aes |
| | WEP | RC4 |
| | WPA | RC4 |
| | WPA2 | Aes |
| | Lora Alliance | Aes |
| | IEEE 802.15.4 (Zigbee) | Aes |
| Embedded Lib. | Tinysec | Skipjack (CBC), (RC5) |
| | Minisec | Skipjack (OCB) |
| | mbedTLS (ciphers) | Aes, RC4, Xtea, Blowfish, 3-Des, Camellia |
| | mbedTLS (hash functions) | MD5, Sha-1, Sha-256, Sha-512 |

Table 2.3: Standards and libraries involving lightweight algorithms.

### 2.4.1 Iso/iec cryptographic standards.

The International Organization for Standards (ISO) and the International Electrotechnical Commission (IEC) are tasked with issuing and maintaining standards regarding information and communication technology.

Three of their standards are particularly relevant for lightweight cryptography. The first is *ISO/IEC 29167: Information technology – Automatic identification and data capture techniques*, in particular parts 10, 11 and 13. Those deal with the symmetric ciphers that should be used for securing “air interface communications”, that is, rfid tags. These parts describe respectively Aes-128, Present-80 and Grain-128A. Other parts deal with public key cryptography.

Another set of relevant iso/iec standards are those with number 29192 which deal specifically with “lightweight cryptography”. The following algorithms are part of this series of standards: the block ciphers Present and Clefia, the stream ciphers Trivium and Enocoro, and the hash functions Photon, Spongent and Lesamnta-LW. The criteria for algorithms to be considered for inclusion in this standard are listed in the following quote from Annex A of said standard.

> a) The security of the cryptographic mechanism. 80-bit security is considered to be the minimum security strength for lightweight cryptography. It is however recommended that at least 112-bit security be applied for systems that will require security for longer periods (refer to SD12 for security strength references, as the period of protection provided is determined by the security strength as well as the computing power of the adversary who wishes to break the algorithm.).
>
> b) The hardware implementation properties (for hardware targeted mechanisms). The chip area occupied by the cryptographic mechanism (reduced compared to existing ISO standards) and the energy consumption. (clear advantage over existing ISO standards, e.g. ISO/IEC 18033, ISO/IEC 9798, ISO/IEC 11770).
>
> c) The software implementation properties (for software targeted mechanisms). In particular, the code size and the required RAM size. (Less resource requirements compared to existing standards on the same platform are considered as potentially lightweight for software environments).
>
> d) The nature of any licensing issues affecting the cryptographic mechanism.
>
> e) The maturity of the cryptographic mechanism.
>
> f) The generality of the lightweight properties claimed for the cryptographic mechanism (i.e. the more independent the claimed lightweight property is from implementation in a specific technology, the better, as it will be usable by a wider audience).

At the time of writing, the block ciphers Simon and Speck designed by the nsa were being considered for inclusion in this standard.

Finally, standard 18033 describes “Encryption algorithms”. Some of those can be considered lightweight, such as the block ciphers Aes, Misty1 and Hight (in 18033-3) and the stream cipher Snow 2.0 (in 18033-4).

### 2.4.2 Regional Cryptographic Standards

Several regional standards deal with cryptography in general and some of the algorithm specified in them can be considered to be lightweight. In the USA, cryptographic standards are handled by the National Institute for Standards and Technology (NIST) which famously standardized the AES after an open competition. This institution is currently working towards a standard for lightweight cryptography, as explained in a detailed report [MBTM16]. Their intention is to agree upon several *profiles* corresponding to different algorithms, use cases and constraints. Then, possibly different algorithms will be standardized for use in each of these profiles. The legacy cipher Skipjack was a NIST standard but it is now deprecated.

In Europe, the NESSIE project selected several block ciphers including the AES and MISTY1. Its failure to find good stream ciphers lead to the eSTREAM competition. At its end, a portfolio of stream ciphers was published. It is divided into two profiles, one software oriented and one hardware oriented. Several of those stream ciphers can be considered to be lightweight: Trivium, Grain, MICKEY and Salsa20.

Finally, the latest Russian standard for block ciphers contains the 64-bit block cipher Magma.

### 2.4.3 Communication protocols

Several communication protocols specify a form of encryption which, given the nature of the devices running them, have to be lightweight. For example, cell phones are not nearly as powerful as computers, although Moore's law and modern smartphones complicate this picture.

The GSM and 3G networks deal with cell phone communication. They specify that communications should be encrypted using A5/1, A5/2, A5/3 (KASUMI in counter mode), SNOW 3G, ZUC or KASUMI, the latter being a variant of MISTY1.

Bluetooth connects devices over short distances. The original specification required the stream cipher E0 but it was later replaced by the AES. A more recent variant called "Bluetooth smart" aims at lower energy consumption. It also relies on the AES for its security.

Modern WiFi connections are secured using WPA or WPA2. The former uses RC4 while the latter moved on to the AES. The previous standard was WEP, which used RC4, but practical attacks exist against it.

Several protocols have recently been proposed to connect wireless IoT devices to one another. The one put forward by the Lora Alliance uses the AES. The same is true for IEEE 802.15.4, which is used e.g. in Zigbee.

### 2.4.4 IoT Oriented Libraries.

Let us look at two libraries intended for embedded devices. The first we consider is `tinysec`[10] which is used in the security-related stack of the TinyOS operating system. It uses Skipjack in CBC mode, although RC5 was also considered and found to be quite efficient [KSW04]. Its successor is `minisec`[11] and it also uses Skipjack but in OCB mode. The library `mbedTLS` which also targets embedded devices but is not tied to TinyOS offers several algorithms, namely the ciphers AES, RC4, XTEA,

[10] It is described in the wiki of the TinyOS operating system: http://tinyos.stanford.edu/tinyos-wiki/index.php/TinySec.

[11] See http://tinyos.stanford.edu/tinyos-wiki/index.php/MiniSec.

Blowfish, 3-Des and Camellia as well as the hash functions MD5, Sha-1, Sha-256 and Sha-512.

## 2.5 Two Faces for Lightweight Crypto

Providing a formal definition of "lightweightness" is a difficult task because different algorithms corresponding to different sets of requirements claim to fall under its umbrella. In this section, I argue that the area of lightweight should be split into two distinct fields. The first is *ultra-lightweight cryptography*, described in Section 2.5.1 (p. 50). The second is *IoT cryptography* and is discussed in Section 2.5.2 (p. 52).

As evidenced by all the primitives listed in Section 2.2 (p. 34), *a lot* have been proposed: the list in Section 2.2.5.3 (p. 41) contains only block ciphers published by academics and it contains 47 entries! More importantly, even within this a priori narrower subset, algorithms differ greatly. Let us look at two extreme cases:

- Ktantan encrypts blocks consisting of at most 64 bits using an 80-bit key and 254 very simple rounds, while
- Lea uses 32-bit modular additions, xors and rotations to encrypt 128-bit blocks with keys of length 128, 192 or 256.

Both of these algorithms would be considered lightweight, and rightfully so: the circuit needed to evaluate the non-linear function of Ktantan consists only in a few gates while Lea is one of the top performers in the Felics triathlon. And yet, a category so wide that two algorithms so different both fit comfortably in it must be of little use in terms of classification.

The distinction between these block ciphers goes beyond their intended target — although Ktantan is indeed hardware-oriented and Lea software oriented. In fact, their differences highlight another gap: what security level is desirable in the context of lightweight cryptography? There are two broad schools of thought on this matter.

- Lightweight algorithms are intended to run on cheap devices securing cheap objects, for example rfid tags used to track an inventory of T-shirts. What would be the point in paying for a high level of security in this context? Indeed, the consequences of an adversary successfully attacking these tags would be local at worst.
- On the other hand, IoT devices are, by definition, connected to the internet. It implies that they can be used e.g. for Denial-of-Service attacks, as has already happened. In this context, can we afford *not to* have a maximum level of security?

Below, I will argue that both points of view are correct. Rather, the mistake lies in lumping both use cases together.

### 2.5.1 Ultra-Lightweight Crypto

As its name indicates, this type of algorithm deals with the most constrained use cases. On top of the power of the devices, what defines this field is also their connectivity.

For example, RFID tags used for challenge-response based access control may not need full-on 256-bit security against adversaries having access to the full codebook. As the throughput of these devices is very limited, it makes sense to discard attacks requiring too much data. Similarly, should the secret key used by this card be recovered, the consequences would be restricted geographically to whatever place this card was used in. It would be the same as a physical key being lost and revoking the access of this card would be the same as changing door locks. In this context, "weak" cryptography with only 80-bit keys may be understandable. However, a great emphasis on side-channel protection is likely to be necessary in many cases.

**Definition 2.5.1** (Ultra-Lightweight Crypto)**.** *An* ultra-lightweight *cryptographic algorithm is one running on very cheap devices which are not connected to the internet, which are easily replaced if necessary and have a limited shelf-life.*

Possible use cases for such algorithms include RFID tags, Rain [Rob16] tags, smart cards, remote car keys, memory encryption... Besides, many algorithms already fit this bill such as Grain, Ktantan, Photon, Present, Prince, Skinny and Trivium to name a few.

Because the implementation constraints are particularly stringent in this context, some specific trade-offs can be relevant. For example, Prince is unlikely to make it to the top of the Felics triathlon because its design was aimed at low-latency in hardware. Lower block sizes are also acceptable.

Here are some properties an ideal ultra-lightweight algorithm and its implementation may have.

- *Type:* block/stream cipher for versatility and small memory footprint.
- *Block size:* 64 bits (or more if possible)[12];
- *Key size:* at least 80 bits, more if possible;
- *Relevant attacks:* since the devices running ultra-lightweight algorithms have very little computing power, they cannot be expected to produce large amounts of data. Thus, it makes sense to only consider attacks with rather low data complexity.[13]
- *SCA resilience:* countermeasures must be easy to implement by design.
- *Use of non-volatile memory:* using non-volatile memory to store the key is cheaper, even though it may require extra care when designing the algorithm.
- *Functionality:* only one type of operation per device — for example, the block cipher used on a given smart card will only be used in a challenge-response protocol. Thus, there is no need for a hash function or a MAC.
- *Flexibility:* it must be possible to optimize any of the relevant quantities. In other words, if a low latency is needed, the algorithm should allow it at the cost of a possible increase in area or decrease in throughput. Similarly, a low area should be possible, etc. To quote [Rob16] (emphasis his): "*flexibility* gives the opportunity to find the right trade-off".

[12] In [Rob16], Robshaw explains that even Rain RFID tags can afford blocks of 64 bits: "there is no demand for very short block lengths (*e.g.* 48 bits)".

[13] This limitation has already been suggested by the designers of Prince when they issued the "Prince challenge". It is further mentioned in Chapter 4.

The ecosystem of ultra-lightweight algorithm can be expected to be diverse. While a unique algorithm capable of fitting in every niche of the design space would be welcome, it is likely that different algorithms are used in different cases.

### 2.5.2 IoT Crypto

The other subfield of lightweight cryptography is *IoT cryptography.* It is oriented toward the IoT in its most literal sense, that is, it deals with objects connected to the internet. While remaining computationally weak compared to a desktop computer or a higher end smartphone, these devices perform multiple tasks. Accordingly, the primitive they use must be versatile: unlike ultra-lightweight devices which only need one cryptographic operation, IoT ones need to both encrypt and authenticate their communications with their user, authenticate the updates from their manufacturers, etc.

Another key difference is the importance of their security. The consequences are further reaching in the case of IoT ones due to their network connection. For example, the attacks against the“smart” light bulbs presented in [ROSW16] can spread from one light bulb to the next. Furthermore, it could be used effectively to jam the WiFi signal in a vast geographical area because of the overlap between the frequencies involved. Similarly, an insecure IoT-enabled device can be used in a DoS attack to take down websites. As we can see, the security level needed in those cases is much higher: 80-bit keys are not acceptable in this context, at least 128 bits are necessary.

**Definition 2.5.2** (IoT Crypto)**.** *An* IoT *cryptographic algorithm is one running on a low-power device connected to a global network such as the internet.*

Unlike in the ultra-lightweight case, there should ideally be only one algorithm or one suite of algorithms for all IoT devices: as they are all networked, they must all use the same primitives. Since some of these devices will run in conditions in which an attacker may physically access them, such as outdoor security cameras, it is crucial that SCA counter-measures be easy to implement.

Because IoT devices perform multiple tasks, the cryptographic operations will be performed by their multi-purpose micro-controllers rather than an electronic circuit. Thus, software efficiency is much more important in this case. Several lightweight algorithms have been explicitly designed for software rather than hardware implementation: Chaskey, FLY, LEA, PRIDE, SPARX...

In light of all this, let us list some of the properties such an IoT algorithm should have.

- *Type:* block cipher or sponge. In the IoT case, the device must be able to perform many different operations so a versatile primitive is needed. Furthermore, as the intended target is a micro-controller, the idea of “burning” the key into the circuit does not make sense. Instead, the key will have to occupy some registers which may as well be used to build the larger internal state of a sponge.
- *Block size:* 96 bits[14] is the minimum, higher sizes must be preferred. Ideally, the internal state and the key — if any — should all fit into the registers of a typical micro-controller.

[14] A block size of 96 bits is such that the data complexity of an attack based on the birthday paradox is $2^{16}$ times higher than for a 64-bit block. Attacking 64-bit block cipher is practical, as illustrated by [BL16], but multiplying the data complexity of such attacks by $2^{16}$ is sufficient to have some security margin.

- *Key size:* at least 128 bits.
- *Relevant attacks:* the security model must be more conservative than in the ultra-lightweight case. For example, limiting the amount of data available to the adversary would be too restrictive.
- *SCA resilience:* countermeasures must be easy to implement by design.
- *Use of non-volatile memory:* the use of non-volatile memory is less relevant in this context as the main target is software implementation.
- *Functionalities:* encryption, authentication, hashing... Such devices communicate non-trivial data with different actors, which means that they require several cryptographic functionalities.
- *Flexibility:* the algorithm must be decently efficient on a wide range of micro-controllers. Hardware efficiency is less important but it may help with hardware acceleration.

## 2.6 Final Remarks

Lightweight cryptography has received significant attention in the last two decades and even more so in the last 5 years. The need for such algorithms is well established, as evidenced by the NIST effort to standardize such algorithms and the short-comings of the AES in this context.

| | Ultra-Lightweight | IoT |
|---:|---:|---:|
| Block size | 64 bits | ≥ 128 bits |
| Security level | ≥ 80 bits | ≥ 128 bits |
| Relevant attacks | low data/time complexity | Same as “regular” crypto |
| Intended platform | dedicated circuit (ASIC, RFID...) | micro-controllers, low-end CPUs |
| SCA resilience | important | important |
| Functionality | one per device, e.g. authentication | encryption, authentication, hashing... |
| Connection | to a central hub | to a global network |

Table 2.4: A summary of the differences between ultra-lightweight and IoT cryptography.

However, the spectrum of use cases encompassed by “lightweightness” has become *too* wide. Thus, I propose to split the field of lightweight cryptography into two areas corresponding to two different types of requirements: ultra-lightweight and IoT cryptography, whose particularities are summarized in Table 2.4. There are of course common criteria between those cases such as the emphasis that needs to be put on resilience against SCA. Still, I think that the difference between the two is relevant. In particular, I think this separation is necessary because of the different levels of security they demand: connecting a family of devices to a global network and protecting them with an 80-bit key is not a desirable situation, and yet it is what may happen if an ultra-lightweight algorithm is used where an IoT one is needed.

Chapter 3

# Vanishing Differences in Gluon

Consider a function $g : \mathcal{S} \to \mathcal{S}$ where $\mathcal{S}$ is some finite space of size $2^N$ and suppose that it is not a permutation, i.e. that it has collisions. It is well known that for a random $g$ the complexity of a collision search is of $2^{N/2}$ calls to $g$. However, not only the collision search complexity but also some related problems are not well studied when collisions have a certain structure, which is the case in several designs [BD08, Gol97]. It might be clear that iterating such a function may lead to an entropy loss, but again, the scale of this loss and its implications on the security of stream ciphers and hash functions is not well known or is underestimated. In this chapter, we introduce a particular parameter, called the Collision Probability Spectrum (cps), which is based on the number of solutions of the following equation

$$g(a + y) = g(a) \tag{3.1}$$

and investigate its consequences over the iterated images and preimages of $\mathcal{S}$ by $g$. We assume that the composition of two such functions has certain properties, which is formalized as an independence assumption. For a large class of mappings two important facts are proved in Theorem 3.1.2.

- First, the size of the iterated image of $g$ is inversely proportional to the number $i$ of iterations:

  $$|g^i(\mathcal{S})| \sim \frac{|\mathcal{S}|}{\frac{\kappa}{2} \cdot i},$$

  where $\kappa$ depends on the cps and where $i$ has to be smaller than $\sqrt{|\mathcal{S}|}$. Otherwise, the result does not hold because of the cycles in the functional graph.

- Second, an element $y \in g^i(\mathcal{S})$ is the root of a *collision tree* consisting of elements $x_l$ such that any of $g(x_l), g^2(x_l), \ldots, g^i(x_l)$ is equal to $y$. The average size of this tree is $\nu_i$:

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

  with the same restriction $i < \sqrt{|\mathcal{S}|}$.

These phenomena are summarized in Figure 3.1. The dots represent elements of $\mathcal{S}$ and there is an oriented edge from $x$ to $y$ if $g(x) = y$. Here, $g(a + x) = g(a)$ always has exactly 3 solutions.

Using this framework, we construct an attack on Gluon-64. We find collisions for the update function with the help of a sat-solver and demonstrate a preimage

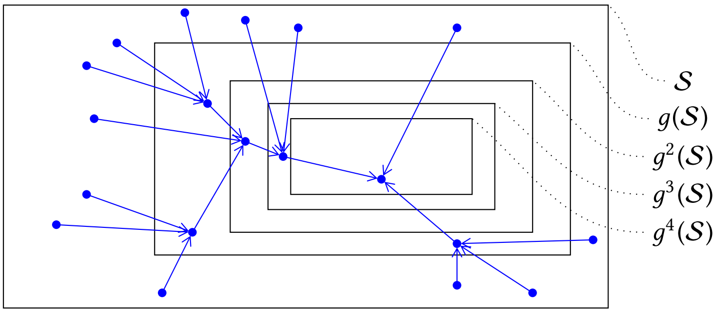


Figure 3.1: Collision trees and output shrinkage of iterative non-injective functions.

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

This chapter is organized as follows. We introduce our theoretical framework in Section 3.1 and discuss its application to existing primitives. We investigate the security of T-sponge against collision and preimage search in Section 3.3. Finally, in Section 3.4, we obtain inner-collisions of the update function of Gluon-64 with the help of a SAT-solver and show a preimage attack.

But first, let us put this work into context by listing some previous works on similar topics. Bellare and Kohno [BK04] studied how the number of preimages to $g(a)$ affects the complexity of the collision search with the notion of *balance* of a function. In [FO90], Flajolet and Odlyzko studied several characteristics of random mappings, in particular the distribution of preimage sizes, the cycle size and the size of the iterated image. Their result was applied by Hong and Kim [HK05] to the Mickey [BD08] cipher. Indeed, they found experimentally that the size of the iterated images of this function was essentially the size of the space divided by the number of iterations, a behavior which they showed experimentally to correspond to the prediction of Flajolet *et al.*. However, the resulting attacks were found to be less efficient than a simple collision search [Röc08], though they allow a time/memory trade-off. The collision trees implied by our findings were observed in the stream cipher Mickey by Helleseth *et al.* in [HJKK13]

This chapter focuses on the macroscopic effects of iterating a function with inner-collisions. A recent work by Daemen [Dae16], published after the paper this chapter is based on [PK15], explores the local effects of these inner-collisions using the Walsh spectra of such lossy functions.

## 3.1 Theoretical Framework

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

### 3.1.1 Collision Probability Spectrum and Function Model

**Definition 3.1.1** (Collision Probability Spectrum). *Let $\mathcal{S}$ be a finite space and let $g : \mathcal{S} \rightarrow \mathcal{S}$ be a function. We denote $\gamma_k$ the probability that the following equation has* *exactly $k$ solutions for $a \in \mathcal{S}$ picked uniformly at random in $\mathcal{S}$:*

$$g(a+x) = g(a), \tag{3.2}$$

*so that*

$$\gamma_k = \Pr\left[\#\{x \in \mathcal{S}, g(a+x) = g(a)\} = k \mid a \xleftarrow{\$} \mathcal{S}\right] \tag{3.3}$$

*The solutions $x$ of this equation are called* vanishing differences. *The set of all the elements $a$ of $\mathcal{S}$ such that Equation (3.2) has exactly $k$ solutions is denoted $V_k$. Finally, the set $\Gamma = \{\gamma_k\}_{k\geq 1}$ is the* Collision Probability Spectrum *(CPS) of $g$.*

An equivalent definition of the CPS is that it is the probability distribution of the number of solutions of Eq. 3.2. We now make some remarks regarding these definitions:

- Since 0 is always a solution of Equation (3.2), we have that $\gamma_0 = 0$.
- If $g$ is a permutation, then $\Gamma(g) = \{\gamma_1 = 1, \gamma_k = 0 \text{ for } k > 1\}$.
- The input space can be partitioned into $\mathcal{S} = \bigcup_{k=1}^{\infty} V_k$ and the output space can be partitioned into $g(\mathcal{S}) = \bigcup_{k=1}^{\infty} g(V_k)$. Both are disjoint unions.
- The size of $g(V_k)$ is $|g(V_k)| = |\mathcal{S}| \cdot \gamma_k / k$ because to each element in $g(V_k)$ correspond $k$ elements in $V_k$ (see Figure 3.2). As a consequence,

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

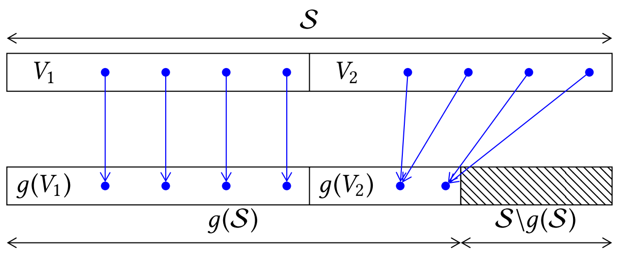


Figure 3.2: The effect of $g$ with CPS $\{\gamma_1 = \gamma_2 = 1/2\}$ on $\mathcal{S}$.

### 3.1.2 Composition of Functions with Known CPS

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

**Assumption 3.1.1** (Independence Assumption)**.** *Let $g$ be a function with CPS $\Gamma$. Then there is no correlation between the events $x \in V_j$ and $g(x) \in V_k$ for any $j,k$.*

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

**Definition 3.1.2.** *Suppose $g$ is a function on $\mathcal{S}$. Then $\ell_i$ defined as*

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

*is called the* shrinking ratio *of $g$.*

Our first theorem shows how to compute the shrinking ratio of the composition of two functions with known and possibly different CPS.

**Theorem 3.1.1.** *Let $g$ and $g'$ be functions with* CPS *$\Gamma = \{\gamma_k\}_{k\geq 1}$ and $\Gamma' = \{\gamma'_k\}_{k\geq 1}$, respectively. Then the shrinking ratio of the composition $g \circ g'$ is computed as follows:*

$$\ell_1(g \circ g') \;=\; \left(\frac{1}{\ell_1} - \sum_{k=1}^{\infty} \frac{\gamma_k}{k}\left(1 - \frac{1}{\ell'_1}\right)^k\right)^{-1}.$$

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

$$\ell_{i+1} = \left(\frac{1}{\ell_1} - \sum_{k=1}^{\infty} \frac{\gamma_k}{k}\left(1 - \frac{1}{\ell_i}\right)^k\right)^{-1}.$$

*Full proof of Theorem 3.1.1.* We shall look at the effect multiple iterations of $g$ have over sets $\{x_0, \dots, x_k\}$ where $g(x_j) = g(x_{j'})$ for all $j, j'$.

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

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

$$C_{m+1} = \sum_{k=m}^{\infty} \gamma_{k+1}\binom{k}{m}\left(\frac{1}{\ell'_1}\right)^m\left(1 - \frac{1}{\ell'_1}\right)^{k-m}. \tag{3.4}$$

Furthermore, we have:

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

and so:

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

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

$$\begin{aligned}
\frac{\ell'_1}{\ell_1(g \circ g')} &= \sum_{k=0}^{\infty} \frac{\gamma_{k+1}}{k+1} \sum_{m=0}^{k} \binom{k+1}{m+1} \left(\frac{1}{\ell'_1}\right)^m \left(1 - \frac{1}{\ell'_1}\right)^{k-m} \\
&= \sum_{k=0}^{\infty} \frac{\gamma_{k+1}}{k+1} \sum_{m=1}^{k+1} \binom{k+1}{m} \left(\frac{1}{\ell'_1}\right)^{m-1} \left(1 - \frac{1}{\ell'_1}\right)^{k-(m-1)} \\
&= \sum_{k=0}^{\infty} \frac{\gamma_{k+1} \cdot \ell'_1}{k+1} \left( \sum_{m=0}^{k+1} \binom{k+1}{m} \left(\frac{1}{\ell'_1}\right)^m \left(1 - \frac{1}{\ell'_1}\right)^{k+1-m} - \left(1 - \frac{1}{\ell'_1}\right)^{k+1} \right).
\end{aligned}$$

Note that $\sum_{m=0}^{k+1} \binom{k+1}{m} \left(\frac{1}{\ell'_1}\right)^m \left(1 - \frac{1}{\ell'_1}\right)^{k+1-m} = 1$ (binomial theorem), so in the end we obtain:

$$\begin{aligned}
\frac{1}{\ell_1(g \circ g')} &= \sum_{k=0}^{\infty} \frac{\gamma_{k+1}}{k+1} \left(1 - \left(1 - \frac{1}{\ell'_1}\right)^{k+1}\right) \\
&= \sum_{k=1}^{\infty} \frac{\gamma_k}{k} \left(1 - \left(1 - \frac{1}{\ell'_1}\right)^k\right) = \frac{1}{\ell_1} - \sum_{k=1}^{\infty} \frac{\gamma_k}{k} \left(1 - \frac{1}{\ell'_1}\right)^k ,
\end{aligned}$$

which proves the theorem. □

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

**Definition 3.1.3.** *Suppose $g$ is a function on $\mathcal{S}$ with* CPS *$\Gamma$. Then*

- *$\kappa(g)$ denotes the* collision average *of $g$. It is the average number of non-zero solutions of Equation (3.2): $\kappa(g) = \sum_{k \geq 1} \gamma_k \cdot k - 1$.*
- *$\nu_i(g)$ denotes the* average tree size *of $g$. It is the average number of elements in a collision tree rooted in $g^i(\mathcal{S})$. More formally, it is the average number of pairs $(x_l, k_l) \in \mathcal{S} \times [1, i]$ such that $g^{k_l}(x_l) = y$ for $y \in g^i(\mathcal{S})$.*

**Theorem 3.1.2.** *Let $g$ be a function with* CPS *$\Gamma$, then for $i < \sqrt{|\mathcal{S}|}$ the shrinking ratio and the average tree size are approximated as follows for large enough $i$:*

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

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

$$\begin{aligned}
\frac{1}{\ell_{i+1}} &= \sum_{k=1}^{\infty} \frac{\gamma_k}{k} \left(1 - \left(1 - \frac{k}{\ell_i} + \frac{k(k-1)}{2 \cdot \ell_i^2} + \mathrm{o}(\ell_i^{-2})\right)\right) \\
&= \frac{1}{\ell_i} \sum_{k=1}^{\infty} \gamma_k \left(1 - \frac{k-1}{2 \cdot \ell_i} + \mathrm{o}(\ell_i^{-1})\right),
\end{aligned}$$

so that we have:

$$\frac{\ell_i}{\ell_{i+1}} = \sum_{k=1}^{\infty} \gamma_k - \sum_{k=1}^{\infty} \frac{\gamma_k \cdot (k-1)}{2 \cdot \ell_i} + \mathrm{o}(\ell_i^{-1}) = 1 - \frac{\kappa/2}{\ell_i} + \mathrm{o}(\ell_i^{-1}) \,,$$

which in turn implies

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

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

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

- $\ell_1$ elements in exactly 1 iteration,
- $\ell_2$ elements in exactly 2 iterations,

  ...

- $\ell_i$ elements in exactly $i$ iterations.

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

Finally, we define the following quantities in the same way as Flajolet *et al.* [FO90].

**Definition 3.1.4.** *We call* cycle length *and* tail length*, denoted respectively $\mu$ and $\lambda$, the average smallest values such that*

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

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

To better understand the behavior of these quantities, we made some experiments. For every $N$ between 12 and 17 included, we generated 100 functions with a given cps and, for each of them, we picked 40 elements at random in $\mathbb{F}_2^N$ and computed $\lambda/2^{N/2}$ and $\mu/2^{N/2}$ for each of them. The average of these values for cpss corresponding to different values of $\kappa$ are given in Fig. 3.3. As we can see, both $\lambda/2^{N/2}$ and $\mu/2^{N/2}$ are almost equal to $\sqrt{\pi/(8\kappa)}$. These results lead us to the following conjecture.

**Conjecture 3.1.1.** *Let $g$ be a function of $\mathcal{S}$ with* cps *$\Gamma$. Experimentally, we found the following values for the tail length $\lambda$ and the cycle length $\mu$:*

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

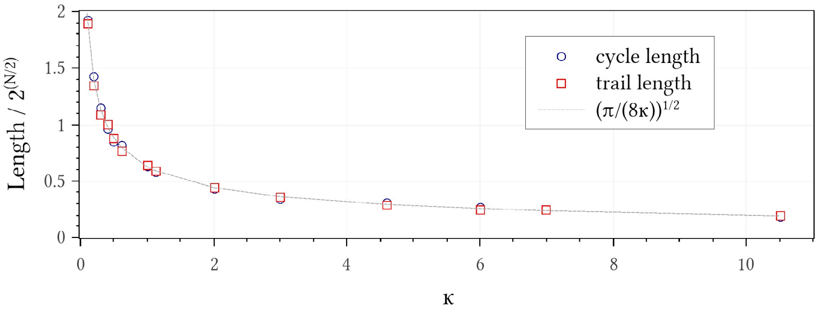


Figure 3.3: Average value of $\lambda/2^{N/2}$ and $\mu/2^{N/2}$ for different $\kappa$.

## 3.2 Other properties of the CPS

The CPS of a function is preserved by some transformations as shown in Lemma 3.2.1. The collision average of $g_1 \circ g_2$ has a simple expression given in Lemma 3.2.2.

**Lemma 3.2.1.** *Let $g$ be a function with CPS $\Gamma$, $P : \mathcal{S} \rightarrow \mathcal{S}$ be a permutation and $J : \mathcal{S} \rightarrow \mathcal{S}$ be injective over $g(\mathcal{S})$. Then $g' = J \circ g \circ P$ has CPS $\Gamma$ as well.*

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

**Lemma 3.2.2.** *Let $g_1$ have collision average $\kappa_1$ and $g_2$ have collision average $\kappa_2$. Then $g_1 \circ g_2$ has collision average $\kappa_1 + \kappa_2$.*

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

- if $g_1(x) = g_1(y)$, which happens in $\kappa_1$ cases on average,
- or if $g_1(x) \neq g_1(y)$ but $g_2\big(g_1(x)\big) = g_2\big(g_1(y)\big)$. There are on average $\kappa_2/\ell_1$ solutions for $g_2(X) = g_2(Y)$ in $g_1(\mathcal{S})$ where $\ell_1$ is the shrinking ratio of $g_1$. However, each of these solutions is the image of $\ell_1$ elements of $\mathcal{S}$ by $g_1$.

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

Lemma 3.2.2 had to hold at least for $g_1 = g_2$ because otherwise we would have had a contradiction with the asymptotic behavior of $\ell_i$ described in Theorem 3.1.2.

### 3.2.1 Independence Assumption in Practice

In this Section, we investigate some results from the literature about particular functions and see how relevant our model is. A summary of this Section is given in Tab. 3.1.

| Function | $\kappa$ | $\ell_1$ | $\ell_i/i$ | $\nu_i/i^2$ | Reference for the CPS |
|---|---|---|---|---|---|
| Mickey's update function | 0.625 | 1.407 | $2^{-1.7}$ | $2^{-2.7}$ | [TBA+13] |
| Random mapping | 1 | 1.582 | $2^{-1}$ | $2^{-2}$ | [FO90] |
| Gluon-64's update function | 6.982 | 3.578 | $2^{1.8}$ | $2^{0.8}$ | Section 3.4.2 |

Table 3.1: Characteristics derived from the CPS of some functions.

#### 3.2.1.1 Random Mappings

The authors of [FO90] study random mappings and give the probability that some $x \in \mathcal{S}$ has $r$ preimages by a random mapping $g$. From this we deduce that the CPS of a random function is given by the Poisson distribution with $\lambda = 1$: $\Gamma = \{e^{-1}/(k-1)!\}_{k \geq 1}$. Our framework implies $\kappa = 1$, $\ell_1 = 1/(1-e^{-1})$ and $\ell_{i+1} = 1/\big(1-\exp(-1/\ell_i)\big)$. These results are coherent with those of [FO90] which provides a good sanity check for our approach.

Furthermore, the authors of [HK05] observed that $\log_2(\ell_i) \approx \log_2(i) - 1$, which also corresponds to $\kappa = 1$ . Finally, the trail and cycle length given in Conj. 3.1.1 match those predicted by [FO90] if we replace $\kappa$ by 1.

#### 3.2.1.2 A5/1

The update function of A5/1 does not satisfy the independence assumption. The author of [Gol97] computed its CPS and established that $\ell_1 = 1.6$ and $\kappa = 1.25$.

If the assumption held, then the probability for an element in $\mathcal{S}$ to be in $g^{100}(\mathcal{S})$ would be about $2^{-6}$, which is very different from the $2^{-2.5}$ actually observed by Biryukov *et al.* [BSW01]. The reason is that the update function A5/1 may keep one of its three LFSR's untouched, which means that $x \in V_j$ and $g(x) \in V_k$ are *not* independent events in its case.

#### 3.2.1.3 MICKEY

The update function of the MICKEY [BD08] stream-ciphers (v1 and v2) fits our model. Hong and Kim [HK05] performed some experiments on the first version of MICKEY and, in particular, estimated the size of $g^{2^k}(\mathcal{S})$ for several values of $k$. Their results are coherent with our model. For instance, they observed that $\log_2(\ell_i)$ (which they denote by $\overline{EL}(f^i)$) is approximated as $log_2(\ell_i) \approx \log_2(i) - 1.8$. The constant term 1.8 implies $\kappa/2 \approx 2^{-1.8}$.

In turn, from the CPS values computed in [TBA+13] (actually, the values $\gamma_k/k$) we obtain the theoretical value $\kappa = 0.625$, which corresponds to a difference of about 7% with the experiments in [HK05].

## 3.3 Improved Collision and Preimage Search

In this section we explore generic collision and preimage search methods against functions with fixed collision spectrum.

### 3.3.1 Basic Collision Search

First, we reformulate the result from Bellare and Kohno [BK04] with our notation.

**Theorem 3.3.1** ([BK04]). *Let $g$ be a function with CPS $\Gamma$, and let $\kappa$ be its collision average. Then the birthday collision attack on $g$ requires about*

$$Q = \sqrt{\frac{|\mathcal{S}|}{\kappa + 1}}. \tag{3.5}$$

*queries to $g$.*

The original paper [BK04] used the parameter *balance* of $g$, denoted $\mu(g)$, which is computed as

$$\mu(g) = \log_{|g(\mathcal{S})|}\left(\frac{|\mathcal{S}|^2}{\sum_{y \in \mathcal{S}} |g^{-1}(y)|^2}\right) \tag{3.6}$$

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

$$\mu(g) \approx 1 - \frac{\log_2\left(\sum_{k=1}^{\infty} k \cdot \gamma_k\right) + \log_2\left(\sum_{k=1}^{\infty} \gamma_k / k\right)}{\log_2\left(|\mathcal{S}|\right)}. \tag{3.7}$$

If Conjecture 3.1.1 holds, then the memory-less collision search based on Floyd's cycle finding algorithm should be $\sqrt{\kappa}$ as fast as in the case of a random function.

### 3.3.2 Collision Attacks on T-Sponges

Now we demonstrate that the entropy loss caused by collisions in the update function of a T-sponge construction, though observable, can be mitigated by a large rate parameter. For a remainder on the T-sponge construction, see Section 1.2.1.3. Here, we denote a sponge-based hash function by $H : \mathbb{F}_2^* \to \mathbb{F}_2^{rj}$, the internal state space by $\mathcal{S} = \mathbb{F}_2^{r+c}$, and the update function by $g : \mathcal{S} \to \mathcal{S}$.

#### 3.3.2.1 Collision Search in T-Sponges

The following theorem shows that to get a significant speed-up in the collision search, the collision average $\kappa$ should be at least of the same magnitude as $2^r$.

**Theorem 3.3.2.** *Let $g$ be a random mapping from $\mathbb{F}_2^{r+c}$ with CPS $\Gamma$. Let $H$ be a T-sponge of capacity $c$ and rate $r$ updated with $g$. Then the probability of success of a brute-force collision attack on $H$ is*

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

*where $Q$ is the number of queries to $g$.*

For a completely random mapping we have $\kappa = 1$, so that the theorem has the same form as in [BDPVA07].

*Proof.* Our proof is a modified version of the one used by Bertoni *et al.* in the paper where they introduced the sponge construction [BDPVA07]. In particular, we use the same terminology: we call the elements of $\mathbb{F}_2^{c+r}$ "nodes" and we partition the space according to the value of the bits in the capacity to obtain $2^c$ "super-nodes", each containing $2^r$ nodes. There is an oriented edge from node $x$ to node $y$ if and only if $y = g(x)$. Finding a collision in $H$ boils down to finding two different paths in this graph starting from points in the super-node with capacity zero to an identical super-node.

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

$$\Pr\left[F(y_i) = k\right] = \frac{\gamma_k}{k} \text{ if } k \geq 1 \text{ , } \Pr\left[F(y_i) = 0\right] = 1 - \frac{1}{\ell_1}$$

which has a mean equal to 1 and a variance equal to $\kappa$.

The value of the fan-in of the super-node $Y$ is the sum of the fan-ins of its nodes:

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

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

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

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

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

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

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

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

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

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

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

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

□

We see that a function with $\kappa > 1$ does not exactly provide $c/2$ bits of security against birthday attacks. Such functions can be found in real cryptographic primitives such as Gluon-64. However, we also immediately see that this effect is small since typical values of $\kappa$ are of order of magnitude 1. Having $\kappa \approx 10$ already requires that the function exhibits significantly non-random properties as is the case with the update function of Gluon-64 which has $\kappa = 6.982$, as we establish later in Section 3.4. Meanwhile, $2^r$ is at least in the hundreds. Thus, the designers of a T-sponge need not really worry about the number of collisions in the update function *provided that the rate is high enough.*

### 3.3.3 Improved Preimage Attack

#### 3.3.3.1 Principle of the Iterated Preimage Attacks

In order to describe the specific type of preimage attack we consider, we need the following concept.

**Definition 3.3.1** (Keyed Walk). *Consider a set $\{g_k\}_{k\in\mathcal{K}}$ of random functions of $\mathcal{S}$ with* CPS*'s $\{\Gamma_k\}_{k\in\mathcal{K}}$ and a fixed starting point $x_0 \in \mathcal{S}$ and let $\{k_1,...,k_l\}$ be a set of $l$ elements of $\mathcal{K}$. We call* keyed walk *the sequence*

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

It can for instance correspond to the successive values of the internal state of a T-sponge or of a Davies-Meyer based Merkle-Damgård hash function as we discuss in the next sections. Consider a keyed walk directed by a sequence $\{k_1,k_2,...,\alpha,\alpha,...,\alpha\}$ ending with $z$ copies of the same symbol $\alpha$. Then, intuitively, much entropy will have been lost because of the $z$ iterations of $g_\alpha$ so that it should be easier to find a second sequence of keys leading to the same final value. This is formalized by the next theorem. A graphical representation of the phenomena it uses is given in Figure 3.4.

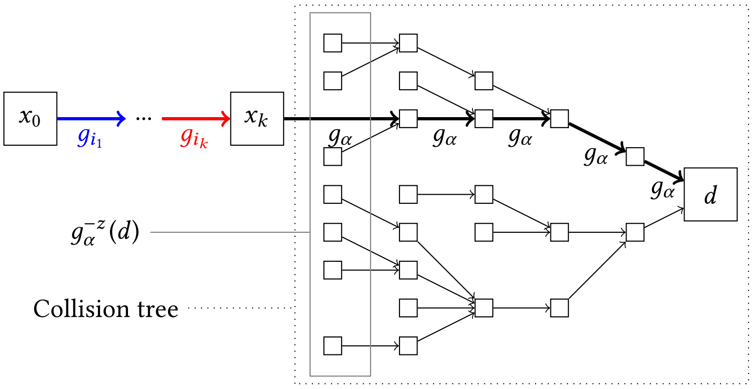


Figure 3.4: The two targets of the iterated preimage attacks on $d$ where $d$ is in $g_\alpha^z(\mathcal{S})$ and $z = 5$. Different colors correspond to different function calls.

**Theorem 3.3.3.** *Let $\{g_k\}_{k\in\mathcal{K}}$ be a set of random mappings of $\mathcal{S}$ with* CPS*'s $\{\Gamma_k\}_{k\in\mathcal{K}}$ and consider a sequence $\{k_1,k_2,...,\alpha,...,\alpha\}$ of $l$ keys from $\mathcal{K}$ ending with $z$ identical keys $\alpha$. Given the final value $d$ of the corresponding keyed walk, the value of $\alpha$ and the number $z$, it is possible to find, for large enough $z$:*

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

2. *a keyed walk ending in $d$ after precisely $z$ calls to $g_\alpha$ in time $|\mathcal{S}| \cdot 2/\kappa$.*

*where $\kappa$ is the collision average of $\Gamma_\alpha$.*

*Proof.* Let $d$ be the final element in the walk. From the structure of the walk, we know that $d \in g_\alpha^z(\mathcal{S})$. Using Theorem 3.1.2, we know that there are $(\kappa/2) \cdot z$ elements in $g_\alpha^{-z}(d)$ and that the collision tree rooted at $d$ contains $(\kappa/4) \cdot z^2$ elements. Therefore, such an element of $g_\alpha^{-z}(d)$ is found with probability $\kappa \cdot z/(|\mathcal{S}| \cdot 2)$ and an element in the collision tree with probability $\kappa \cdot z^2/(|\mathcal{S}| \cdot 4)$.

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

These attacks can be generalized to the case where the end of the message is periodic instead of constant, i.e. if it ends with $z$ copies of $(\alpha_1, \alpha_2, ..., \alpha_p)$. We simply need to replace $g_\alpha$ by $g' = g_{\alpha_1} \circ ... \circ g_{\alpha_p}$. The $\kappa$ involved in the complexity computations is then that of $g'$, i.e. $\sum_{i=1}^{p} \kappa_i$ where $\kappa_i$ is the collision average of $g_{\alpha_i}$ (see Lemma 3.2.2 in Section 3.2). The constraint on $z$ being large is only such that we can assume that $z$ has the asymptotical behaviors described in Theorem 3.1.2.

#### 3.3.3.2 Application to a T-Sponge

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

While the flat sponge claim provides a good description of the security offered by a sponge (be it a T-sponge or a P-sponge) against collision search and, for P-sponge, against second preimage search, there is a gap between the number of queries it allows and the best algorithm known for preimage search. In particular, there is to the best of our knowledge no algorithm allowing a preimage search with complexity below $2^c$ calls to the sponge function.[1] Theorem 3.3.4 bridges the gap between the $2^{c/2}$ bound of the flat sponge claim and the $2^c$ bound for preimage search by applying Theorem 3.3.3 to the T-sponge structure.

**Theorem 3.3.4.** *Let $H$ be a T-sponge with update function $g$, and let $\kappa$ be the collision average of $g$. Let $M$ be a message such that its last $z$ injections to the sponge are identical. Then a preimage to $H(M)$ can be found with complexity*

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

Such messages can be quite common. For instance, the last $z$ calls of $g$ can be blank calls for the sole purpose to slow down the hashing as suggested by the Keccak

[1] This case corresponds to the case where the attacker inverts the squeezing operations in time $2^c$ to retrieve the last internal state of the sponge before the squeezing and then uses a meet-in-the middle approach to find a valid message leading to this internal state in time $2^{c/2}$ (see [GPP11]). However, this second step cannot be carried out in the case of a T-sponge since the update function cannot be inverted.

team in a NIST presentation [BDPVA13].[2] Such an attack can be prevented by setting an upper-bound of about $2^{r+2}/\kappa$ for the length of the message which in turn means that $r$ has to be high in a T-sponge.

#### 3.3.3.3 Similarity to the Herding Attack

The *herding attack* was introduced in [KK06] and is also referred to as the Nostradamus attack. In a herding attack, an attacker commits to a digest $d$ and, when given a challenge $P$, has to find a suffix $S$ such that $H(P||S) = d$. To achieve this she builds, during an offline phase, a so-called *diamond structure* which is essentially a binary collision tree with $2^{\ell}$ nodes and rooted at $d$. The nodes of the tree contain the value of the internal state as well as the message block which needs to be absorbed to go to its child. During the online phase, she uses the diamond to find efficiently the suffix $S$: all she has to do is find a way to reach any of the $2^{\ell+1} - 1$ nodes in the diamond from the given starting point.

#### 3.3.3.4 Application to a Merkle-Damgård Construction

When a block cipher is used in Davies-Meyer mode to build a Merkle-Damgård-based hash function, the successive chaining values $h_i \in \mathcal{S}$ are obtained from the previous one and the $i$-th message block: $h_i = E_{m_i}(h_{i-1}) \oplus h_{i-1} = g_{m_i}(h_{i-1})$. Because of the feedback of $h_{i-1}$, we do not expect $g_k$ to be a permutation and, therefore, expect such a construction to be vulnerable to iterated preimage attacks. The padding used for Merkle-Damgård constructions usually takes into account the length of the message so that we need a message of the same length. Therefore, it is not enough to aim at an element in the collision tree, we need to find an element which is precisely in $g_{\alpha}^{-z}(d)$ so that a preimage search requires $2^{N+1}/\kappa$: if the CPS of $g_k$ is such that $\kappa > 2$ then the iterated preimage attack is more efficient than brute-force. Furthermore, if there is an efficient way around the padding (e.g., with expandable messages [KS05]), the efficiency becomes $2^{N+2}/(\kappa z)$ where $N$ is the size of the internal state of the hash function.

## 3.4 Preimage Attack on GLUON-64

### 3.4.1 The GLUON- Family of Hash Functions

Introduced in [BDM+12], the GLUON family of hash functions consists of three members corresponding to different security levels: GLUON-64, GLUON-80 and GLUON-112. They have a T-sponge structure and have characteristics summarized in Table 3.2.

The function $g$ used to update the internal state has the same structure in the three cases. It can be seen as a stream-cipher based on a Feedback with Carry Shift Register (FCSR). The concept of FCSR has evolved through time as the first stream-cipher [AB05] based on a component bearing this name got broken [HJ11]. When we talk about FCSR in this paper, we refer to the last version of this structure, i.e. the one used in X-FCSR v2 [ABL+09] and, of course, GLUON. For example, a representation of the FCSR used by GLUON-64 is given in Fig. 3.5 where blue arrows correspond to

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

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

Table 3.2: Characteristics of the hash functions of the GLUON family.

feedbacks shifted to the right and red ones to the left. The value of the shift is given by the label of the arrow.

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

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

where "$\ll s$" is a C-style notation for the shift of the content of a cell by $s$ bits to the left and $\oplus$ and $\wedge$ are respectively the bitwise addition and multiplication in $\mathbb{F}_2^r$.

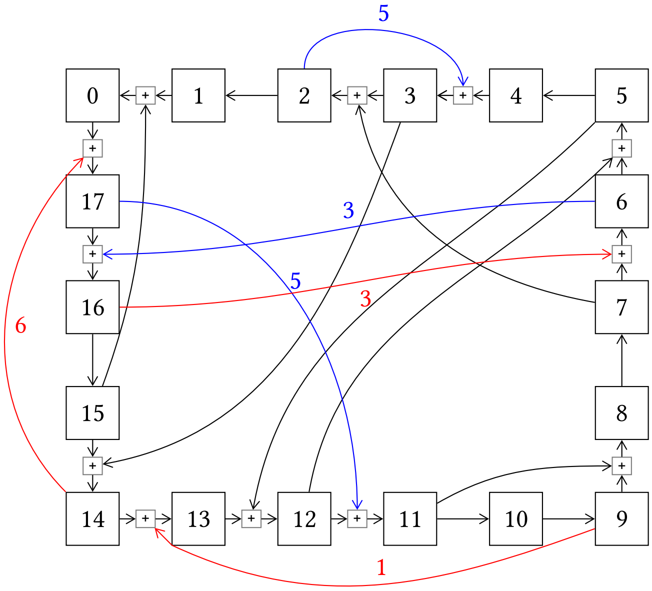


Figure 3.5: The FCSR used in GLUON-64.

The update function of every member of the GLUON family is made of three steps: padding of the input and loading in an FCSR (pad), clocking of the FCSR ($\rho$) and filtering $\Phi$. We describe these steps separately.

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

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

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

Overall, $g$ is equal to $\Phi \circ \rho^{d+4} \circ \text{pad}$. The function pad is a bijection and we shall consider that the restriction of $\Phi$ over the set of the pairs main register/carry register reachable after $d + 4$ calls to $\rho$ starting in the image of pad is collision-free. The designers of GLUON claim:

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

However, this claim raises an obvious question. What happens during these first rounds, *before* the main cycle is reached?

It turns out that we can answer this using a SAT-solver. It is possible to encode the equation

$$(\rho^k \circ \text{pad})(a + x) = (\rho^k \circ \text{pad})(x) \tag{3.8}$$

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

We solved (3.8) for many values of $a$ and for $k = 10$ for each member of the GLUON family. While no non-zero solutions were found for any $a$ for GLUON-80 and GLUON-112, it turns out that (3.8) has many solutions for GLUON-64. We used Algorithm 3.1 to find which $V_k$ several random elements $a \in \mathcal{S}$ belong to by enumerating all the values of $x$ such that (3.8) holds. This algorithm works by solving (3.8) for $x$, thus (possibly) finding a solution $x_1$; then solving (3.8) with the constraint that the solution must be different from $x_1$ in at least one bit, thus (possibly) finding $x_2$, etc. until no more solutions can be found. If there are $k$ such $x \neq 0$, then $a$ is in $V_{k+1}$.

**Algorithm 3.1** Enumerating all the solutions of $g(a + \delta) = g(a)$.
**Inputs:** function $g$, value $a$;
**Output:** list of all $\delta$ such that $g(a + \delta) = g(a)$.

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

### 3.4.2 CPS and Preimage Attack on GLUON-64

We ran Algorithm 3.1 for Gluon-64 on 24,000 different elements chosen uniformly at random in $\mathcal{S} = \mathbb{F}_2^{r+c}$. This allowed us to approximate the cps of the update function. Our results are summarized in Figure 3.6. Note that non-zero $\gamma_k$ were observed well after $k = 20$.

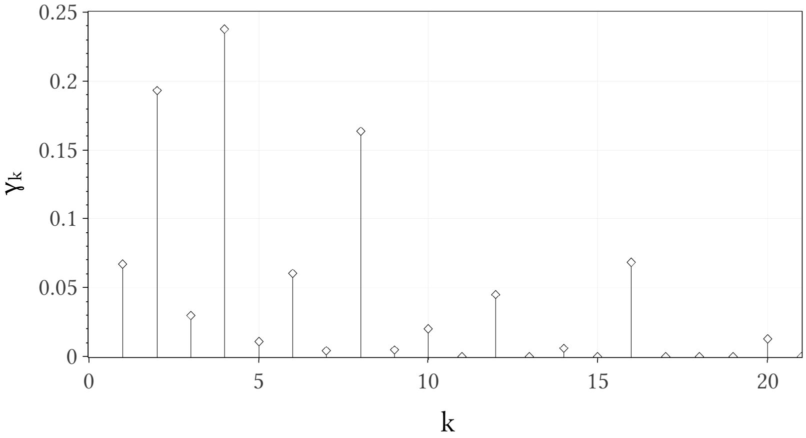


Figure 3.6: Approximation of the cps of the function used by Gluon-64.

We found that $\gamma_1 = 0.065$, $\ell_1 = 3.578$ and $\kappa = 6.982$ which are much worse than what one should expect from a random function, namely $\gamma_1 = e^{-1} \approx 0.368$, $\ell_1 = 1/(1 - e^{-1}) \approx 1.582$ and $\kappa = 1$. This means that finding a preimage in a scheme equivalent to appending $z$ identical words at the end of the message has a complexity of $2^{136+2}/(6.982 \cdot z) = 2^{128} \cdot (146.7/z)$. For $z > 147$, this is more efficient than classical brute-force. The complexities for some values of $z < 2^{(r+c)/2} = 2^{68}$ are given in

Table 3.3.

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

Table 3.3: Complexity $C$ of a preimage search for $d = H(m)$ where $H$ is Gluon-64 and $m$ is unknown except for $z$ identical bytes in its end.

The number of possible values for the internal state of Gluon-64 after the absorption of different values is described in Figure 3.7. It increases when a message block is absorbed (green vertical arrow) but decreases when it absorbs a constant several times (red vertical arrow). As we can see, it drops after each call to $g$, although the amplitude of this decrease gets smaller as $g$ is iterated.

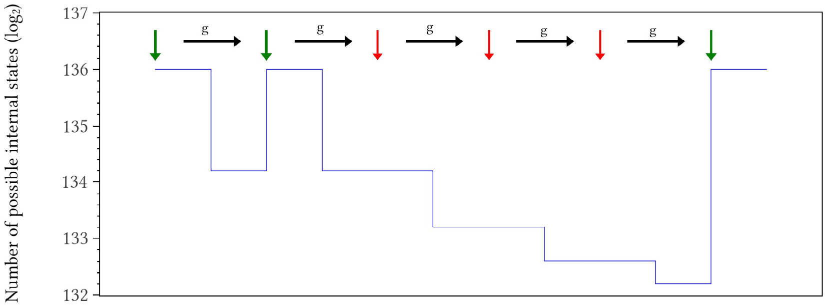


Figure 3.7: Evolution of the number of possible values for the internal state of Gluon-64.

Chapter 4

# Differential and Structural Analysis of Prince

While impractical attacks still provide the academic community with valuable insights into the security provided by different block ciphers, their components, their design strategies, etc., cryptanalysis in the industry is more focused on practical attacks. In order to promote this view, the Technical University of Denmark (DTU), NXP Semiconductors and the Ruhr University of Bochum challenged the cryptographic community[1] with finding low data complexity attacks on the block cipher Prince [BCG+12]. More precisely, they accepted attacks requiring only at most $2^{20}$ chosen plaintexts or $2^{30}$ known plaintexts. Furthermore, extra rewards (from 1000 to 10000€) were given for attacks on at least 8 rounds which require at most $2^{45}$ bytes of memory (about 32 Terabytes) and at most $2^{64}$ encryptions of the round-reduced variant attacked.

Studying Prince in this setting may provide valuable data on multiple accounts. First of all, Prince implements a simplified version of the so-called fx construction: encryption under key $(k_0||k_1)$ consists of xoring $k_0$ to the plaintext, applying a block cipher called Prince-core keyed with $k_1$ and then output the result xored with $L(k_0)$ where $L$ is a simple linear bijection. This strategy allows for a greater key size without the cost of a sophisticated key schedule. However, it is impossible to make a security claim as strong as for a more classical construction. Second, Prince-core introduced a property called $\alpha$-reflection. If we denote by $\text{EC}_{k_1}$ the encryption under Prince-core with subkey $k_1$, then the corresponding decryption operation is $\text{EC}_{k_1 \oplus \alpha}$ for a constant $\alpha$. In other words, decryption is merely encryption under a related-key. The consequences of this property have already been studied and, in particular, some values of $\alpha$ different from the one used have been showed to lead to weaker algorithms [SBY+15].

Furthermore, several ciphers were developed that implement the same property in a similar fashion: Mantis [BJK+16] and Qarma [Ava17].

Prince has already been the target of several attacks, notably [JNP+14b] where the security of the algorithm against multiple attacks was assessed, [SBY+15] which investigated the influence of the value of $\alpha$, [LJW13] which described Meet-in-the-Middle attacks on the block cipher and, finally, [CFG+15] proposed the best attack to date in terms of time complexity for a high number of rounds. A list of the crypt-

[1] The Prince challenge, issued by NXP Semiconductors, is described on the following webpage: https://www.emsec.rub.de/research/research_startseite/prince-challenge/.

analyses of round-reduced Prince is provided in Table 4.1, where time complexity is measured in encryption units and Memory complexity is measured in 64-bit blocks. Attacks working only on Prince-core or for modified versions of Prince (different $\alpha$ or S-Box) are not shown. The attacks presented in [GR16, RR16a, RR16b] were published after the paper containing the results presented in this chapter [DP15]. As I was not involved in deriving the Meet-in-the-Middle attacks presented in this paper, I do not present them in this thesis. The interested reader should refer to our paper to learn more about those.

| Reference | Type | Rounds | Data (CP) | Time | Memory |
|---|---|---|---|---|---|
| [JNP+14b] | Integral | 4 | $2^{4}$ | $2^{64}$ | $2^{4}$ |
| | | 6 | $2^{16}$ | $2^{64}$ | $2^{6}$ |
| Section 4.2 | Diff. / Logic | 4 | $2^{10}$ | 5s | $<< 2^{27}$ |
| | | 6 | $2^{14.9}$ | $2^{32.9}$ | $<< 2^{27}$ |
| [DP15] | MitM | 6 | $2^{16}$ | $2^{33.7}$ | $2^{31.9}$ |
| | | 8 | $2^{16}$ | $2^{50.7}$ (online) | $2^{84.9}$ |
| | | 8 | $2^{16}$ | $2^{65.7}$ (online) | $2^{68.9}$ |
| | | 10 | $2^{57}$ | $2^{68}$ (online) | $2^{41}$ |
| [LJW13] | MitM | 8 | $2^{53}$ | $2^{60}$ | $2^{30}$ |
| | | 9 | $2^{57}$ | $2^{64}$ | $2^{57.3}$ |
| [CFG+15] | Multiple diff. | 9 | $2^{46.89}$ | $2^{51.21}$ | $2^{52.21}$ |
| | | 10 | $2^{57.94}$ | $2^{60.62}$ | $2^{61.52}$ |
| [GR16] | Truncated diff. | 4 | $2^{3}$ | $2^{18.25}$ | small |
| | | 4 | $2^{8.75}$ | $2^{8.15}$ | small |
| [RR16a] | Acc. brute-force | 6 | 2 (KP) | $2^{96.8}$ | small |
| | | 6 | 2 (KP) | $2^{94.05}$ | $2^{24.6}$ |
| [RR16b] | Acc. brute-force | 8 | 2 (KP) | $2^{122.7}$ | small |
| | Acc. brute-force | 10 | 2 (KP) | $2^{124.1}$ | small |
| | Acc. brute-force | 12 | 2 (KP) | $2^{125.1}$ | small |
| | MitM | 8 | 2 (KP) | $2^{109.3}$ | $2^{65}$ |
| | MitM | 10 | 2 (KP) | $2^{122.2}$ | $2^{53.3}$ |

Table 4.1: The best attacks on round-reduced Prince in the single-key model.

As stated before, most of the attacks usually presented in the literature have impractical complexities. For instance, differential attacks and linear attacks require large amounts of chosen (respectively known) plaintexts, both of which may be impossible to gather to begin with if the algorithm is implemented on a small-device with little computer and, hence, a small throughput. Therefore, we focused our efforts on Meet-in-the-Middle (MitM) attacks, algebraic/logic attacks where the fact that a ciphertext is the encryption of a plaintext is encoded as an equation which is fed to a solver and, surprisingly, differential attack for which we found a heuristic method decreasing significantly the data complexity.

In this chapter, I describe different low data complexity attacks on round-reduced

Prince which were submitted to the Prince challenge and which turned out[2] to be some of the best ones on Prince reduced to 6 rounds.

In Section 4.2 (p. 76), we show how the equation given to a SAT-solver can be modified so as to make an attack on 4 rounds practical. We also explain how the power of the filter used to discard wrong pairs in a differential attack can be raised to the power 4 when attacking 6-round Prince by considering groups of pairs. Combining these two ideas, we attack 6-round Prince using a differential attack to recover half of the key and a SAT-solver to recover the other half. In Section 4.3 (p. 83), we present some observations about the cycle structure of the internal rounds of Prince and how they imply the existence of alternative representations of the cipher highlighting a poor diffusion in some subsets of the input space. While we do not use these to attack Prince directly, we show that the size of these subsets is far smaller than what would be expected for a random permutation and actually find such sets for the 4-round Prince-core.

## 4.1 Specification of Prince

Prince is a 64-bit block cipher with a 128-bit key. It is based on a variant of the the FX-construction which was proposed by Kilian and Rogaway [KR96] as a generalization of the DESX scheme. The master key $k$ is split into two 64-bit parts $k = k_0 \parallel k_1$ and $k_0$ is used to generate a third subkey $k'_0 = (k_0 \ggg 1) \oplus (k_0 \gg 63)$. Both $k_0$ and $k'_0$ are used as pre- and post- whitening keys respectively. The full version of the cipher has 12 rounds and is depicted on Figure 4.1.

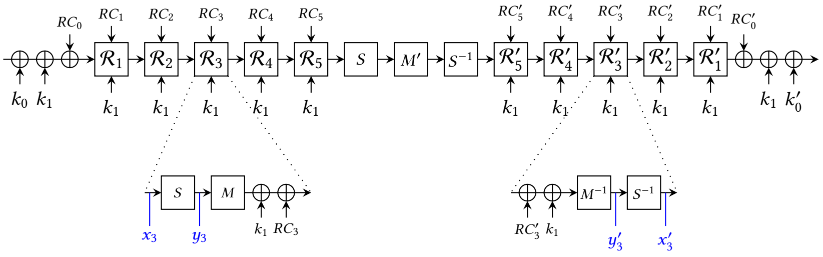


Figure 4.1: The Prince cipher.

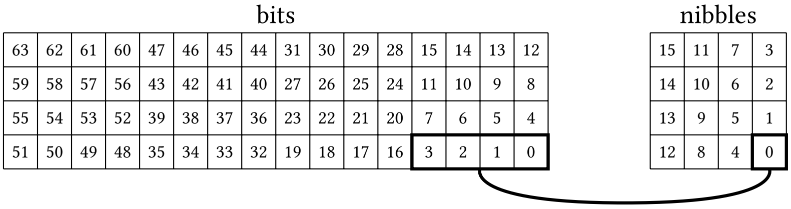
The encryption is somewhat similar to the AES and consists of a nibble-based substitution layer $S$ and a linear layer $M$. The operation $M$ can be divided into a ShiftRows operations and a matrix multiplication $M'$ operating independently on each column but not nibble-oriented. Furthermore the matrix $M'$ is an involution and, combined to the fact that the round constants satisfy the relation $RC_i \oplus RC'_i = \alpha$ where $\alpha$ = C0AC29B7C97C50DD, the decryption process $D_{k_0,k_1,k'_0}$ is equal to the encryption process $E_{k'_0,k_1 \oplus \alpha,k_0}$. For further details about Prince we refer the reader to [BCG+12].

In this chapter, we denote the plaintext and the ciphertext by $p$ and $c$ respectively. For the first $R$ rounds of $2R$-round Prince, we denote the internal state just before (resp. after) the $r$-th SubNibble layer by $x_r$ (resp. $y_r$) while for the last $R$ rounds those

[2] This results was announced at the rump session of Crypto'14 by Christian Rechberger. The slides of the corresponding talk, "Update on the 10000 Euro Prince cipher-breaking challenge: Results of Round-1", are available online: http://crypto.2014.rump.cr.yp.to/d037206eda8f9278cef1ea26cd62e51f.pdf. Against 6-round Prince, my attacks were *ex aequo* with Patrick Derbez's.

internal states are denoted by $y'_r$ and $x'_r$ respectively as shown on Figure 4.1. Given a collection of messages $\{p^0, \ldots, p^m, \ldots\}$, the notation $x_r^m[i]$ holds for the nibble $i$ of the state $x_r$ of the message $p^m$. As PRINCE is not fully nibble-oriented we use the notation $x_r[i]_b$ to refer to the bit $i$ of the state $x_r$ and the following relation holds for all $i \in \{0, \ldots, 15\}$ :

$$x_r[i] = x_r[4i+3]_b \parallel x_r[4i+2]_b \parallel x_r[4i+1]_b \parallel x_r[4i]_b.$$



bits

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 63 | 62 | 61 | 60 | 47 | 46 | 45 | 44 | 31 | 30 | 29 | 28 | 15 | 14 | 13 | 12 |
| 59 | 58 | 57 | 56 | 43 | 42 | 41 | 40 | 27 | 26 | 25 | 24 | 11 | 10 | 9 | 8 |
| 55 | 54 | 53 | 52 | 39 | 38 | 37 | 36 | 23 | 22 | 21 | 20 | 7 | 6 | 5 | 4 |
| 51 | 50 | 49 | 48 | 35 | 34 | 33 | 32 | 19 | 18 | 17 | 16 | 3 | 2 | 1 | 0 |

nibbles

| | | | |
|---|---|---|---|
| 15 | 11 | 7 | 3 |
| 14 | 10 | 6 | 2 |
| 13 | 9 | 5 | 1 |
| 12 | 8 | 4 | 0 |

Figure 4.2: Ordering of the bits/nibbles in PRINCE.

Finally, we use the following notation for some functions.

$R$ The composition of $S$ and $M$ so that $R(x) = M\big(S(x)\big) = SR\big(M'(S(x))\big)$.

$\mathrm{E}^r_{k_0||k_1}$ PRINCE reduced to $r$ rounds.

$\mathrm{EC}_{k_1}$ full PRINCE-core.

$\mathrm{EC}^r_{k_1}$ PRINCE-core reduced to $r$ rounds.

## 4.2 Combining a Differential Attack with a SAT-Solver

In this section, I describe how PRINCE can be attacked with a SAT-solver. The basic idea, *Differential over Definition*, is introduced in Section 4.2.1 (p. 76) where a practical attack against 4-round PRINCE is presented. Then, a particular differential pattern is exploited in Section 4.2.2 (p. 79) to attack 6-round PRINCE. While this attack is based on a differential cryptanalysis, the final key recovery is performed with a SAT-solver. The implementation of this attack is described in Section 4.2.3 (p. 82).

### 4.2.1 Attacking 4-Round PRINCE with a SAT-Solver

#### 4.2.1.1 Encoding PRINCE as a CNF Formula

The idea is to generate a CNF formula where a set $p$ of Boolean variables correspond to the 64 bits of the plaintext, $c$ to the 64 bits of the ciphertext and $k$ to the 128 bits of the key, and such that there exists a unique assignment of the variables satisfying the CNF corresponding to the case $E_k(p) = c$.

Hence, if we generate such a formula, set the variables in $p$ and $k$ to a chosen value and use a SAT-solver to find an assignment satisfying the CNF formula, the variables in $c$ will correspond to the ciphertext. Solving such a formula is easy, an observation which we can relate to the fact that the evaluation of a block cipher has to be "easy" from the point of view of complexity theory.

Another way to use such a formula is to fix the variables in $p$ and in $c$ according to a known plaintext/ciphertext pair, solve the CNF and recover the key from the variables corresponding to it. Unless the number of rounds is very small (at most 3 in the case of PRINCE), solving such a system is impractical. Again, we can relate this observation to the fact that recovering the key given one or several plaintext/ciphertext pairs has to be "hard". Our approach consists of using some knowledge about the internal state of the cipher to simplify the task of the SAT-solver and make such a resolution possible for a higher number of rounds.

In order to encode a PRINCE encryption as a CNF formula, we introduce several sets of 64 Boolean variables corresponding to each step of each round: one for the internal state at the beginning of the round ($x_r$), one for the internal state after going through the S-Box ($y_r$), etc. We also use Boolean variables corresponding to the key bits.

Our task is then to create a CNF formula connecting these variables in such a way as to ensure that, for instance if $k[0,...,63]$ is fixed, it has only one solution where $y_r[0,...,63]$ is indeed the image of $x[0,...,63]_r$ by $S$, etc.

In order to encode the linear layer, we use the alternative representation of $M'$ from [CFG+15] where it was shown that $M'$ operates on columns of 4-bits independently by first rotating them by a column-dependent number of bit and then XORing the parity of the column in each bit. We thus add variables corresponding to the parity of the columns and encode the corresponding XORs as CNF formulas. The $SR$ operation is only a permutation of the bits so we simply set the corresponding bits to be equal.

The encoding of the S-Box is less simple to obtain. In order to find the best one, we chose to look for it directly instead of using the ANF as an intermediate step. Indeed, since the S-Box is 4x4, it is small enough for us to brute-force all clauses[3] involving input and output bits and check if they hold for every input.

Doing this lead us to find 29 clauses with 3 variables. However, they are not sufficient to completely specify the S-Box so we used a greedy algorithm to find the best clauses with 4 variables to add to this encoding. In the end, we have 29 clauses with 3 variables and 9 clauses with 4 variables which are such that the only solutions of the CNF made of all these clauses are all the assignments corresponding to pairs $\big(x, S(x)\big)$ for all $x \in [0,15]$.

These clauses with 3 variables can be interpreted as simple implications. For example, if $o[3,...,0]_b = S(i[3,...,0]_b)$ then the following two clauses hold with probability is 1 :

$$\big(i[1]_b \vee o[2]_b \vee o[3]_b\big) \wedge \big(i[1]_b \vee o[1]_b \vee o[2]_b\big).$$

They are logically equivalent to the following implication:

$$\overline{i[1]_b} \implies \big((o[2]_b \vee o[3]_b) \wedge (o[1]_b \vee o[2]_b)\big).$$

#### 4.2.1.2 Differential Over Definition

The approach consisting of using the knowledge from a differential trail to ease the task of a SAT-solver used to attack a cryptographic primitive has been explored in [MZ06] in order to attack MD4 and MD5. The authors of this paper first use heuristic methods to find a high probability differential trail leading to a collision and then use a SAT-solver to find a pair of messages which satisfies this trail. In the same paper, we can find the following observation:

[3] A clause is the logical OR of several variables. For example, if $\overline{x}$ is the negation of $x$ then each of $a \vee b$, $a$ and $\overline{a} \vee b \vee \overline{c}$ are clauses.

> An interesting result of our experiments with SAT-solvers is the importance of having a differential path encoded in the formula.

As we shall see, this also holds for block ciphers. Attacking 4 rounds Prince-core takes more than 10 hours if we simply encode as a CNF that some plaintexts are encrypted into known ciphertexts but we can drastically reduce this time while breaking Prince with its whitening keys using *differential over-definition.*

**Definition 4.2.1.** *We call* Differential Over Definition *the following algorithm which simplifies a* CNF *formula using the knowledge that the variables correspond to bits of the internal state of an encryption following a certain trail.*

*For all pairs of variables in the* CNF*, proceed as follows:*

- *If they are assumed to be equal, replace all occurrences of the first one by the second one.*
- *If they are assumed to be different, replace all occurrences of the first one by the negation of the second one.*

While the idea behind this algorithm is simple, it is necessary for cryptographers to implement it efficiently “by hand”. Indeed, the only input of a SAT-solver is a CNF formula, i.e. merely a list of clauses from which deriving what variables are equal to each other without knowledge of the structure of the problem is far from trivial. For instance, it would be necessary for the SAT-solver to “understand” that the set of clauses used to model one S-Box call all correspond to a unique function so that identical inputs lead to identical outputs; all this without having any distinction between the input and output bits. That is why differential over-definition, an easy algorithm for the cryptographer to implement, is a valuable pre-processing step when using a SAT-solver for cryptography. Indeed, it leads to gains in time complexity of several orders of magnitude.

This algorithm can be implemented efficiently using a hashtable containing the correspondences between the variables. Once this algorithm has been run, the CNF is over defined: the solution would have been such that the equalities hold anyway but there are less variables and less clauses in the CNF. However, if the pair actually does not follow the trail, the CNF has become unsatisfiable. This is a difference between our work and the one described in [MZ06]: we do not always know before hand if the CNF has a solution. We can think of this as a trade-off between “solving one CNF known to be true” and “solving many over-defined CNF’s which may or may not be true”: the second approach loses time by requiring several calls to a SAT-solver but each of these calls takes less time thanks to the over-definition.

Such an over definition can be used in different ways.

1. It can be used to directly attack a cipher. By propagating only the zero differences holding with probability 1 inside a group of 8 encryptions with many zero input differences, we can reduce the time complexity of an attack on 4 rounds from more than 10 hours to a few seconds (see below). Furthermore, such a formula is always true.
2. It can be used for key-recovery in a differential attack. Instead of implementing an algorithm recovering the key from a pair following a particular trail by peeling off layer after layer of encryption in our attack on 6 rounds described in Sections 4.2.2 (p. 79) and 4.2.3 (p. 82), we simply re-used the code of our

attack on 4 rounds and over-defined the cnf modeling the encryptions of right pairs according to the high probability trail we used.

We implemented the attack described in Algorithm 4.1 to attack 4-round Prince (with its whitening keys) using the sat-solver Minisat [ES03] and obtained an average total time of 5.13s and average time spent solving the cnf of 3.06s. The designers of Prince did not consider SAT-based attacks but they did investigate algebraic attacks. They manage to attack 4-round Prince-core in less than 2s while our attack requires about 5s to attack 4-round Prince, a cipher which uses twice as much key material.

**Algorithm 4.1** Differential over-Definition based attack against 4-round Prince.
**Input:** 4-round Prince instance ;
**Output:** secret key $k_0||k_1$

```
Query $2^{10}$ plaintexts/ciphertexts where the first 10 bits take all possible values.
Select a subset of 8 plaintexts/ciphertexts maximizing the number of 0-differences in the output.
Encode the 8 encryptions as a cnf $A$.
Overdefine $A$ by propagating zero-differences with probability 1.
Use a sat-solver to retrieve the key bits from $A$
return $k_0||k_1$
```

### 4.2.2 Amplified Differential Trails

Our attacks rely on some differences propagating identically in different pairs. To better describe this, we introduce the following definitions.

**Encryption** We call *encryption* a pair plaintext/ciphertext encrypted under a fixed key.

**Pair** A *pair* is a set of two encryptions where the plaintexts are separated by a known difference.

**Family** A *family* is a group of pairs with a particular structure. They are generated from a single pair $\{(p[0],...,p[b-1]),(p'[0],...,p'[b-1])\}$, where $p[i]$ and $p'[i]$ are nibbles. Suppose that the input difference covers the first three nibbles so that $p[3]=p'[3]=c[3],...,p[b-1]=p'[b-1]=c[b-1]$ for some constants $c[i]$. Then the family corresponding to this pair is made by exchanging some nibbles between the two encryptions in the pair so as to obtain the following pairs:

$$\left\{\begin{array}{c}(p[0],p[1],p[2],c[3],...,c[b-1])\\(p'[0],p'[1],p'[2],c[3],...,c[b-1])\end{array}\right. \quad \left\{\begin{array}{c}(p'[0],p[1],p[2],c[3],...,c[b-1])\\(p[0],p'[1],p'[2],c[3],...,c[b-1])\end{array}\right.$$

$$\left\{\begin{array}{c}(p[0],p'[1],p[2],c[3],...,c[b-1])\\(p'[0],p[1],p'[2],c[3],...,c[b-1])\end{array}\right. \quad \left\{\begin{array}{c}(p[0],p[1],p'[2],c[3],...,c[b-1])\\(p'[0],p'[1],p[2],c[3],...,c[b-1]).\end{array}\right.$$

Overall, if there are $n$ nibble with non-zero differences in the input then a family is made of $2^{n-1}$ pairs and $2^n$ encryptions.

In the case of Prince, we consider differential trails where the input differences are only over one column and such that all the pairs in a family follow the same trail for the first three rounds. For example, the trails we describe in Section 4.2.2.1 (p. 80) are either followed by all the elements in a family or none of them. A similar heuristic is used in [BS01] to perform a multiset attack on the SASAS structure.

This behavior comes from the fact that the transitions in the trails we study depend only on the transitions occurring during the first round, which are the same in all pairs of a family, and on the actual value of some nibbles that the differences have not had time to reach, and which are the same in all encryptions of the structure.

In a recent paper [Tie16], Tiessen introduced the notion of *polytopic cryptanalysis*. It generalizes differential attacks by considering the propagation of differential patterns linking a set of more than 2 encryptions (a *polytope*). The attack targeting a family rather than a simpler pair can be seen as a special case of polytopic cryptanalysis. Using this framework, each family is a polytope and the differential trail that is followed by all or none of the pairs in the family is a polytopic trail with a particularly high probability.

#### 4.2.2.1 Our Trails

There has already been some differential cryptanalyses of Prince, see for example [CFG+15], which is the best attack to date, and also [ALL12].

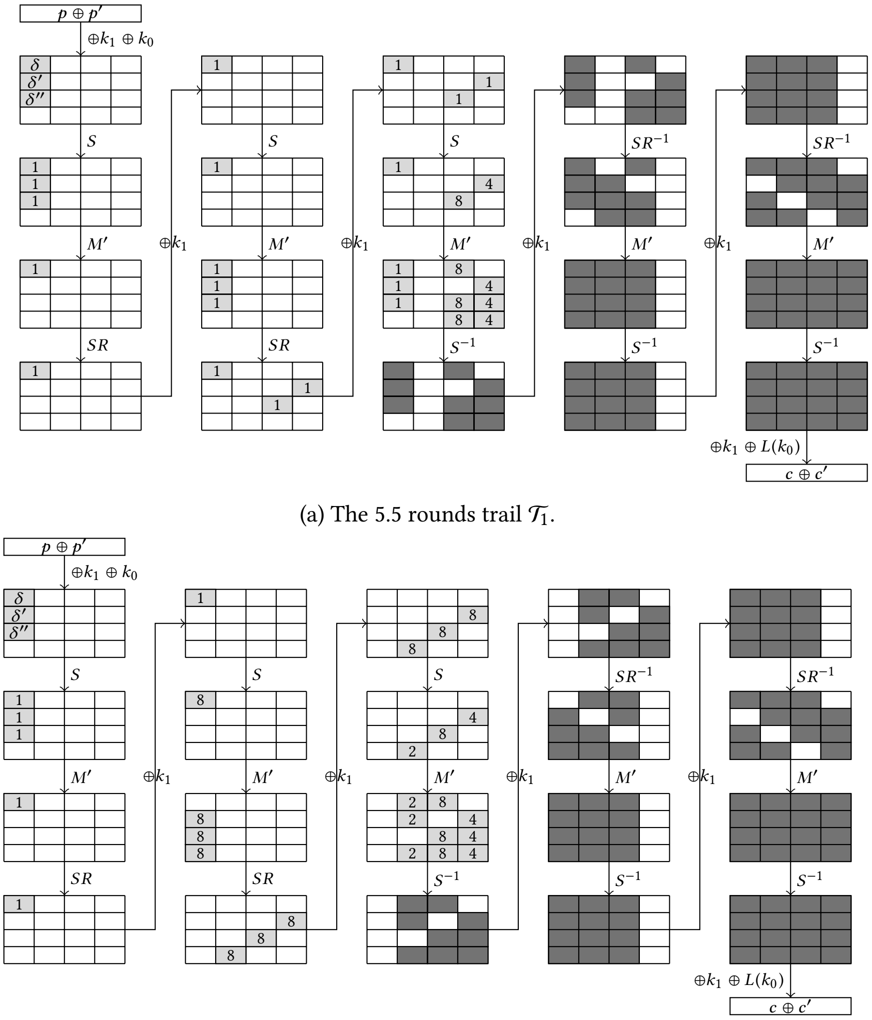
Here, we consider trails which are completely specified during the first 3 rounds and then propagate with probability 1 for 2.5 rounds before having spread to the full internal state. Figure 4.3a shows a first trail covering 5.5 rounds in this way, which we denote $\mathcal{T}_1$. Each array corresponds to the differences between the internal states of two encryptions under 6-round Prince and each cell gives the value of the difference: light gray corresponds to a fully specified non-zero value at the nibble level (e.g. a difference of 1), dark gray to an unknown non-zero difference and white to a zero difference. A very similar trail with a probability 2 times smaller, $\mathcal{T}_2$, is given in Figure 4.3b. To compute their probabilities, we use the difference distribution table of the S-Box, as defined later in Section 8.2.1 (p. 139). If we let the input difference be $(1,1,1,0,\dots,0)$, then $\mathcal{T}_1$ has a probability of $2^{-2\cdot 3} \cdot 2^{-2} \cdot 2^{-2-2-3} = 2^{-15}$ and $\mathcal{T}_2$ has a probability of $2^{-2\cdot 3} \cdot 2^{-2} \cdot 2^{-2-3-3} = 2^{-16}$.

Querying enough families at random to find one right family for any of these would require $(2^{-15}+2^{-16})^{-1} = 2^{14.41}$ families with an input difference over 3 nibbles, i.e. $2^{14.41} \cdot 2^{3} = 2^{17.41}$ encryptions. However, we can use structures to decrease this complexity.

| Hexadecimal | `0x1` | `0x2` | `0x4` | `0xb` | `0xc` | `0xd` |
|---|---|---|---|---|---|---|
| Binary | `0001` | `0010` | `0100` | `1011` | `1100` | `1101` |
| Probability | 1/4 | 1/8 | 1/8 | 1/8 | 1/4 | 1/8 |

Table 4.2: Input differences possibly mapped to 1 by the S-Box of Prince.

The input differences which might lead to an output difference of 1 are those listed in Table 4.2. As we can see, the second bit from the right in little-endian notation is only involved in `0x2` and `0xb` which, taken together, only have a probability of 1/4 of leading to a difference of 1. Hence, we use the following structures where b



(a) The 5.5 rounds trail $\mathcal{T}_1$.

(b) The trail $\mathcal{T}_2$.

Figure 4.3: The two 5.5-round trails we use.

is a bit taking all possible values and c is constant across the structure:

bbcb bbcb bbcb cccc cccc ... cccc.

We found experimentally that such structures contain several[4] right families with probability $2^{-5.9}$ on average when we take into account all possible input differences, i.e. $(\delta, \delta', \delta'', 0, \ldots, 0)$ where $\delta$, $\delta'$, $\delta'' \in \{1, 4, c, d\}$. Hence, obtaining at least 2 right families only requires about $2^{9+5.9} = 2^{14.9}$ queries to the encryption oracle on average.

[4] Actually, a structure of size $2^{12}$ where the first three nibbles take all values contains 64 right families with probability about $2^{-5.9}$. If we reduce these to form the structures of $2^9$ plaintext/ciphertext encryptions we described, only some of these 64 families are still present, hence the presence of either 0 or several right families in a structure.

#### 4.2.2.2 Filtering Right Pairs

Full diffusion has been achieved by the 6-th round. Thus, we guess 16 bits of key material to be able to partially invert the last round on one column. A guess leads to the correct nibble having a zero difference in every pair of the family with probability $2^{-4 \cdot 4} = 2^{-16}$. We repeat this independently over each column and obtain either 64 bits of key material or none at all. Since there are either several right families or none at all in the structures we consider, we only return the key guesses which come from several families as well as the corresponding families.

This is a powerful filter: while we expect each family from the structure to yield about one 64-bit candidate, the probability to have a collision is very small[5].

### 4.2.3 Implementing the Differential Attacks Against 6 Rounds

Pseudo-code describing our attack on 6-round Prince is provided in Algorithm 4.2.

**Algorithm 4.2** SAT-based differential attack against 6-round Prince.
**Input:** 6-round Prince instance
**Output:** secret key $k_0||k_1$

```
while the key has not been retrieved do
    Query a structure $\mathcal{S} = \left((p^0, c^0), \ldots, (p^{2^{12}-1}, c^{2^{12}-1}\right)$
    $H \leftarrow$ empty hashtable of lists of families indexed by 64-bits integers
    for all families $\mathcal{F}$ in $\mathcal{S}$ do
        for all columns of the internal state do
            for all 16-bits key guesses $k_{16}$ do
                for all pairs in $\mathcal{F}$ do
                    Invert key addition for the column using $k_{16}$
                    Invert $S^{-1}$ for the column
                    Invert $M'$ for the column
                end for
                if the correct nibble has a zero difference in all pairs then store $k_{16}$
            end for
        end for
        Combine all guesses from each column into 64-bits guesses
        for all 64-bits guesses $k_{64}$ append $\mathcal{F}$ to $H[k_{64}]$
    end for
    for all $k_{64}$ among the keys of $H$ do
        if $H[k_{64}]$ contains strictly more than 1 element then
            for all families $\mathcal{F}$ in $H[k_{64}]$ do
                Generate CNF $A$ encoding all encryptions in $\mathcal{F}$ with key s.t. $k_1 + L(k_0) = k_{64}$.
                for all trails $\mathcal{T}$in $\{\mathcal{T}_1, \mathcal{T}_2\}$ do
                    $B \leftarrow DoD(A, \mathcal{T})$
                    if $B$ is satisfiable then retrieve $k_0||k_1$ from the solution of $B$ and return it
                end for
            end for
        end if
    end for
end while
```

We ran this attack 10 times and found that about $2^{5.75}$ structures were needed on average. The filtering step is the most time consuming: finding a right pair requires about 1h 30min but the SAT-solver requires about 0.5s to recover the full key or

[5] Each structure yields $2^{9-3} = 2^6$ families for each of the $4^3$ interesting input differences so that we consider the families by groups of $2^{12}$. This implies that a collision has a probability of about $\binom{2^{12}}{2} \cdot 2^{-64} \approx 2^{-41}$.

(rarely) to discard the pair. For this reason, we approximate the complexity of this attack by the complexity of its filtering step. We query $2^{5.9}$ structures of $2^9$ encryptions and, for each, encryption, we invert the last round by guessing $2^{16}$ bits of key material for each of the $2^2$ columns. Hence, this attack requires about $2^{5.9+9+16+2} = 2^{32.9}$ partial decryptions and $2^{14.9}$ chosen/plaintexts. Memory complexity is dominated by the SAT-solver but is (well) below 1 Go, i.e. (well) below $2^{27}$ 64-bits blocks.

## 4.3 Structural Analysis of PRINCE

The $\alpha$-reflection introduced along with PRINCE [BCG+12] is the name given to the following property of a block cipher $E_k$: $E_k^{-1} = E_{k\oplus\alpha}$. In other words there is a constant $\alpha$ such that decryption for a key $k$ is the same operation as encryption under key $k \oplus \alpha$. PRINCE-core implements this property by having a three-parts structure as described here:

$$\mathrm{EC}_{k_1} = F^{-1}_{k_1\oplus\alpha} \circ I \circ F_{k_1},$$

where $F_k$ corresponds to 5 rounds of a classical Substitution-Permutation Network construction and where $I$ is an involution. This structure was later borrowed by both Qarma [Ava17] and Mantis [BJK+16].

Since we are going to study the structure of the cycles of different functions in a fashion similar to the way Biryukov analyzed the inner-rounds of some involutional ciphers in [Bir03], we borrow the notion *cycle type* of a permutation from this paper.

**Definition 4.3.1** (Cycle Type)**.** *The* cycle type *of a permutation $\pi$ is an (ordered) multiset containing the cycle lengths of the permutation. The cycle type of $\pi$ is denoted by $C(\pi)$.*

In what follows, we do not represent the round constants for the sake of simplicity. However, not only do our results hold in their presence but we could actually generalize them to any key schedule preserving the fact that the subkeys of symmetric rounds have an XOR equal to $\alpha$.

### 4.3.1 Small Cycles in Round-Reduced PRINCE

The central involution is $I = S^{-1} \circ M' \circ S$. Therefore, it is isomorphic to $M'$, a linear involution operating on each column of the internal state independently. It is easy to check experimentally the result given in [SBY+15] stating that $M'$ has exactly $2^{32}$ fixed points, meaning that $I$ also has $2^{32}$ fixed points. Therefore, $I$ has $2^{32}$ cycles of length 1 and $2^{63} - 2^{31}$ cycles of length 2.

The cycle type of $I^\alpha : x \mapsto I(x) \oplus \alpha$ is more sophisticated but still contains a fair amount of small cycles. As both $I$ and $x \mapsto x \oplus \alpha$ operate on each column of the internal space independently, we denote $I_i^\alpha$ the restriction of $x \mapsto I(x) \oplus \alpha$ to column $i$ and $I_i$ that of $I$. Since each of the functions $I_i^\alpha$ operates only on a space of size $2^{16}$, it is easy to generate their complete cycle structures independently by searching the whole space. Each $I_i^\alpha$ has a cycle type made of many "small" cycles, the largest having a length of 2844. This is explained by the fact that both $I$ and $x \mapsto x \oplus \alpha$ are involutions and each column of $I$ has exactly $2^8$ fixed points. Thus, most of the cycles have a particular structure[6] described in [MS87b] which we recall in Figure 4.4. We remark that to each cycle of $I_i^\alpha$ correspond two fixed points of $I_i$.

[6] While there are some cycles which do not have this structure, they form a negligible minority: for $f_0$, 256 elements out of 65536 are on such cycles, 64 for $f_1$, 8 for $f_2$ and 194 for $f_3$.

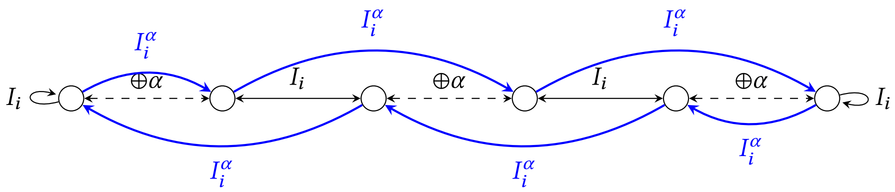


Figure 4.4: The structure of a cycle of $I_i^\alpha$ for $i \in [0,3]$.

After generating the cycle type for each $I_i^\alpha$, we combine them to obtain the cycle type of $x \mapsto I(x) \oplus \alpha$ using Algorithm 4.3. The cycle type of this function is too complex to be printed completely but some information extracted from it is given in Table 4.3. If we pick $x$ uniformly at random, the expected length of the cycle it is on is $2^{30.7}$.

**Algorithm 4.3** Generating the cycle type of $I^\alpha$ from those of its columns.

```
for $i \in [0,3]$ do
    $C_i \leftarrow$ List of the cycle length of $I_i^\alpha$
end for
$C \leftarrow$ Hashtable indexed by integers
for $(\ell_0,\ell_1,\ell_2,\ell_3) \in C_0 \times C_1 \times C_2 \times C_3$ do
    $\ell \leftarrow \mathrm{lcm}(\ell_0,\ell_1,\ell_2,\ell_3)$
    $C[\ell] \leftarrow C[\ell] + \ell^{-1} \cdot \prod_{i=0}^{3} \ell_i$
end for
return $C$
```

| Cycle Length $\ell$ | #{cycles of length $\ell$} | Pr[$\ell(x) = \ell, x$ drawn uniformly] |
|---|---|---|
| 1 | 0 | 0 |
| 2 | $2^7$ | $2^{-57}$ |
| 4 | $2^{10.25}$ | $2^{-53.75}$ |
| 8 | $2^{15.46}$ | $2^{-48.54}$ |
| 10080 | $2^{33.06}$ | $2^{-17.63}$ |
| 110880 | $2^{31.96}$ | $2^{-15.27}$ |
| $\leq 2^{10}$ | – | $2^{-22.4}$ |
| $\leq 2^{15}$ | – | $2^{-12.4}$ |
| $\leq 2^{24}$ | – | $2^{-4.1}$ |

Table 4.3: Information about the cycle type of $I^\alpha$, where $\ell(x)$ is the length of the cycle on which $x$ is.

Recall that $\mathrm{EC}_{k_1}^4$ is the permutation of $\{0,1\}^{64}$ corresponding to an encryption under key $k_1$ by PRINCE-core reduced to 4 rounds. Then $x \mapsto \mathrm{EC}_{k_1}^4(x) \oplus \alpha$ has the same cycle type as $I^\alpha$ due to the cancellation of the last round of one encryption with the first round of the next. Indeed, to each cycle of this function corresponds one cycle of $I^{\alpha}$, as illustrated in Figure 4.5 where a cycle $(x_0, x_1, x_2, x_3)$ of length 4 of $x \mapsto \mathrm{EC}^4_{k_1}$ is represented along with the corresponding cycle of $I^{\alpha}$ (dashed line).

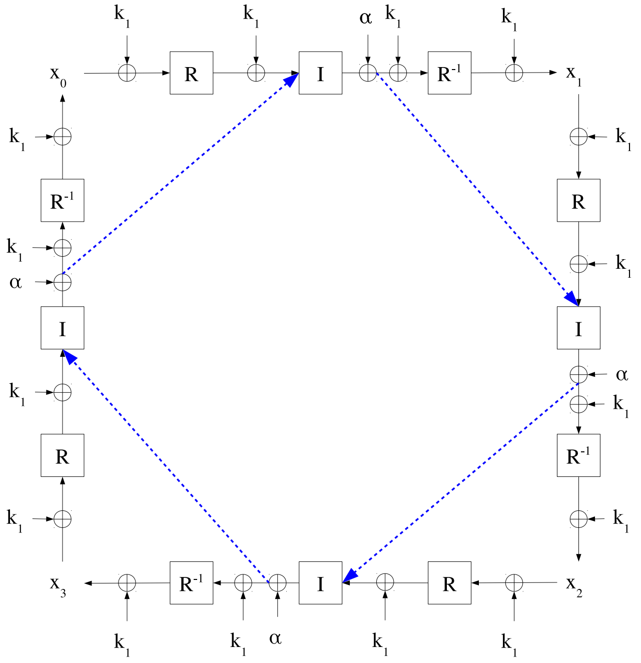


Figure 4.5: Correspondence between a cycle of $x \mapsto \mathrm{EC}^4_{k_1}(x) \oplus \alpha$ and a cycle of $I^{\alpha}$.

A first consequence of these observations is the existence of a distinguisher for 4-round Prince-core requiring about $2^{27.4}$ adaptatively chosen plaintexts. As stated in Table 4.2, an element picked at random is on a cycle of length at most $2^{15}$ with probability $2^{-12.4}$. Therefore, we repeat the following experiment multiple times:

1. pick an element $x$ uniformly at random,
2. check if it is on a cycle of length at most $2^{15}$ by iterating $x \mapsto \mathrm{EC}^4_{k_1}(x) \oplus \alpha$ at most $2^{15}$ times.

The experiment is a success if $x$ is on a cycle of length at most $2^{15}$. If the permutation is $\mathrm{EC}^4_{k_1}$ for some $k_1$, then its probability of success is $2^{12.4}$ but if the permutation is a random permutation[7], then the probability of success becomes $2^{-49}$. We confirmed experimentally the success probability of this experiment for $\mathrm{EC}^4_{k_1}$.

A second consequence is the existence of "small" sets of plaintext/ciphertext encryptions where the set of the ciphertexts is the image of the set of the encryptions

[7] Recall that the probability for $x$ to be on a cycle of length $\ell$ for a permutation of $[0, N-1]$ is equal to $1/N$. Hence, the probability that the length is smaller than $2^{15}$ for a permutation of $[0, 2^{64}-1]$ is $\sum_{\ell=1}^{2^{15}} 2^{-64} = 2^{-49}$.

by a function significantly simpler than a PRINCE encryption. This topic is studied in the next section.

### 4.3.2 Simplifications of the Representation of PRINCE

The particular cycle types of the round-reduced versions of PRINCE studied above lead to simpler alternative representations of the encryption algorithm.

#### 4.3.2.1 Consequences of the Cycle Type of $I$

Suppose that an encryption is such that the input of $I$ is one of the $2^{32}$ fixed-points of this function. Then the key addition before and after this function cancel each other so that only the addition of $\alpha$ remains. Then, since $M$ is linear, the operations $M^{-1} \circ (\oplus\alpha) \circ M$ become simply the addition of $M^{-1}(\alpha)$. Thus, the 4 center rounds — minus the first and last key addition — become a simple S-Box layer which we denote $S'$ and which is defined by

$$S'(x) = S^{-1}\Big(S(x) \oplus M^{-1}(\alpha)\Big).$$

This simplifying process is summarized in Figure 4.6. Note that if $M^{-1}(\alpha)$ has any nibble equal to 0 then the function $S'$ is the identity for this nibble. However, for the value of $\alpha$ chosen by the designers of PRINCE, there is no such nibble.

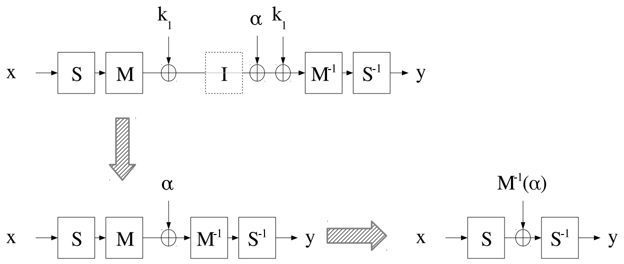


Figure 4.6: Simplification of the 4 center-rounds if the input of $I$ is a fixed point.

The simplification goes further. Indeed, since $S'$ operates only at the nibble level, it commutes with the operations $SR$ and $SR^{-1}$ (up to a reordering of the S-Boxes in $S'$). Therefore, if we add one round before and one round after $S'$, we can replace $SR^{-1} \circ S' \circ SR$ by $S''$ where $S''$ is another S-Box layer. Hence, 6-round PRINCE operates on each column of the internal state independently: each output bit depends only on 16 bits of the input, 28 bits[8] of $k_1$ and at most 18 bits of $k_0$. This simplification is summarized in Figure 4.7.

Similar simplifications occur if instead of having a fixed point we have a particular collision between two encryptions. This setting corresponds to the so-called *mirror slide attack* described by Dunkelman *et al.*in [DKS12]. Consider two encryp-

[8] In each column, 16 bits from the corresponding column of $k_1$ are used as well as 16 bits from the corresponding column of $SR^{-1}(k_1)$. Since the top nibble of these two sets is the same, we are left with $32 - 4 = 28$ bits.

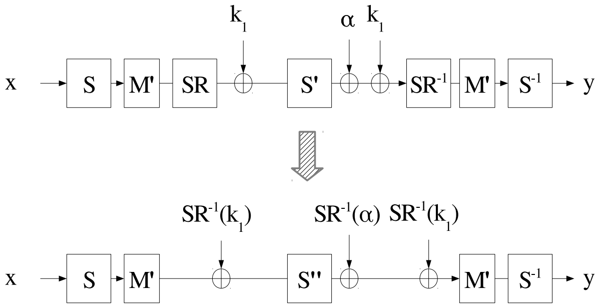


Figure 4.7: Simplification of the 6 center-rounds if the input of $I$ is a fixed point.

tions $(p^0, c^0)$ and $(p^1, c^1)$ by Prince-core as follows

$$\begin{aligned} c^0 = \mathrm{EC}_{k_1}(p^0) &= \left(F^{-1}_{k_1 \oplus \alpha} \circ I \circ F_{k_1}\right)(p^0) \\ c^1 = \mathrm{EC}_{k_1}(p^1) &= \left(F^{-1}_{k_1 \oplus \alpha} \circ I \circ F_{k_1}\right)(p^1) \end{aligned}$$

which are such that $F_{k_1}(p^0) = I\left(F_{k_1}(p^1)\right)$. In this case, we have that

$$\begin{aligned} c^0 &= \left(F^{-1}_{k_1 \oplus \alpha} \circ F_{k_1}\right)(p^1) \\ c^1 &= \left(F^{-1}_{k_1 \oplus \alpha} \circ F_{k_1}\right)(p^0), \end{aligned}$$

where 6 rounds of $F^{-1}_{k_1 \oplus \alpha} \circ F_{k_1}$ can be simplified exactly as described and therefore only operate on each column separately.

In conclusion, if an encryption is such that the input of $I$ is a fixed-point of this function or if two encryptions form a mirror slide pair, then 4 rounds of Prince consist simply in 16 parallel operations on each nibble and 6 rounds of Prince in 4 parallel operations on each column.

#### 4.3.2.2 Consequences of the Cycle type of $I^\alpha$

Consider a sequence of plaintexts $(p^0, \dots, p^{\ell-1})$ and their corresponding ciphertexts $(c^0, ..., c^{\ell-1})$ such that the input $x_5^i \oplus k_1$ of the sixth round for the plaintext $p^i$ is the image of $x_5^{i-1} \oplus k_1$ by $I^\alpha$. We call such a sequence a *cycle set* and we give a representation of such a sequence on Figure 4.8: if two values are equal then they are connected by a line; red lines correspond to the cycle of $I^\alpha$ this set is built out of and blue lines correspond to the propagation of these equalities through identical operations, namely $x \mapsto k_1 \oplus R^{-1}(x \oplus k_1)$.

There is a unique function mapping $p^i$ to $c^{i-1}$ in every cycle set which corresponds to the encryption algorithm where the 4 center-rounds have been removed and replaced by a simple addition of $\alpha$. This means that this function undergoes the simplifications described above except that these cover 2 more rounds. In particular, for 6-round Prince-core, the function mapping $p^i$ to $c^{i-1}$ only operates at the nibble level and, for 8-round Prince-core, it operates at the column level. At least 10 rounds are necessary to obtain full diffusion. Prince has 12 rounds in total.

The cycle sets we consider cover the 4 center-rounds of Prince but it is possible to generalize this construction to an arbitrary number of rounds. However, the cycle

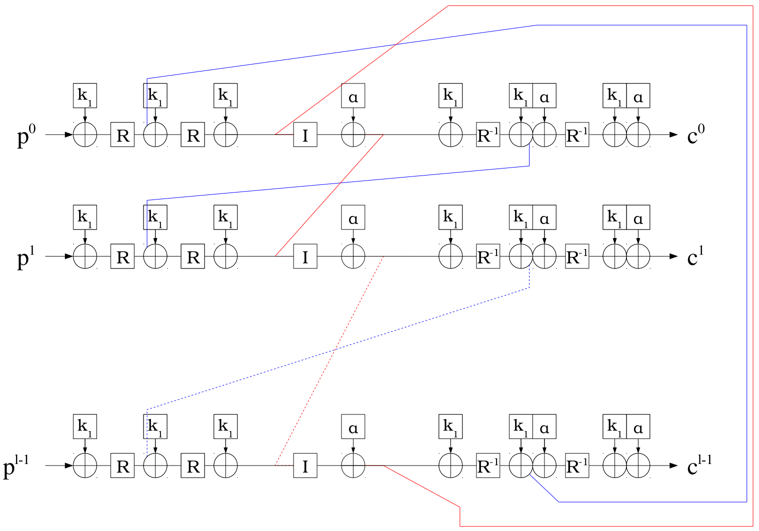


Figure 4.8: A cycle set of 6-round PRINCE-core.

set sizes are abnormally small in this case because of the cycle type of $I^{\alpha}$. Indeed, a random plaintext/ciphertext pair is in a cycle set of size $2^{30.7}$ and in a cycle set of size smaller than $2^{15}$ with probability $2^{-12.4}$. In other cases, including *a priori* if we have a cycle covering at least 6 rounds, the expected size of a cycle set is the expected size of the cycle of a random permutation a random element is on, namely $2^{63}$ for a 64-bit permutation.

Should the cycle sets of PRINCE become identifiable, the security of up to 8 rounds may be compromised as the alternative versions of the cipher we described in this section are much weaker than the original cipher. Furthermore, since small cycles are not unlikely to be found, the data complexity of such an attack may remain feasible.

Chapter 5

# Truncated Differentials in Twine

The Generalized Feistel Network (GFN), introduced by Nyberg in [Nyb96], is a modification of the regular Feistel Network which uses more than 2 branches. Having more branches allows the use of a simpler Feistel function, the branch permutation taking care of the diffusion. However, the simple branch rotation used in most GFN with $b$ branches requires $b$ rounds to obtain full diffusion. To improve this number, more sophisticated permutations were introduced in [SM10] and one such permutation has been used by the authors of Twine [SMMK13], a lightweight block cipher with a GFN structure: while Twine uses 16 branches, only 8 rounds are necessary for full diffusion.

Twine is therefore a good example of common trade-offs in lightweight cryptography as it has a simple round function iterated many times, while also being one of the only instances of a GFN with improved diffusion layer.

A similar block cipher is LBlock [WZ11], a lightweight block cipher which served as the basis for the design of LBlock-s, a variant with a different S-Box and key schedule used in the Lightweight Authenticated Cipher (Lac) submitted to the CAESAR competition by a related team [ZWW+14]. While LBlock is described as a “regular” two-branched Feistel Network, the rotation used in its permutation layer and the simplicity of its Feistel function make it equivalent to a GFN similar to Twine. The designers of Twine pointed out this resemblance in [SMMK13].

The particular permutation layer of Twine implies a vulnerability of its round function against truncated differential cryptanalysis [Knu95]. Unlike “normal” differential cryptanalysis, this technique does not rely on studying fully specified trails where each bit of difference is supposed to have a particular value but instead on looking at more general patterns where some bit differences may take both values 0 and 1. In the case of word oriented cipher, we can restrict the investigation to trails where the differences are studied at the word level: either there is at least one difference over the whole word or there is none. Trails where some of the bits are not specified are often used when adding rounds on the top and the bottom of a differential distinguisher. However, using a truncated differential covering all the rounds can also yield powerful attacks. For example, such an approach has been used recently by Lallemand *et al.* [LN15] to attack the lightweight block cipher KLEIN [GNL11]. Truncated differentials have also been used to enhance the search for high probability differentials. Two recent examples are the attack on the block cipher Prince [CFG+15] covering the highest number of rounds and a differential forgery attack on the authenticated cipher Lac [Leu16].

As we introduce new attacks on Twine, we summarize the complexities of the best attacks against this cipher in the single-key model in Table 5.1.

| Reference | Type | Version | Data | Time | Memory |
|---|---|---|---|---|---|
| [ÇKB12] | Biclique | full Twine-80 | $2^{60}$ | $2^{79.1}$ | $2^{8}$ |
| | | full Twine-128 | $2^{60}$ | $2^{126.82}$ | $2^{8}$ |
| [ZJ14] | Impossible diff. | 23-round Twine-80 | $2^{57.85}$ | $2^{79.09}$ | $2^{78.04}$ |
| | | 24-rounds Twine-128 | $2^{58.1}$ | $2^{126.78}$ | $2^{125.61}$ |
| [BDP15] | Meet-in-the-Middle | 25-round Twine-128 | $2^{48}$ | $2^{124.7}$ | $2^{109}$ |
| [BDP15] | Impossible diff. | 25-round Twine-128 | $2^{59.6}$ | $2^{125.8}$ | $2^{78.6}$ |
| Section 5.3.3 | Truncated diff. | 23-round Twine-128 | $2^{58}$ | $2^{126.78}$ | $2^{89}$ |
| | | | $2^{62}$ | $2^{125.94}$ | |
| | | | $2^{64}$ | $2^{124.35}$ | |

Table 5.1: The best attacks on Twine in the single-key model and their complexity.

In Section 5.2, we highlight a property of the permutation used in Twine. Rounds of encryption can be grouped into blocks of 4 rounds in such a way that two halves of the internal states evolve independently from one another during the first 3 rounds and then exchange information only during the fourth. This property is iterative. We also discuss why LBlock and its simpler variant LBlock-s exhibit the same 4-round behavior. As a consequence of this observation, we describe several high probability truncated differential trails for all these ciphers. We then leverage them in Section 5.3 to attack 23-rounds of Twine-128 using comparatively low memory. Finally, we use these truncated trails to optimize a search for high probability differentials and show that the conservative choice of S-Box made by the designers of Twine greatly limits the differential effect in this primitive — unlike in LBlock-s for instance.

## 5.1 Descriptions of Twine, LBlock and LBlock-s

### 5.1.1 Description of Twine

This block cipher uses 16 branches of 4-bits and has a very simple round function (see Figure 5.1): the Feistel function consists of a xor of a sub-key and a call to a unique S-Box based on the inverse function in $GF(2^4)$ (see Section 8.3.1.1 (p. 147)). Then, the branches are shuffled using a sophisticated nibble permutation ensuring faster diffusion than a simple branch rotation [SM10]. One version of Twine uses an 80 bits key, another uses a 128 bits key and we denote these versions Twine-80 and Twine-128. They only differ by their key-schedule and both have 36 rounds. Both key schedules have a sparse gfn structure which uses only 2 and 3 S-Box calls per round for Twine-80 and Twine-128 respectively. The gfn used by the key-schedule of Twine-128 is depicted on Figure 5.2. We refer the reader to [SMMK13] for the 80-bit version. At each round, some fixed nibbles of the key-state are used as round keys for the block cipher. One round of Twine is depicted on Figure 5.1.

Given a collection of messages $\{P^0, \ldots\}$, the nibble with index $i$ taken at round $r$ of message $m$ is denoted $x_r^m[i]$. The master key is denoted $K$ while the round key

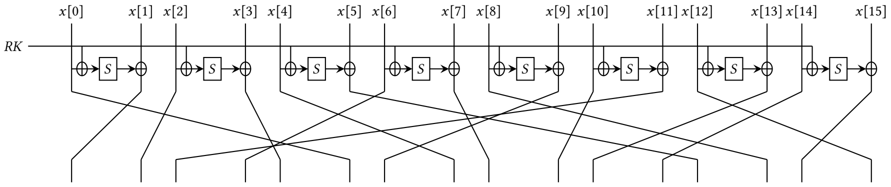


Figure 5.1: The round function of TWINE.

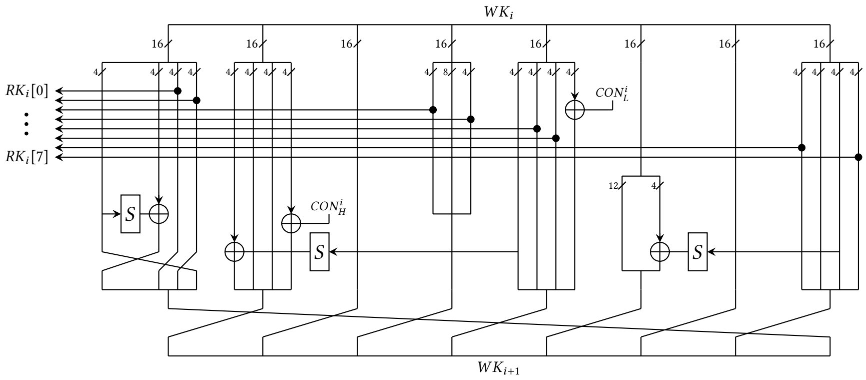


Figure 5.2: The key-schedule of TWINE-128.

used at round $r$ is denoted $RK_r$.

### 5.1.2 Descriptions of LBlock and LBlock-s

LBlock [WZ11] is a two-branched Feistel Network. The Feistel function of consists of a key addition, an S-Box layer $S$ made of 8 different 4-bit S-Boxes and a nibble permutation $P$. In addition to the usual Feistel structure, there is a rotation by 8 bits to the left on the right branch before the xor. This operation on the right branch leads to a strong structural similarity with Twine, as the authors of this cipher acknowledged. The complete round function is described in Figure 5.3. In total, 32 rounds are used to encrypt a block.

Unlike Twine, LBlock only uses 80-bit keys. Its key-schedule is similar to that of Present [BKL+07]: it relies on a rotation of the 80-bits register used to store the master key and on the application of two S-Boxes.

LBlock-s, the block cipher used in the authenticated cipher Lac [ZWW+14], is identical to LBlock except that the S-Box layer uses a unique S-Box instead of 8 different ones and that its key-schedule is closer to the one of Twine-80. The S-Boxes of LBlock and that of LBlock-s all have similar differential and linear properties.

## 5.2 The 4-Round Structure of Twine, LBlock and LBlock-s

The round functions of Twine can be described using an equivalent representation which allows a clearer representation of some differential path. This alternative representation is given in Figure 5.4a. A similar representation of LBlock(-s) can be obtained, as shown in Figure 5.4b. This observation highlights the similarities between these two designs.

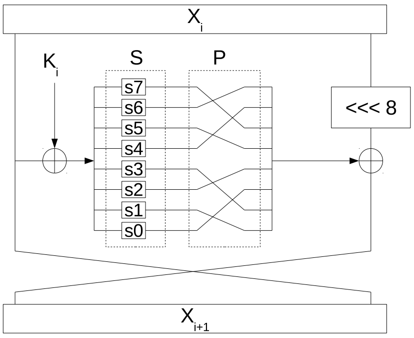
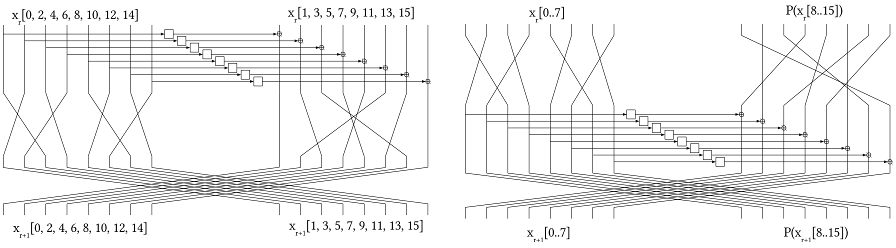


Figure 5.3: The round function of LBlock


x_r[0, 2, 4, 6, 8, 10, 12, 14]
x_r[1, 3, 5, 7, 9, 11, 13, 15]
x_{r+1}[0, 2, 4, 6, 8, 10, 12, 14]
x_{r+1}[1, 3, 5, 7, 9, 11, 13, 15]
x_r[0..7]
P(x_r[8..15])
x_{r+1}[0..7]
P(x_{r+1}[8..15])

(a) Twine

(b) LBlock(-s)

Figure 5.4: Round functions alternative representations

For Twine, we simply move all the branches going into Feistel functions to the left and those receiving its output to the right. This means we simply move branches with even indices on the left and those with odd ones on the right, as described in Figure 5.4a.

The process leading to the alternative description of LBlock(-s) is more complicated than for Twine and is summarized in Figure 5.5. The S-Boxes and the permutation layer $P$ both operate on 4-bit nibbles, which implies that $P \circ S$ is equivalent to $S' \circ P$ where $S'$ is a reordered S-Box layer. Then, instead of applying $P$ within the Feistel function, we apply it before entry and then apply the functional inverse $1/P$ of $P$ on the same branch to compensate. Finally, we note that the rotation $R$ and the inverse permutation $1/P$ are applied on the same data, so we combine them into one operation $R \circ (1/P)$. Finally, we replace the two 32-bit words of the internal state of LBlock by eight 4-bit nibbles each and thus obtain the representation given in Figure 5.4b.

We represent 4 rounds of Twine using our alternative representation on the left of Figure 5.6. In this picture, S-Boxes are not shown and xors are represented by circles. The basic representation is on the left and another representation which highlights the two components is on the right. The numbers correspond to nibble indices in the "regular" representation of Twine, i.e. as in Figure 5.1. As we can see, the 16 branches can be grouped into two disjoint *components*, red and blue, such that branches from one component interact only with each other during 3 rounds

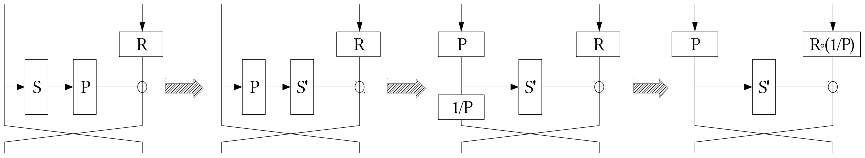


Figure 5.5: How to obtain the alternative representation of LBlock(-s).

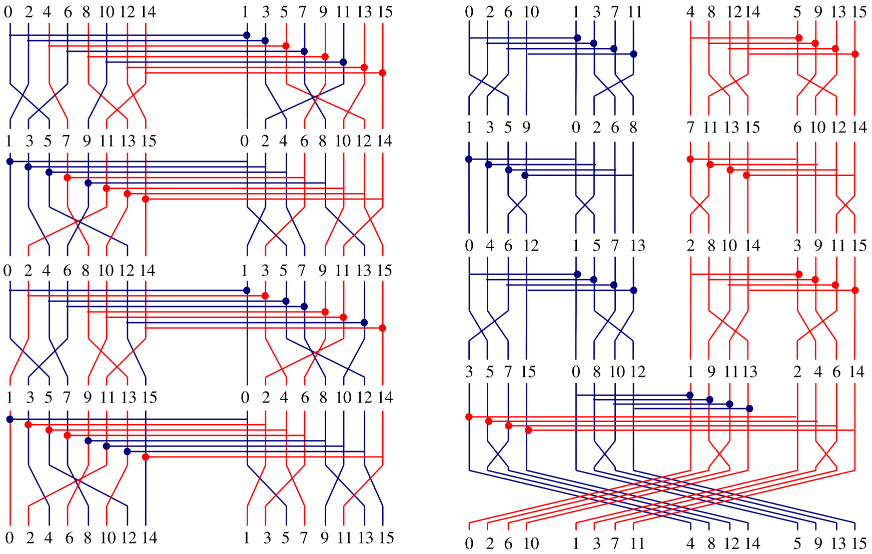


Figure 5.6: Alternative representations of 4 rounds of Twine.

out of 4. However, during the last round, branches from each component interact only with branches from the other component. Furthermore, these components are stable in the sense that such groups of 4 iterations can be plugged together to cover any number of rounds and remain separated for all rounds with index $r$ where $r \not\equiv 3$ mod 4. Indeed, in Figure 5.6, the branches which are blue at the output of the fourth round are exactly those which are red at the input of the first round. If we draw these components separated from one another, we obtain another description of 4 rounds of Twine given on the right of Figure 5.6. The same can be done with LBlock(-s), see Figure 5.7.

## 5.3 Truncated Differential Cryptanalysis of Twine

Because of the particular structure it has over 4 rounds, Twine exhibits some truncated differential patterns with high probability. These are described in Section 5.3.1. Then, they are used in a key-recovery attack targeting 23-round Twine-80 in Section 5.3.2.

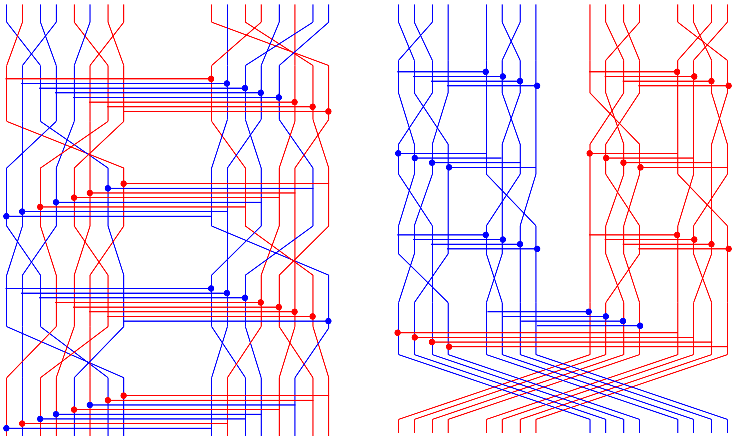
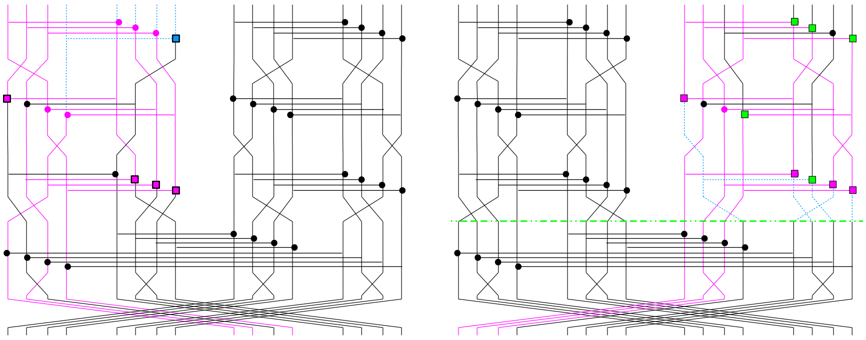


Figure 5.7: Alternative representation of 4 rounds of LBlock(-s).

### 5.3.1 Truncated Differentials over 4 Rounds

The simplest truncated differential pattern deduced from the 4-round behavior of Twine implies 4 active branches in the input and 4 active branches in the output of 4 rounds at the cost of 4 difference cancellations during round 3. More precisely, let $(x[0], x[2], x[6], x[10])$ have non-zero differences. Then these differences will propagate to the full blue component during the next two rounds. During round 3, if the differences in $(x[0], x[4], x[6], x[12])$ cancel themselves with the differences in $(x[1], x[5], x[7], x[13])$ after going through the key addition and the S-Box layer, then the differences do not propagate to the red component. Hence, the differences remain contained in the blue component for another 3 round with probability 1. Since 4 cancellations happen with probability $2^{-16}$ and since such truncated characteristics can be "plugged" so as to cover as many rounds as we want, we have a truncated differential covering $4r$ rounds with probability $2^{-16 \cdot r}$.

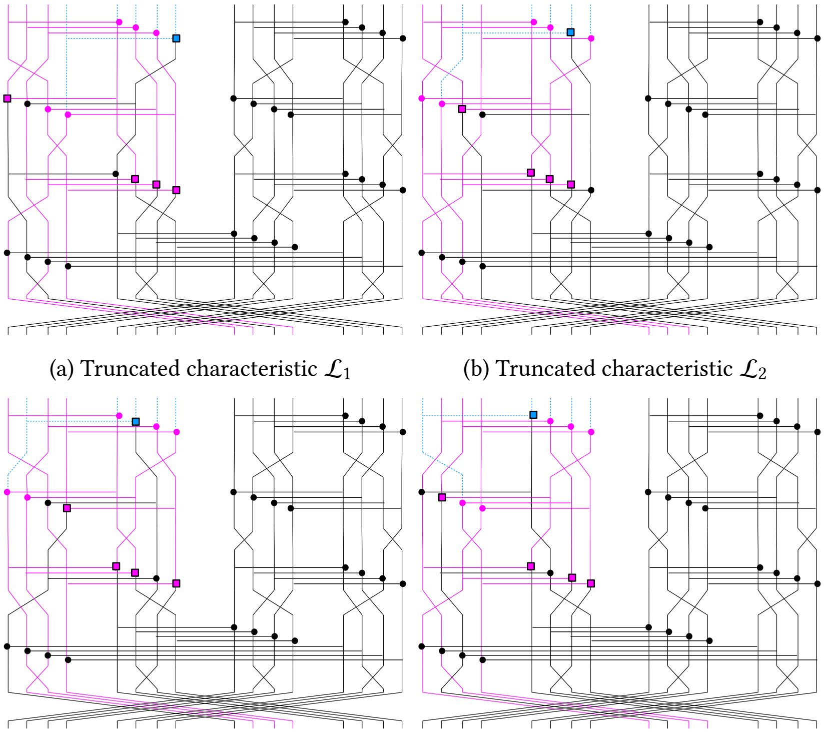


(a) $\mathcal{L}_1$ (pink) and its counterpart $\mathcal{L}'_1$

(b) $\mathcal{R}_1$ (pink) and its counterpart $\mathcal{R}'_1$

Figure 5.8: 4-rounds truncated differentials for Twine and their modified versions.

Other slightly different characteristics involve three active branches in the input and the output after 4 rounds in such a way that only 4 cancellations are necessary, meaning that they also have a probability of $2^{-16}$. Two of them are described in



(a) Truncated characteristic $\mathcal{L}_1$

(b) Truncated characteristic $\mathcal{L}_2$

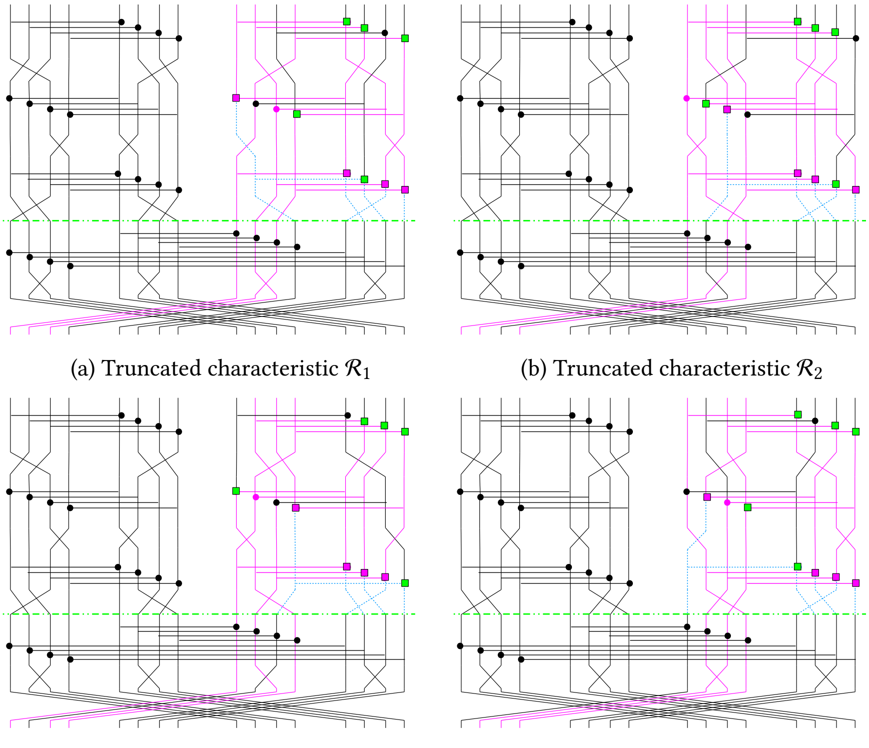
(c) Truncated characteristic $\mathcal{L}_3$

(d) Truncated characteristic $\mathcal{L}_4$

Figure 5.9: Truncated differential characteristics on the left component of Twine and their extensions towards the top.

Figure 5.8a and the others are in Figure 5.9, where non-zero differences are purple and zero differences are black. They all work by having one cancellation during the second round and three during the third. As before, the first and fourth rounds have probability 1. However, we can extend them for the first 4 rounds by adding non-zero differences over all the components — which is represented by a blue dotted line in Figure 5.9. At the cost of one more cancellation, hence a probability of $2^{-20}$, we can use structures made of $2^{32}$ plaintext/ciphertext couples giving raise to $\binom{2^{32}}{2} \approx 2^{63}$ pairs following the first 4-round trail with probability $2^{-20}$, meaning that about $2^{43}$ will be right pairs for this trail. Without extending the trail, we would get $\binom{2^{24}}{2} \times 2^{-16} \approx 2^{41}$ right pairs.

As we can see, such differential trails move on to the right component after 4 rounds. There are similar trails covering it described in Figure 5.10 and 5.8b. As before, black represents zero differences, purple non-zero ones and purple squares the cancellations which must occur during encryption. These figures also represent, in dotted blue, the difference propagation during the first 3 rounds without any constraints regarding the cancellations so that this trail has probability 1. The green squares represent the cancellations which must be observed when starting from the bottom and partially decrypting a pair of ciphertext having the correct output difference.



(a) Truncated characteristic $\mathcal{R}_1$

(b) Truncated characteristic $\mathcal{R}_2$

(c) Truncated characteristic $\mathcal{R}_3$

(d) Truncated characteristic $\mathcal{R}_4$

Figure 5.10: Truncated differential characteristics on the right component of Twine and their extensions towards the bottom.

It is therefore possible to cover as many rounds as we want using a characteristic $\mathcal{L}_i, \mathcal{R}_i, ..., \mathcal{L}_i, \mathcal{R}_i$ for any $i \in [1,4]$. Such a trail would cover $4r$ rounds with probability $2^{-16 \cdot r}$. We also denote $\mathcal{L}'_i$ the trail $\mathcal{L}_i$ extended on top so as to have 8 non-zero input differences at the cost of one additional cancellation and $\mathcal{R}'_i$ the trail $\mathcal{R}_i$ reduced to 3 rounds and where no cancellations occur. Both $\mathcal{L}'_i$ and $\mathcal{R}'_i$ correspond to the case where the dotted blue lines contain non-zero differences.

### 5.3.2 Efficient Key Recovery

The 5 cancellations preventing the difference from spreading to the other component can be grouped into 2 sets, one of 2 cancellations and one of 3, each depending on a distinct set of 5 and 6 sub-keys. This phenomenon is illustrated in Figure 5.11, where dotted lines correspond to zero differences and squares to cancellations.

Starting from a pair of plaintexts separated by the correct input difference, it is easy to generate the set of all the sub-keys combinations which would lead to the trail we expect as follows:

1. Try all possible combinations of the sub-keys involved in the blue part of Figure 5.11 and store only those leading to the correct cancellations. There are $2^{4 \times 5} = 2^{20}$ possibilities, out of which $2^{20-3 \times 4} = 2^8$ lead to the correct pattern.

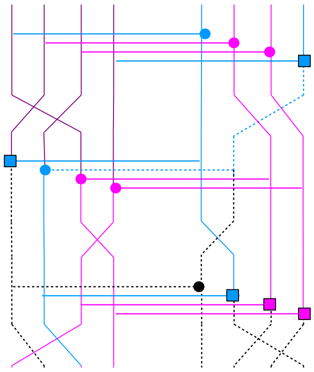


Figure 5.11: Data-paths involved in the 5 cancellations happening in $\mathcal{L}_1'$.

2. Try all possible combinations of the sub-keys involved in the pink part of Figure 5.11 and store only those leading to the correct cancellations. There are $2^{4\times 6} = 2^{24}$ possibilities, out of which $2^{20-2\times 4} = 2^{16}$ lead to the correct pattern.
3. Combine the $2^8$ and $2^{16}$ independent sub-candidates to obtain $2^{24}$ candidates of $4 \times (5+6) = 44$ bits each.

A very similar algorithm can be used to recover the candidates yielding the correct cancellations when partially decrypting the ciphertexts of the same pair. Doing so generates another $2^{24}$ candidates of 44 bits each.

### 5.3.3 Combining Truncated Differentials to Attack 23-Rounds Twine-128

The high level idea of this attack is to discard some combinations of values for the set made of the 12 sub-keys used to update the left component during the first 3 rounds and the 12 sub-keys used to update the right component during the last 3 rounds. These form of set of 24 nibbles, i.e. 96 bits. The first and last 4-rounds blocks of the truncated differential trails described in Figure 5.12 all depend on the sub-keys in this set, although each of the trails only uses a different set of 88 bits out of the 96 bits available. It is therefore easy to combine the information deduced from each. A complete description of our attack is provided in Section 5.3.3.1 and its complexity is estimated in Section 5.3.3.2.

#### 5.3.3.1 Details of the Attack

Using the trails described in the previous Section, we can cover 23 rounds with probability $p = 2^{-84}$ in four different ways. The combination of these different 4-rounds characteristics is described in Figure 5.12, where 0 means there is no difference on this nibble and $x$ means there is some non-zero difference. The nibbles are ordered as in the right side of Figure 5.6. They all require the same input truncated difference and are mutually exclusive.

1. **Data generation.** First of all, we need to generate the pairs from which we are going to extract information about the sub-keys. For this purpose, we use $2^s$ structures of $2^{32}$ plaintext/ciphertext couples each. In these structures, nibbles $x_0[0..3, 6, 7, 10, 11]$ take all possible values while the others are constant. We

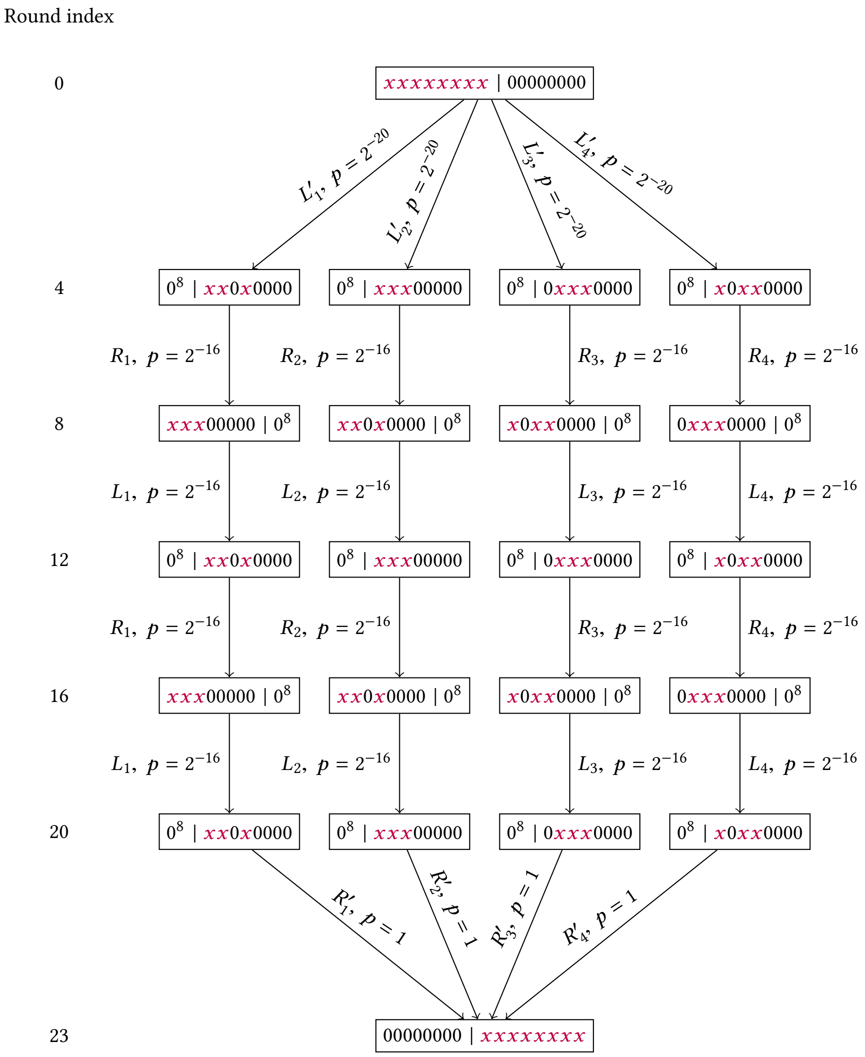


Figure 5.12: The four distinct 23-rounds truncated differential trails we use to attack Twine.

thus obtain about $2^{s+63}$ pairs with the correct input difference at a cost of $D = 2^{32+s}$ queries to an encryption oracle. We then obtain all the pairs which also have the correct output difference, namely[1] $0^8x^8$, at the cost of $2^s$ sorting of arrays of $2^{32}$ ciphertexts. Since this output difference has probability $f = 2^{-32}$, this leaves $N_p = 2^{s+63} \cdot f = 2^{s+31}$ pairs with the correct input and output differences. Among these, there are $N_r = 2^{s+63} \cdot p = 2^{s-21}$ right pairs for each of the 4 truncated differential trails described in Figure 5.12 — which means that $s$ must be at least equal to 21. Note that $N_p = N_r f/p$ and $D = N_r/p$.

Now that we have the data we need, we process it as follows for each of the 4

[1] The order of the nibbles in this difference corresponds to the order of the nibbles in our alternative representation.

trails, $t$ being the index of the trail considered.

2. **Counters increment.** For $t \in [1,4]$:

   a) Let $\mathcal{T}_t$ be an array of size $2^{88}$. For each of the $N_p$ pairs which passed the filter, we run the algorithms described in Section 5.3.2 to recover $2^{24}$ sub-candidates for the subset of 11 sub-keys used in the first 3 rounds and $2^{24}$ sub-candidates for the other subset of 11 sub-keys used in the last 3 rounds. This leads to $K = 2^{48}$ candidates living in a space of size $\mathcal{S} = 2^{88}$

3. **Discarding candidates.** We now have 4 tables $\mathcal{T}_t, t \in [1,4]$ of $\mathcal{S}$ counters. In each table, each of the $\mathcal{S}$ candidates has been incremented $N_p$ times with probability $K/\mathcal{S} = 2^{-40}$. We thus approximate the distribution of the counters by a normal distribution with average value $\mu_{\text{wrong}} = N_p K/\mathcal{S} = N_r(fK)/(p\mathcal{S})$ and variance $\sigma^2_{\text{wrong}} = N_p(K/\mathcal{S})(1-K/\mathcal{S}) \approx N_r(fK)/(p\mathcal{S})$. However, the correct counter has also been incremented by each of the $N_r$ correct pairs, meaning that its average value is $\mu_{\text{right}} = N_p K/\mathcal{S} + N_r = N_r\big((fK)/(p\mathcal{S}) + 1\big)$. We define $\mu_0$ in order to express $\mu_{\text{wrong}}$, $\sigma^2_{\text{wrong}}$ and $\mu_{\text{right}}$ easily:

$$\mu_0 = \frac{f \cdot K}{p \cdot \mathcal{S}},\ \mu_{\text{wrong}} = N_r\mu_0,\ \sigma^2_{\text{wrong}} = N_r\mu_0,\ \mu_{\text{right}} = N_r(\mu_0 + 1).$$

   We then combine the information from these counters. To achieve this, we recall that the indices in the tables $\mathcal{T}_t$ correspond to different subsets of 88 bits of a set of sub-keys of 96 bits in total. Therefore, we can associate a single representative in each table $\mathcal{T}_t$ to each candidate of 96 bits. Hence, we can give a score to each 96-bits candidate by taking the average of the scores of their representatives in each table. As a consequence, the score of a wrong candidate follows a normal distribution with the following parameters:

$$\mathcal{N}\Big(\frac{4 \cdot \mu_{\text{wrong}}}{4}, \frac{4 \cdot \sigma^2_{\text{wrong}}}{4^2}\Big) = \mathcal{N}\Big(\mu_{\text{wrong}}, \frac{\sigma^2_{\text{wrong}}}{4}\Big).$$

   Similarly, the score of the right candidate is a sample from a normal distribution $\mathcal{N}\big(\mu_{\text{right}}, \sigma^2_{\text{right}}/4\big)$. If we want a probability of keeping the right candidate of about 1/2, we need to discard all the candidates having a score below $\mu_{\text{right}}$. We denote Pr [wrong] the probability to keep a wrong candidate, i.e. the probability that a wrong candidate has a score greater than $\mu_{\text{right}}$. It is given by:

$$\Pr[\text{wrong}] = \frac{1}{2}\left[1 - \text{erf}\left(\frac{\mu_{\text{right}} - \mu_{wrong}}{\sqrt{2\sigma^2_{\text{wrong}}/4}}\right)\right] = \frac{1}{2}\left[1 - \text{erf}\left(\sqrt{\frac{2N_r}{\mu_0}}\right)\right]. \quad (5.1)$$

Unsurprisingly, the number of wrong candidates discarded increases with the number $N_r$ of right pairs for each trail. Table 5.2 gives the value of the probability Pr [wrong] of keeping a wrong candidate depending on the value of $N_r$ as well as the corresponding data complexity knowing that $\mu_0 = 2^{-32+56+84-96} = 2^{12}$. Note also that the maximum value of $N_r$ corresponds to the full code-book, i.e. when we query all $2^{32}$ possible structures, in which case, $N_r = 2^{32-21} = 2^{11}$.

#### 5.3.3.2 Complexity Estimation

The memory complexity of the truncated differential attack described in the previous section is straightforward to evaluate. We need to store at most $2^{63}$ plaintext/ciphertext pairs and 4 times $2^{88}$ counters. These counters are on average equal to $N_r \cdot 2^{12}$ with $N_r$ equal to at most $2^{11}$. Hence, 32 bits are more than enough for each of them. Storing the counters is clearly the dominating factor here, meaning that the memory complexity of this attack is $4 \cdot 2^{88} = 2^{90}$ counters of 32 bits or $2^{89}$ internal states.

We need $N_r \cdot 2^{53}$ plaintext/ciphertext pairs, meaning that the data complexity is $N_r \cdot 2^{53}$.

This also implies that we need at least the time taken to generate these. Furthermore, we also need to compute the possible candidates for each of the $N_r \cdot 2^{52}$ pairs which passed the filter. As seen in Section 5.3.2, this can be done in time $2^{48}$ for each pair. Hence, we also need to perform a counter increment $4 \cdot N_r \cdot 2^{52} \cdot 2^{48} = N_r \cdot 2^{102}$ times. Finally, for all the candidates with a high enough score, we need to brute-force the 32 remaining bits of the key. This requires $2^{128} \cdot \Pr[\text{wrong}]$ encryptions. The corresponding complexities for different values of $N_r$ are given in Table 5.2.

| $N_r$ | $\Pr[\text{wrong}]$ | Data | Time | Memory |
|---|---|---|---|---|
| $2^5$ | $2^{-1.22}$ | $2^{58}$ | $2^{126.78}$ | $2^{89}$ |
| $2^7$ | $2^{-1.47}$ | $2^{60}$ | $2^{126.53}$ | |
| $2^9$ | $2^{-2.06}$ | $2^{62}$ | $2^{125.94}$ | |
| $2^{11}$ | $2^{-3.67}$ | $2^{64}$ | $2^{124.34}$ | |

Table 5.2: Data, time and memory complexity of a truncated differential attack on Twine-128.

## 5.4 Optimizing the Search for High Probability Differentials

As shown in the previous section truncated differentials can be used directly to attack (round-reduced) block ciphers. However they can also be used to optimize the search for high probability differentials. Indeed, by providing a "template" which differential characteristics should follow, it can reduce the size of the search space significantly and make the computation of a lower bound on a differential probability tighter. A similar approach was used in [CFG+15] to identify high probability differentials for Prince which were then used in a multiple differential attack which is the best attack on this cipher today. Lac [ZWW+14], a lightweight candidate of the Caesar competition based on a simplified version of LBlock called LBlock-s, has been the target of another high probability differential search by Leurent [Leu16] which uses similar ideas.

In both cases, the method has been the same: first identify a high probability differential trail and then use a heuristic method to compute a lower bound on the probability of a differential by essentially clustering all characteristics following said truncated differential. Since we have iterated truncated trails covering any amount of rounds for Twine, we apply this method on this cipher to identify high probability differentials.

For a truncated characteristic $T$ covering $r$ rounds, we denote $\Pr[\delta \rightsquigarrow \Delta]$ the probability of the differential $(\delta \rightsquigarrow \Delta)$ obtained by summing the probabilities of all the differential trails mapping $\delta$ to $\Delta$ which follow the truncated trail. Using these probabilities, we build a matrix $M(C)$ such that $M(T)_{i,j} = \Pr[i \rightsquigarrow j]$. To obtain the distribution of $\Delta$ given $\delta$, we simply multiply a vector made of zeroes everywhere except in position $\delta$, where it is equal to 1, by $M(T)$. Note that the sum of the probabilities of the values $\Delta$ obtained in this fashion is not equal to 1 as the truncated trail itself does not have a probability of 1. Given $M(T)$, finding the differential with the highest probability can be done easily by finding the maximum coefficient in the matrix. The size of $M(T)$ is limited by only taking into account the values of $\delta$ and $\Delta$ which are coherent with $T$.

In order to obtain the distribution of $\Delta$ after two iterations of the trail $T$, we multiply the same vector by the matrix $M(T) \times M(T)$, where "×" denotes regular matrix multiplication. This construction can of course be iterated.

In the case of Twine, we computed two matrices $M(\mathcal{L}_1)$ and $M(\mathcal{R}_1)$ corresponding to the truncated trails $\mathcal{L}_1$ and $\mathcal{R}_1$ described in Figures 5.8a and 5.8b respectively, where zero differences are represented in black and squares correspond to places where cancellations are necessary. Both $M(\mathcal{L}_1)$ and $M(\mathcal{R}_1)$ are square matrices of size $2^{12} \times 2^{12}$ because both trails have only 3 non-zero nibbles as both their input and output. Using different multiplications of these, we found the high probability differentials given in Table 5.3.

| Rds. | Input difference $\delta$ | Output difference $\Delta$ | $\Pr[\delta \rightsquigarrow \Delta]$ | $2^{-2(\#\text{Active S-Boxes})}$ |
|---|---|---|---|---|
| 4 | 10 20 00 60 00 00 00 00 | 00 00 20 00 60 00 00 60 | $2^{-17.496}$ | $2^{-18}$ |
| | 60 20 00 60 00 00 00 00 | 00 00 20 00 60 00 00 10 | $2^{-17.496}$ | |
| | 30 60 00 30 00 00 00 00 | 00 00 60 00 30 00 00 10 | $2^{-17.759}$ | |
| | 10 60 00 30 00 00 00 00 | 00 00 60 00 30 00 00 30 | $2^{-17.759}$ | |
| 8 | 10 20 00 60 00 00 00 00 | 60 20 00 10 00 00 00 00 | $2^{-34.542}$ | $2^{-36}$ |
| | 10 20 00 60 00 00 00 00 | 60 20 00 f0 00 00 00 00 | $2^{-34.981}$ | |
| | f0 20 00 60 00 00 00 00 | 60 20 00 10 00 00 00 00 | $2^{-34.981}$ | |
| | d0 f0 00 80 00 00 00 00 | 80 f0 00 d0 00 00 00 00 | $2^{-34.994}$ | |
| 12 | 10 20 00 10 00 00 00 00 | 00 00 20 00 60 00 00 10 | $2^{-52.083}$ | $2^{-54}$ |
| | 10 20 00 60 00 00 00 00 | 00 00 20 00 10 00 00 10 | $2^{-52.083}$ | |
| | 80 f0 00 80 00 00 00 00 | 00 00 f0 00 d0 00 00 80 | $2^{-52.144}$ | |
| | 80 f0 00 d0 00 00 00 00 | 00 00 f0 00 80 00 00 80 | $2^{-52.144}$ | |
| 16 | 60 20 00 60 00 00 00 00 | 60 20 00 60 00 00 00 00 | $2^{-67.538}$ | $2^{-72}$ |
| | 30 60 00 30 00 00 00 00 | 30 60 00 30 00 00 00 00 | $2^{-67.595}$ | |
| | 90 30 00 90 00 00 00 00 | 90 30 00 90 00 00 00 00 | $2^{-67.626}$ | |
| | 80 f0 00 80 00 00 00 00 | 80 f0 00 80 00 00 00 00 | $2^{-67.762}$ | |

Table 5.3: High probability differentials for round-reduced Twine.

As we can see, the highest probability for a differential over 4-rounds is higher than we might expect. Indeed, 9 S-Boxes are involved in it and the maximum probability for a differential in the S-Box is $2^{-2}$. Hence, the maximum probability of a characteristic is $2^{-18}$, which is smaller than the value of $2^{-17.5}$ our model predicts and which we checked experimentally. The gain then increases as the number of rounds increases. For 12 rounds, we have 27 active S-Boxes which means that the probability of a characteristic cannot be higher than $2^{-54}$ and yet the highest differential probability is at least $2^{-52.1}$.

Leurent obtained more impressive results for LBlock-s (e.g. a lower bound of $2^{-29.8}$ for 8 rounds) which might be surprising at first glance since the linear layer of these two ciphers are very similar and both use S-Boxes with a maximum differential probability equal to $2^{-2}$. However, the distribution of the coefficients in the difference distribution tables of the S-Boxes of these ciphers are different. For instance, with $S_L$ and $S_T$ denoting the S-Boxes of LBlock-s and Twine respectively, we have $P\Big[S_L(x \oplus \delta) \oplus S_L(x) = 4\Big] = 2^{-2}$ for $\delta \in \{4, 5, 6, 7\}$ while there exists only one $\delta$ such that $P\Big[S_T(x \oplus \delta) \oplus S_T(x) = \Delta\Big] = 2^{-2}$ for any $\Delta \neq 0$. In other words, the distribution of the output differences is closer to being uniform in Twine than in LBlock-s (and LBlock). To study the consequences of these variations in differential behavior, we reiterated our differential search by replacing the S-Box of Twine by that of LBlock-s. We obtained four distinct differentials with probability at least $2^{-31.7}$ for 8-rounds.[2] This result is $2^{4.3}$ times better than what a wide-trail argument would give and $2^3$ times higher than for the Twine S-Box.

Our findings highlight both how large truncated differentials can be leveraged to prove tighter lower bounds on differential probabilities and how the distribution of the coefficients in the difference distribution table of a S-Box as a whole should be taken into account when designing a primitive in contrast to simply looking at the maximum coefficient, as is often the case when wide-trail arguments are used.

The differential properties of S-Boxes, such as those built using a finite field multiplicative inverse, are discussed much more thoroughly in Part II. In particular, Section 8.3.1.1 is devoted to such functions.

[2] Note that Leurent used a truncated differential with 17 active S-Boxes while ours has 18. This difference is likely to account for the factor $2^{1.9}$ separating our results.

Chapter 6

# Design Strategies for ARX-based Block Ciphers

Arx, standing for Addition/Rotation/Xor, is a class of symmetric-key algorithms designed using only the following simple operations: modular addition, bitwise rotation and exclusive-or. In contrast to S-box-based designs, where the only non-linear elements are the substitution tables (S-boxes), arx designs rely on modular addition as the only source of non-linearity.

Notable representatives of the arx class include the stream ciphers Salsa20 and ChaCha20 [Ber08b, Ber08a], the Sha-3 finalists Skein [NLS+10] and blake [AHMP08] as well as several lightweight block ciphers such as tea, xtea [NW97], etc. In fact, in a related work [DLCK+15], Daniel Dinu and some of my colleagues reported that the most efficient block cipher software implementations on small processors belonged to ciphers from the arx class. Those are the Chaskey-cipher [MMH+14] by Mouha *et al.*, Speck [BSS+13] by the American National Security Agency (nsa) and lea [HLK+14] by the South Korean Electronic and Telecommunications Research Institute.[1]

For all these algorithms, the choice of using the arx paradigm was based on four observations.

1. Getting rid of the table look-ups, associated with S-Box based designs, increases the resilience against side-channel attacks.

2. This design strategy minimizes the total number of operations performed during an encryption, allowing particularly fast software implementations.

3. In software, few additional registers are needed to store intermediate results. As loading and spilling to ram are slow operations, minimizing those not only decreases the memory footprint of the algorithm, it also makes it faster.

4. The computer code describing such algorithms is very small, making this approach especially appealing for lightweight block ciphers where the memory requirements are the harshest.

While this was not mentioned in the original paper on Speck, its designers later published an invited paper at the *Design Automation Conference* [BTCS+15]. It provides

[1] Speck and the MAC Chaskey are being considered for standardization by ISO.

little information but its authors do say that they aimed at low code size and tried to prevent word copies.

Despite the widespread use of ARX ciphers, the following problem has remained opened until now.

> Is it possible to design an ARX cipher that is provably secure against single-trail differential and linear cryptanalysis *by design*?

To the best of our knowledge, there has only been one attempt at tackling this issue. In [BVC16] Biryukov *et al.* have proposed several ARX constructions for which it is feasible to compute the exact maximum differential and linear probabilities over any number of rounds. However, these constructions are limited to 32-bit blocks. The general case of this problem, addressing any block size, has still remained without a solution.

More generally, the formal understanding of the cryptographic properties of ARX is far less satisfying than that of, for example, S-Box-based substitution-permutation networks (SPN). Indeed, the wide trail strategy [DR01] (WTS) and the wide trail argument [DR02] provide a way to design S-box based SPNs with provable resilience against differential and linear attacks. It relies on bounding the number of active S-Boxes in a differential (resp. linear) trail and deducing a lower bound on the best expected differential (resp. linear) probability.

In this chapter, we propose two different strategies to build ARX-based block ciphers with provable bounds on the maximum expected differential and linear probabilities, thus providing a solution to the open problem stated above.

The first strategy is called the *Long Trail Strategy* (LTS). It borrows the idea of counting the number of active S-Boxes from the wide trail argument but the overall principle is actually the opposite to the wide trail strategy as described in [DR01]. While the WTS dictates the spending of most of the computational resources in the linear layer in order to provide good diffusion between small S-boxes, the LTS advocates the use of large and comparatively expensive S-Boxes in conjunction with cheaper and weaker linear layers. We formalize this method and describe the *Long Trail argument* that can be used to bound the differential and linear trail probabilities of a block cipher built using this strategy. This strategy is described in Section 6.2 (p. 106).

In Section 6.3 (p. 112), we propose the LAX construction, where bit rotations are replaced with a more general linear permutation. The bounds on the differential probability are expressed as a function of the branching number of the linear layer. We note that the key insight behind this construction has been published in [Wal03b], but its realization has been left as a challenge. But first, we introduce the notation and concepts we need in Section 6.1.

## 6.1 Preliminaries

Let $f : \mathbb{F}_2^n \rightarrow \mathbb{F}_2^n$, $(a, b) \in \mathbb{F}_2^n \times \mathbb{F}_2^n$ and $x \in \mathbb{F}_2^n$. We denote the probability of the differential trail $(a \rightsquigarrow b)$ by $\Pr[f(x) \oplus f(x \oplus a) = b]$ and the correlation of the linear approximation $(a \rightsquigarrow b)$ by $(2 \Pr[a \cdot x = b \cdot f(x)] - 1)$ where $y \cdot z$ is the scalar product of $y$ and $z$.

In an iterated block cipher, not all differential (respectively linear) trails are possible. Indeed, they must be coherent with the overall structure of the round function. For example, it is well known that a 2-round differential trail for the AES with less than 4 active S-Boxes is impossible. To capture this notion, we use the following definition.

**Definition 6.1.1** (Valid Trail). *Let $f$ be an $n$-bit permutation. A trail $a_0 \rightarrow \ldots \rightarrow a_r$ for $r$ rounds of $f$ is a* valid trail *if* $\Pr[a_i \rightarrow a_{i+1}] > 0$ *for all $i$ in $[0, r-1]$. The set of all valid $r$-round differential (respectively linear) trails for $f$ is denoted $\mathcal{V}_\delta(f)^r$ (resp. $\mathcal{V}_\ell(f)^r$).*

We use the acronyms MEDCP and MELCC to denote respectively *maximum expected differential characteristic probability* and *maximum expected linear characteristic correlation* – a signature introduced earlier in [KS07]. The MEDCP of the keyed function $f_{k_i} : x \mapsto f(x \oplus k_i)$ iterated over $r$ rounds is defined as follows:

$$\textsc{medcp}(f^r) = \max_{(\Delta_0 \rightarrow \ldots \Delta_r) \in \mathcal{V}_\delta(f)^r} \prod_{i=0}^{r-1} \Pr[\Delta_i \rightsquigarrow \Delta_{i+1}] ,$$

where $\Pr[\Delta_i \rightsquigarrow \Delta_{i+1}]$ is the expected value of the differential probability of $\Delta_i \rightsquigarrow \Delta_{i+1}$ for the function $f_k$ when $k$ is picked uniformly at random. $\textsc{melcc}(f^r)$ is defined analogously. Note that $\textsc{medcp}(f^r)$ and $\left(\textsc{medcp}(f^1)\right)^r$ are *not* equal.

As designers, we thrive to provide upper bounds for both $\textsc{medcp}(f^r)$, so as to evaluate the resilience against single-trail differential cryptanalysis, and $\textsc{melcc}(f^r)$ to better investigate single-trail linear attacks. Doing so allows us to compute the number of rounds $f$ needed in a block cipher for the probability of all trails to be too low to be usable. In practice, we want $\textsc{medcp}(f^r) \ll 2^{-n}$ and $\textsc{melcc}(f^r) \ll 2^{-n/2}$ where $n$ is the block size.

While this strategy is the best known, the following limitations must be taken into account by algorithm designers.

1. The quantities $\textsc{medcp}(f^r)$ and $\textsc{melcc}(f^r)$ are relevant only if we make the *Markov assumption*, meaning that the differential and linear probabilities are independent in each round. This would be true if the subkeys were picked uniformly and independently at random but, as the master key has a limited size, it is not the case.

2. These quantities are averages taken over all possible keys: it is not impossible that there exists a weak key and a $r$-round differential trail $T$ such that the probability of $T$ is higher than $\textsc{medcp}(f^r)$ for this particular key. The same holds for the linear probability.

3. These quantities deal with unique trails. However, it is possible that several differential trails share the same input and output differences, thus leading to a higher probability for said differential transition. This so-called *differential effect* can be leveraged to decrease the data complexity of differential attack. The same holds for linear attacks where several approximations may form a linear hull.

Still, this type of bound is the best that can be achieved in a generic fashion (to the best of our knowledge). In particular, this is the type of bound provided by the wide trail argument used in the AES.

## 6.2 ARX-Based Substitution-Permutation Network

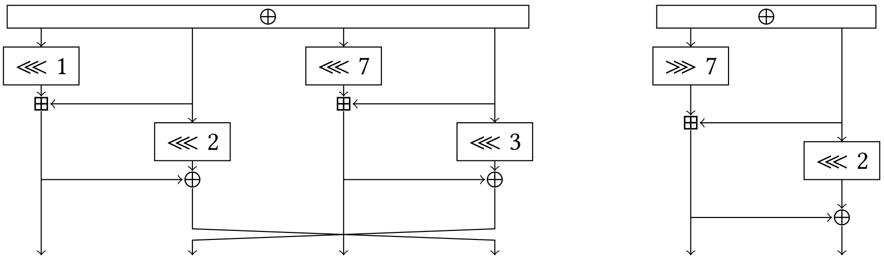
In this section, we present a general design strategy for building ARX-based block ciphers borrowing techniques from SPN design. The general idea is to build an SPN with ARX-based S-boxes instead of with S-boxes based on look-up tables (LUT). The proofs for the bound on the MEDCP and MELCC are inspired by the wide trail argument introduced in the design of the AES [DR02]. However, because of the use of large S-Boxes, the method used relies on a different type of interaction between the linear and non-linear layers. We call the corresponding design strategy the *Long Trail Strategy*. It is quite general and could be also applied in other contexts e.g. for non-ARX constructions.

First, we present possible candidates for the ARX-based S-Box and, along the way, identify the likely reason behind the choice of the rotation constants in SPECK-32. Then, we describe the Long Trail Strategy in more details. Finally, we present two different algorithms for computing a bound for the MEDCP and MELCC of block ciphers built using a LT strategy. We also discuss how to ensure that the linear layer provides sufficient diffusion.

### 6.2.1 ARX-Boxes

**Definition 6.2.1** (ARX-box)**.** *An ARX-box is a permutation on m bits (where m is much smaller than the block size) which relies entirely on addition, rotation and XOR to provide both non-linearity and diffusion. An ARX-box is a particular type of S-Box.*

Possible constructions for ARX-boxes can be found in a recent paper by Biryukov *et al.* [BVC16]. A first one is based on the MIX function of Skein [NLS⁺10] and is called MARX-2. The rotation amounts, namely $\{1, 2, 7, 3\}$, were chosen so as to minimize the differential and linear probabilities. The key addition is done over the full state. The second construction is called SPECKEY and consists of one round of SPECK-32 [BSS⁺13] with the key added to the full state instead of only to half the state as in the original algorithm. The two constructions, MARX-2 and SPECKEY, are shown in Figures 6.1a and 6.1b, where the branch size is 8 bits for MARX-2 and 16 bits for SPECKEY. The differential and linear bounds for them are given in Table 6.1.



(a) MARX-2.

(b) SPECKEY.

Figure 6.1: Key addition followed by the candidate 32-bit ARX-boxes.

While it is possible to choose the rotations used in SPECKEY in such a way as to slightly decrease the differential and linear bounds[2], such rotations are more expen-

[2] Both can be lowered by a factor of 2 if we choose rotations (9, 2), (9, 5), (11, 7) or (7, 11) instead of (7, 2).

| | $r$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Marx-2 | MEDCP$(M^r)$ | 0 | −1 | −3 | −5 | −11 | −16 | −22 | −25 | −29 | −35 |
| | MELCC$(M^r)$ | 0 | 0 | −1 | −3 | −5 | −8 | −10 | −13 | −15 | −17 |
| Speckey | MEDCP$(S^r)$ | 0 | −1 | −3 | −5 | −9 | −13 | −18 | −24 | −30 | −34 |
| | MELCC$(S^r)$ | 0 | 0 | −1 | −3 | −5 | −7 | −9 | −12 | −14 | −17 |

Table 6.1: Maximum expected differential characteristic probabilities (MEDCP) and maximum expected absolute linear characteristic correlations (MELCC) of Marx-2 and Speckey ($\log_2$ scale); $r$ is the number of rounds.

sive on small microcontrollers which only have instructions implementing rotations by 1 and by 8 (in both directions). The designers of Speck made the same observation in [BTCS+15].

### 6.2.2 Naïve Approaches and Their Limitations

A straight-forward approach to build ARX-based ciphers with provable bounds on the MEDCP and the MELCC is to use an SPN structure where the S-boxes are replaced by ARX operations for which we can compute the MEDCP and MELCC. This is indeed the strategy we follow but care must be taken when actually choosing the ARX-based operations and the linear layer.

Let us for example build a 128-bit block cipher with an S-Box layer consisting in one iteration of Speckey on each 32-bit word and with an MDS linear layer, say a multiplication with the `MixColumns` matrix with elements in $\mathbb{F}_{2^{32}}$ instead of $\mathbb{F}_{2^8}$. The MEDCP bound of such a cipher computed using a classical wide trail argument would be equal to 1! Indeed, there exists probability 1 differentials for 1-round Speckey so that, regardless of the number of active S-Boxes, the bound would remain equal to 1. Such an approach is therefore not viable.

As the problem identified above stems from the use of 1-round Speckey, we now replace it with 3-round Speckey where the iterations are interleaved with the addition of independent round keys. The best linear and differential probabilities are no longer equal to 1, meaning that it is possible to build a secure cipher using the same layer as before provided that enough rounds are used. However, such a cipher would be very inefficient. Indeed, the MDS bound imposes that 5 ARX-boxes are active every 2 rounds, so that the MEDCP bound is equal to $p_d^{5r/2}$ where $r$ is the number of rounds and $p_d$ is the best differential probability of the ARX-box (3-rounds Speckey). To push the bound below $2^{-128}$ we need at least 18 SPN rounds, meaning 54 parallel applications of the basic ARX-round! We will show that, with our alternative approach, we can obtain the same bounds with much fewer rounds.

### 6.2.3 The Long Trail Design Strategy

Informed by the shortcomings of the naïve design strategies described in the previous section, we devised a new method to build ARX-based primitives with provable linear and differential bounds. It is based on the following observation.

**Observation 6.2.1** (Impact of Long Trails)**.** *Let $d(r)$ and $\ell(r)$ be the MEDCP and MELCC of some ARX-box iterated $r$ times and interleaved with the addition of independent sub-*

*keys. Then, in most cases:*

$$d(qr) \ll d(r)^q \text{ and } \ell(qr) \ll \ell(r)^q .$$

*In other words, in order to diminish the MEDCP and MELCC of a construction, it is better to allow long trails of ARX-boxes without mixing.*

For example, if we look at SPECKEY, the MEDCP for 3 rounds is $2^{-3}$ and that of 6 rounds is $2^{-15}$ which is far smaller than $(2^{-3})^2 = 2^{-6}$ (see Table 6.1). Similarly, the MELCC for 3 rounds is $2^{-1}$ and after 6 rounds it is $2^{-7} \ll (2^{-1})^2$.

In fact, a similar observation has been made by Nikolić when designing the CAESAR candidate family Tiaoxin [Nik15]. It was later generalized to larger block sizes in [JN16], where Jean and Nikolić present, among others, the AES-based $\mathcal{A}^2_\oplus$ permutation family. It uses a partial Large-Box layer where the Large-Box consists of 2 AES rounds and a word-oriented linear layer in such a way that some of the Large-Box calls can be chained within 2-round long trails. Thus, they may use the 4-round bound on the number of active 8-bit AES S-Boxes, which is 25, rather than twice the 2-round bound, which would be equal to 10 (see Table 6.2). Their work on this permutation can be interpreted as a particular case of the observation above.

| # R | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| # Active S-Boxes | 1 | 5 | 9 | 25 | 26 | 30 | 34 | 50 | 51 | 55 |

Table 6.2: Bound on the number of active 8-bit S-Boxes in a differential (or linear) trail for the AES.

**Definition 6.2.2** (Long Trail). *We call* Long Trail (LT) *an uninterrupted sequence of calls to an ARX-box interleaved with key additions. No difference can be added into the trail from the outside. Such trails can happen for two reasons.*

1. *A* Static Long Trail *occurs with probability 1 because one output word of the linear layer is an unchanged copy of one of its input words.*
2. *A* Dynamic Long Trail *occurs within a specific differential trail because one output word of the linear layer consists of the XOR of one of its input words with a non-zero difference and a function of words with a zero difference. In this way the output word of the linear layer is again equal to the input word as in a Static LT, but here this effect has been obtained dynamically.*

**Definition 6.2.3** (Long Trail Strategy). *The Long Trail Strategy is a design guideline: when designing a primitive with a rather weak but large S-Box (say, an ARX-based permutation), it is better to foster the existence of long trails rather than to have maximum diffusion in each linear layer.*

This design principle has an obvious caveat: although slow, diffusion is necessary! Unlike the WTS, in this context it is better to trade some of the power of the diffusion layer in favor of facilitating the emergence of long trails.

The Long Trail Strategy is a method for building secure and efficient ciphers using a large but weak S-Box $S$ such that we can bound the MEDCP (and MELCC) of several iterations of $x \mapsto S(x \oplus k)$ with independent round keys. In this paper, we focus on the case where $S$ consists of ARX operations but this strategy could have broader applications such as, as briefly discussed above, the design of block ciphers operating on large blocks using the AES round function as a building block.

In a way, this design method is the direct opposite of the wide trail strategy as it is summarized by Daemen and Rijmen in [DR01] (emphasis ours):

> Instead of spending most of the resources on large S-boxes, the wide trail strategy aims at designing the round transformation(s) such that there are no trails with a low bundle weight. In ciphers designed by the wide trail strategy, *a relatively large amount of resources is spent in the linear step* to provide high multiple-round diffusion.

The long trail approach *minimizes* the amount of resources spent in the linear layer and does spend most of the resources on large S-Boxes. Still, as discussed in the next section, the method used to bound the MEDCP and MELCC in the Long Trail Strategy is heavily inspired by the one used in the wide trail strategy.

### 6.2.3.1 A Cipher Structure for the LT Strategy

We can build block ciphers based on the Long Trail Strategy using the following two-level structure. First, we must choose an S-Box layer operating on $w$ words in parallel. The composition of a key addition in the full state and the application of this S-Box layer is called a *round*. Several rounds are iterated and then a word-oriented linear mixing layer is applied to ensure diffusion between the words. The composition of $r$ rounds followed by the linear mixing layer is called a *step*[3], as described in Fig. 6.2. The encryption thus consists of iterating such steps. We used this design strategy to build a block cipher family, SPARX, which we describe in Chapter 7 (p. 117).

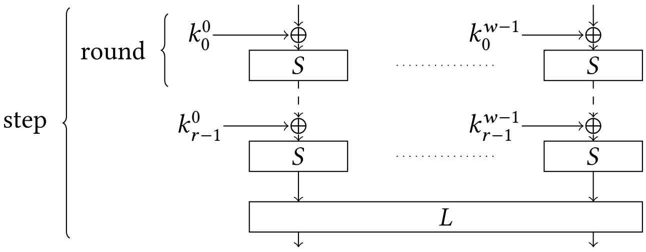


Figure 6.2: A cipher structure for the LT strategy.

### 6.2.3.2 Long Trail-Based Bounds

In what follows, we only discuss differential long trails for the sake of brevity. Linear long trails are treated identically.

**Definition 6.2.4** (Truncated LT-Decomposition). *Consider a cipher with a round function operating on w words. A truncated differential trail is a sequence of values of* $\{0, 1\}^w$ *describing whether an S-Box is active at a given round. The* LT-Decomposition *of a truncated differential trail is obtained by grouping together the words of the differential*

[3] This terminology is borrowed from the specification of LED [GPPR11] which also groups several calls of the round function into a step.

*trails into long trails and then counting how many active long trails of each length are present. It is denoted $\{t_i\}_{i\geq 1}$ where $t_i$ is equal to the number of truncated long trails with length $i$.*

**Example 6.2.1.** *Consider a 64-bit block cipher using a 32-bit S-Box, one round of Feistel network as its linear layer and 4 steps without a final linear layer. Consider the differential trail $(\delta_0^L, \delta_0^R) \rightarrow (\delta_1^L, \delta_1^R) \rightarrow (0, \delta_2^R) \rightarrow (\delta_3^L, 0)$ (see Fig. 6.3 where the zero difference is dashed). Then this differential trail can be decomposed into 3 long trails represented in black, blue and red: the first one has length 1 and $\delta_0^R$ as its input; the second one has length 2 and $\delta_0^L$ as its input; and the third one has length 3 and $\delta_1^L$ as its input so that the* LT *decomposition of this trail is $\{t_1 = 1, t_2 = 1, t_3 = 1\}$. Using the terminology introduced earlier, the first two trails are Static* LT*, while the third one is a Dynamic* LT*.*

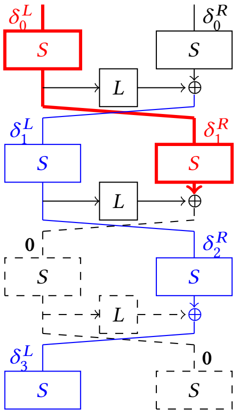


Figure 6.3: An example of active LT decomposition.

**Theorem 6.2.1** (Long Trail Bound)**.** *Consider a truncated differential trail $T$ covering $r$ rounds consisting of an S-Box layer with S-Box $S$ interleaved with key additions and some linear layer. Let $\{t_i\}_{i\geq 1}$ be the* LT *decomposition of $T$. Then the probability $p_D$ of any fully specified differential trail fitting in $T$ is upper-bounded by*

$$p_D \leq \prod_{i\geq 1} \left(\textsc{medcp}(S^i)\right)^{t_i}$$

*where* $\textsc{medcp}(S^i)$ *is an upper-bound on the probability of a differential trail covering $i$ iterations of $S$.*

*Proof.* Let $\Delta_{i,s} \rightsquigarrow \Delta_{j,s+1}$ denote any differential trail occurring at the S-Box level in one step, so that the S-Box with index $i$ at step $s$ sees the transition $\Delta_{i,s} \rightsquigarrow \Delta_{j,s+1}$. By definition of a long trail, we have in each long trail a chain of differential trails $\Delta_{i_0,s_0} \rightsquigarrow \Delta_{i_1,s_0+1} \rightsquigarrow \ldots \rightsquigarrow \Delta_{i_t,s_0+t}$ which, because of the lack of injection of differences from the outside, is a *valid trail* for $t$ iterations of the S-Box. This means that the probability of any differential trail following the same sequence of S-boxes as in this long trail is upper-bounded by $\textsc{medcp}(S^t)$. We simply bound the product by the product of the bounds to derive the theorem. □

### 6.2.4 Choosing the Linear Layer: Bounding the MEDCP and MELCC while Providing Diffusion

In order to remain as general as possible, in this section we do not consider the details of a specific S-Box but instead we focus on fleshing out design criteria for the linear layer. All the information on the S-Box that is necessary to follow the explanation is the MEDCP and MELCC of its $r$-fold iterations including the key additions e.g. the data provided in Table 6.1 for our ARX-box candidates.

As the linear layers we consider may be weaker than those usually to design SPNs, it is also crucial to ensure that ciphers built using such a linear layer are not vulnerable to integral attacks [KW02], in particular those based on the division property [Tod15b]. Incidentally, this gives us a criteria quantifying the diffusion provided by several steps of the cipher.

In this section, we propose two methods for bounding the MEDCP and MELCC of several steps of a block cipher. The first one is applicable to any linear layer but is relatively inefficient, while the second one works only for a specific subset of linear layers but is very efficient.

When considering truncated differential trails, it is hard to bound the probability of the event that differences in two or more words cancel each other in the linear layer. Therefore, for simplicity we assume that such cancellations happen *for free* i.e. with probability 1. Due to this simplification, we expect our bounds to be much higher than the actual best differential probabilities. In other words, we *underestimate* the security of the cipher. Note that we also exclude the cases where the full state at some round has zero difference as the latter is impossible due to the cipher being a permutation.

#### 6.2.4.1 Algorithms for Bounding MEDCP and MELCC of a Cipher

We propose generic approaches that do not depend on the number of rounds per step. In fact, to fully avoid the confusion between *rounds* and *steps* in what follows we shall simply refer to SPN *rounds*. One way to bound the MEDCP and of a cipher is described in Algorithm 6.1.

**Algorithm 6.1** Finding the best bound on the DP of all differential trails covering $r$ rounds for a long-trail based block cipher with S-Box $S$.
**Input:** MEDCP$S^i$ for $1 \leq i; r$;
**Output:** bound on all differential trail probabilities $p_{\max}$.

```
$p_{\max} = 1$
for all truncated trails $T$ covering $r$ rounds do
    $\{t_i\}_{i=0}^{r} \leftarrow$ LT-decomposition of $T$
    $p_T = \prod_{i=0}^{r} \left(\text{MEDCP}S^i\right)^{t_i}$
    $p_{\max} = \min(p_T, p_{\max})$
end for
return $p_{\max}$
```

The algorithm we actually implemented is more sophisticated than Algorithm 6.1 as the latter is not efficient enough in practice. More details about the specifics of the actual algorithm, designed by my colleague Aleksei Udovenko, can be found in the appendix of the full version of our paper which is available on eprint [DPU⁺16b].

Both algorithms can be used to bound the probability of linear trails. Propagation of a linear mask through some linear layer can be described by multiplying the mask by the transposed inverse of the linear layer matrix. In our matrix notation we can easily transpose the matrix but inversion is harder. However, we can build the linear trails bottom-up (i.e. starting from the last round): in this case we need only the transposed initial matrix. Our algorithm does not depend on the direction, so we obtain bounds on linear trails probabilities by running the algorithm on the transposed matrix using the linear bounds for the iterated S-box.

#### 6.2.4.2 Ensuring Resilience Against Integral Attacks

As illustrated by the structural attack against SASAS and its generalization described in Chapter 11.5 (p. 212), an SPN with few rounds may be vulnerable to integral attacks. This attack strategy has been explored by Todo [Tod15b] who proposed the so-called *division property* as a means to track which bit should be fixed in the input to have a balanced output. He also described an algorithm allowing an attacker to easily find such distinguishers.

We implemented this algorithm to search for division-property-based integral trails covering as many rounds as possible. With it, for each matrix candidate we compute a maximum number of rounds covered by such a distinguisher. This quantity can then be used by the designer of the primitive to see if the level of protection provided against this type of attack is sufficient or not.

Tracking the evolution of the division property through the linear layer requires special care. In order to do this, we first make a copy of each word and apply the required XORs from the copy to the original words. Due to such state expansion, the algorithm requires both a lot of memory and time. In fact, it is even infeasible to apply on some matrices. To overcome this issue, we ran the algorithm with reduced word size. During our experiments, we observed that such an optimization may only result in longer integral characteristics and that this side effect occurs only for very small word sizes (4 or 5 bits). In light of this, we conjecture that the values obtained in these particular cases are upper bounds and are very close to the values which could be obtained without reducing the word size.

## 6.3 Replacing Rotations with Linear Layers: the LAX Construction

In this section we outline an alternative strategy for designing an ARX cipher with provable bounds against differential and linear cryptanalysis. It is completely independent of the Long Trail Strategy outlined in the previous sections and uses the differential properties of modular addition to derive proofs of security.

### 6.3.1 Motivation

In his Master thesis [Wal03b], Wallén posed the challenge to design a cipher that uses only addition modulo-$2^n$ and GF(2)-affine functions, and that is provably resistant against differential and linear cryptanalysis [Wal03b, Sect. 5]. In this subsection we partially solve this challenge by proposing a construction with provable bounds against single-trail differential cryptanalysis.

To mitigate the ambiguity of the "+" notation, we denote the XOR with $\oplus$ and the addition modulo $2^n$ with "$\boxplus$".

### 6.3.2 Theoretical Background

**Definition 6.3.1** (xdp$^+$). *The differential probability DP of addition modulo $2^n$ is defined as:*

$$\Pr[\alpha, \beta \rightsquigarrow \gamma] = 2^{-2n} \cdot \#\Big\{(x,y) : ((x \oplus \alpha) \boxplus (y \oplus \beta)) \oplus (x \boxplus y) = \gamma\Big\} \ ,$$

*where $\alpha$, $\beta$ and $\gamma$ are $n$-bit* xor *differences and $x$ and $y$ are $n$-bit values.*

The linear correlation of addition modulo $2^n$ is defined in a similar way. Efficient algorithms for the computation of the differential probability and the linear correlation of addition modulo $2^n$ have been proposed respectively in [LM02] and [Wal03a, NW06, DRS15]. These results also reveal the following property: the magnitude of both the differential probability and the linear correlation is inversely proportional to the number of bit positions at which the input/output differences (resp. masks) differ. For the differential probability, this fact is formally stated in the form of the following proposition.

**Proposition 6.3.1** (Differential probability bound (for $\boxplus$)). *The differential probability is upper-bounded by $2^{-k}$, where $k$ is the number of bit positions, excluding the most significant bit, at which the bits of the differences are not equal:*

$$\Pr[\alpha, \beta \rightsquigarrow \gamma] \leq 2^{-k}, \ \textit{where}\ k = \#\Big\{i : \neg(\alpha[i] = \beta[i] = \gamma[i]), 0 \leq i \leq w-2\Big\} \ .$$

*Proof.* Follows from [LM02, Alg. 2, Sect. 4]. □

A similar proposition also holds for the linear correlation (see e.g. [BVC16]). Proposition 6.3.1 provides the basis of the design strategy described in the following subsection.

### 6.3.3 The Lax Construction

Lax is a block cipher construction with $2n$-bit block and $n$-bit words. We investigate three instances of Lax designated by the block size: Lax-16, Lax-32 and Lax-64. A brief description of the round function of Lax-$2n$ is shown in Figure 6.4 (without the key additions). It operates on two $n$-bit words $x$ and $y$ using a two-word $(k, k')$ and maps to two $n$-bit words $(u, b)$ as follows:

$$\begin{cases} u = L(y \oplus k') \\ v = L\Big((x \oplus k) \boxplus (y \oplus k')\Big) \ , \end{cases}$$

where $L$ is a linear bijection operating on $n$ bits.

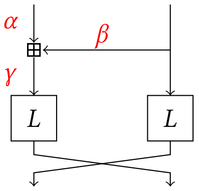


Figure 6.4: The round function of Lax.

We denote by $d$ the branching number of $L$, so that $\mathrm{hw}(x) + \mathrm{hw}(L(x)) \geq d$ for any $x \neq 0$. In fact, the matrix representation of $L$ is chosen as the non-identity part of the generator matrix $G$ of a systematic $[2n, n, d]$ linear code over GF(2) such that $G = [I\ L]$. More specifically, the matrices $L$ for LAX-16, LAX-32 and LAX-64 are derived from the following codes respectively: $[16, 8, 5]$, $[32, 16, 8]$ and $[64, 32, 10]$. In fact, the matrix of LAX-32 is the same as the one used in block cipher ARIA [KKP+04].

### 6.3.4 Bounds on the Differential Probability of LAX

**Lemma 6.3.1.** *For all differences $\alpha \neq 0$, the differential $(\alpha, \alpha \rightsquigarrow \alpha)$ is impossible.*

*Proof.* Let $\Pr[\alpha, \beta \rightsquigarrow \gamma] \neq 0$ for some differences $\alpha$, $\beta$ and $\gamma$. The proof follows from the following two properties of the differential probability of the addition modulo $2^n$ [LM02]:

1. it must hold that $\alpha[0] \oplus \beta[0] \oplus \gamma[0] = 0$, and
2. if $\alpha[i] = \beta[i] = \gamma[i]$ for some $0 \leq i \leq n-2$, then it must hold that $\alpha[i+1] \oplus \beta[i+1] \oplus \gamma[i+1] = \alpha[i]$.

Assuming that $\alpha = \beta = \gamma$, we deduce from the first property that $\alpha[0] = \beta[0] = \gamma[0] = 0$. Furthermore, the second property imposes that $\alpha[i] = \beta[i] = \gamma[i] = 0$, for all $i \geq 1$. Thus, the only value of $\alpha$ for which $\Pr[\alpha, \beta \rightsquigarrow \gamma] \neq 0$ and $\alpha = \beta = \gamma$ is $\alpha = 0$. □

**Theorem 6.3.1** (Differential bound on 3 rounds of LAX-$2n$)**.** *The maximum differential probability of any trail on 3 rounds of LAX-$2n$ is $2^{-(d-2)}$, where $d$ is the branch number of the matrix $L$.*

*Proof.* Let $(\alpha_{i-1}, \beta_{i-1}, \gamma_{i-1})$, $(\alpha_i, \beta_i, \gamma_i)$ and $(\alpha_{i+1}, \beta_{i+1}, \gamma_{i+1})$ be the input/output differences of the addition operations in three consecutive rounds of LAX-$2n$ and let $p_k = \Pr[\alpha_k, \beta_k \rightsquigarrow \gamma_k]$ for $k \in \{i-1, i, i+1\}$ be the differential probability of $(\alpha_k, \beta_k \rightsquigarrow \gamma_k)$. We have to show that $p_{i-1}p_ip_{i+1} \leq 2^{-(d-2)}$ or, equivalently, that $\log_2(p_{i-1}) + \log_2(p_i) + \log_2(p_{i+1}) \leq -(d-2)$. Let $\mathrm{hw}^*(x)$ denote the Hamming weight of $x$, excluding its most significant bit, so that $\mathrm{hw}^*(x) \leq \mathrm{hw}(x) - 1$. We consider two cases.

**Case 1: $\beta_{i-1} \neq \gamma_{i-1}$.** Proposition 6.3.1 imposes $\log_2(p_{i-1}) \leq -\mathrm{hw}^*(\beta_{i-1} \oplus \gamma_{i-1})$ and $\log_2(p_i) \leq -\mathrm{hw}^*(\alpha_i \oplus \beta_i)$. Since $\beta_i = L(\gamma_{i-1})$ and $\alpha_i = L(\beta_{i-1})$ and using the linearity of $L$ we have that $\mathrm{hw}^*(\alpha_i \oplus \beta_i) = \mathrm{hw}^*(L(\beta_{i-1} \oplus \gamma_{i-1}))$. As $\beta_{i-1} \neq \gamma_{i-1}$ it follows that $\mathrm{hw}^*(\beta_{i-1} \oplus \gamma_{i-1}) \neq 0$ and $\mathrm{hw}^*(L(\beta_{i-1} \oplus \gamma_{i-1})) \neq 0$. Thus, we derive:

$$\log_2(p_{i-1}) + \log_2(p_i) \;\leq\; -\mathrm{hw}^*(\beta_{i-1} \oplus \gamma_{i-1})) - \mathrm{hw}^*(L(\beta_{i-1} \oplus \gamma_{i-1})) \quad .$$

The properties of $L$ further imply that $-h(\beta_{i-1} \oplus \gamma_{i-1}) - h(\ell(\beta_{i-1} \oplus \gamma_{i-1})) \leq -d$ and so $-h^*(\beta_{i-1} \oplus \gamma_{i-1}) - h^*(\ell(\beta_{i-1} \oplus \gamma_{i-1})) \leq -(d-2)$. Therefore:

$$\log_2(p_{i-1}) + \log_2(p_i) \leq -(d-2) \quad .$$

**Case 2: $\beta_{i-1} = \gamma_{i-1} \neq 0$.** In this case $\alpha_i = \beta_i = L(\beta_{i-1}) = L(\gamma_{i-1})$. Due to Lemma 6.3.1, $\gamma_i \neq \beta_i$, which means we can apply the argument from *Case 1* on rounds $i$ and $i+1$ to derive the statement of the theorem in this case. □

### 6.3.5 Experimental Results

We have implemented the search algorithm proposed in [BVC16] in order to find the probabilities of the best differential trails in Lax-16 and Lax-32. In Table 6.3, we compare the results to the theoretical bounds computed using Theorem 6.3.1.

| | # Rounds | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Lax-16** | $p_{\text{best}}$ | +0 | −2 | −4 | −7 | −8 | −11 | −13 | −16 | −18 | −20 | −23 | −25 |
| | $c_{\text{best}}$ | +0 | +0 | −1 | −2 | −3 | −5 | −5 | −7 | −8 | −9 | −10 | −11 |
| | $p_{\text{bound}}$ | | | −3 | | | −6 | | | −9 | | | −12 |
| **Lax-32** | $p_{\text{best}}$ | +0 | −2 | −6 | −9 | −11 | −16 | −18 | −20 | −24 | −28 | −29 | −34 |
| | $c_{\text{best}}$ | +0 | +0 | +0 | −4 | −4 | −8 | −8 | −8 | −8 | −12 | −12 | −16 |
| | $p_{\text{bound}}$ | | | −6 | | | −12 | | | −18 | | | −24 |

Table 6.3: Best differential probabilities and best absolute linear correlations ($\log_2$ scale) for up to 12 rounds of Lax.

As we can see, the bound from Theorem 6.3.1 would not hold for the linear case. The problem is the "three-forked branch" in the Lax round function that acts as a xor when the inputs are linear masks rather than differences. Thus, Lax only provides differential bounds and the full solution to the Wallén challenge still remains an open problem.

**Open Problem 6.3.1.** *Is it possible to design a structure similar to Lax in such a way that the maximum linear correlation after several rounds can be upper bounded?*

Chapter 7

# The Sparx Family of Lightweight Block Ciphers

Using the long trail strategy established in the previous chapter, we built a family of lightweight block ciphers called Sparx. All three instances in this family can be entirely specified using only three operations: addition modulo $2^{16}$, 16-bit rotations and 16-bit XOR. To the best of our knowledge this is the first family of arx ciphers that is provably secure against (single trail) linear and differential cryptanalysis *by design*. Furthermore, while one may think that these provable properties imply a performance degradation, we show that it is not the case. On the contrary, Sparx ciphers have very competitive performance on lightweight processors. In fact, the most lightweight version – Sparx-64/128 – is in the top 3 for 16-bit micro-controllers according to the classification method presented in [DLCK+15].

This whole chapter is devoted to this family of ciphers. First, Section 7.1 (p. 117) provides a high level view of these algorithms. More details are provided in Section 7.2 (p. 119) which contains a full specification of each instance in this family. Section 7.3 (p. 121) explains the rationale behind our design choices. Our algorithms are then evaluated from a security stand-point in Section 7.4 (p. 124) and from an implementation point of view in Section 7.5 (p. 128).

## 7.1 High Level View

The plaintexts and ciphertexts consist of $w = n/32$ words of 32 bits each and the key is divided into $v = k/32$ such words. The encryption consists of $n_s$ steps, each composed of an arx-box layer of $r_a$ rounds and a linear mixing layer. In the arx-box layer, each word of the internal state undergoes $r_a$ rounds of Speckey, including key additions. The $v$ words in the key state are updated once $r_a$ arx-boxes have been applied to one word of the internal state. The linear layers $\lambda_w$ for $w = 2, 4$ provide linear mixing for the $w$ words of the internal state.

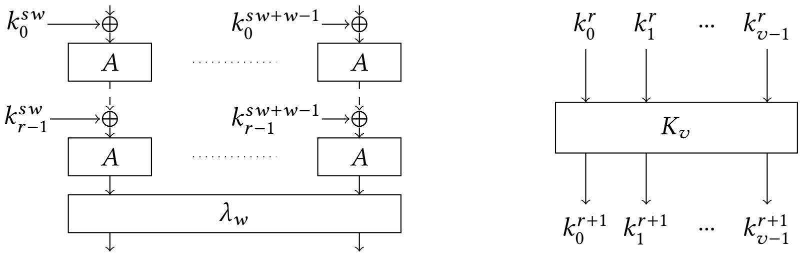
This structure is summarized by the pseudo-code in Algorithm 7.1. The structure of one round is represented in Fig. 7.1, where $A$ is the 32-bit arx-box consisting of one unkeyed Speck32 round. We also use $A^a$ to denote $a$ rounds of Speckey with the corresponding key additions (see Figure 7.2a).

The different versions of Sparx all share the same definition of $A$. However, the permutations $\lambda_w$ and $K_v$ depend on the block and key sizes. The different members of the Sparx-family are specified below. The round keys can either be derived on the fly by applying $K_v$ on the key state during encryption or they can be precomputed and stored. The first option requires less RAM, while the second is faster. The only operations needed to implement any instance of Sparx are:

- addition modulo $2^{16}$, denoted $\boxplus$,
- 16-bit exclusive-or (XOR), denoted $\oplus$, and
- 16-bit rotation to the left or right by $i$, denoted respectively $x \lll i$ and $x \ggg i$.

**Algorithm 7.1** Sparx encryption
**Inputs** plaintext $(x_0, \dots, x_{w-1})$; key $(k_0, \dots, k_{v-1})$
**Output** ciphertext $(y_0, \dots, y_{w-1})$

```
Let $y_i \leftarrow x_i$ for all $i \in [0, \dots, w-1]$
for all $s \in [0, n_s - 1]$ do
    for all $i \in [0, w-1]$ do
        for all $r \in [0, r_a - 1]$ do
            $y_i \leftarrow y_i \oplus k_r$
            $y_i \leftarrow A(y_i)$
        end for
        $(k_0, \dots, k_{v-1}) \leftarrow K_v\big((k_0, \dots, k_{v-1})\big)$        ▷ Update key state
    end for
    $(y_0, \dots, y_{w-1}) \leftarrow \lambda_w\big((y_0, \dots, y_{w-1})\big)$        ▷ Linear mixing layer
end for
Let $y_i \leftarrow y_i \oplus k_i$ for all $i \in [0, \dots, w-1]$        ▷ Final key addition
return $(y_0, \dots, y_{w-1})$
```



(a) Round function of Sparx.

(b) Key schedule.

Figure 7.1: A high level view of step $s$ of Sparx.

We claim that no attack using less than $2^k$ operations exists against Sparx-n/k in neither the single-key nor in the related-key setting. We also faithfully declare that we have not hidden any weakness in these ciphers. Sparx is free for use and its source code is available in the public domain [1].

[1] See https://www.cryptolux.org/index.php/SPARX.

## 7.2 Specification

Table 7.1 summarizes the different Sparx instances and their parameters. The quantity $\min_{\text{secure}}(n_s)$ corresponds to the minimum number of steps for which we can prove that the MEDCP is below $2^{-n}$, that the MELCC is below $2^{-n/2}$ for the number of rounds per step chosen and for which we cannot find integral distinguishers covering this amount of steps.

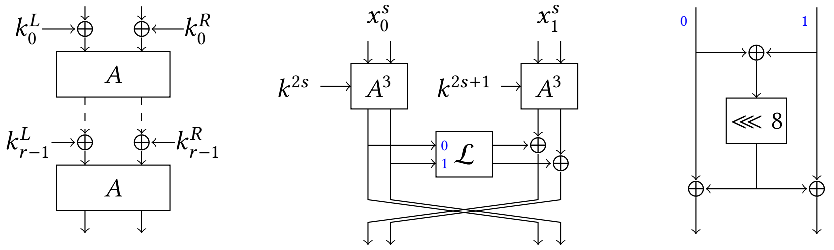
| | Sparx-64/128 | Sparx-128/128 | Sparx-128/256 |
|---|---|---|---|
| # State words $w$ | 2 | 4 | 4 |
| # Key words $v$ | 4 | 4 | 8 |
| # Rounds/Step $r_a$ | 3 | 4 | 4 |
| # Steps $n_s$ | 8 | 8 | 10 |
| Best Attack (# rounds) | 15/24 | 22/32 | 24/40 |
| $\min_{\text{secure}}(n_s)$ | 5 | 5 | 5 |

Table 7.1: The different Sparx instances.

### 7.2.1 Sparx-64/128

The lightest instance of Sparx is Sparx-64/128. It operates on two words of 32 bits and uses a 128-bit key. There are 8 steps and 3 rounds per step. As it takes 5 steps to achieve provable security against linear and differential attacks, our security margin is at least equal to 37% of the rounds. Furthermore, while our Long Trail argument proves that 5 steps are sufficient to ensure that there are no single-trail differential and linear distinguishers, we do not expect this bound to be tight.

A high level view of the different components of Sparx-64/128 is provided in Figure 7.2, where branches have a width of 16 bits (except for the keys in the step structure). The linear layer $\lambda_2$ simply consists of a Feistel round using $\mathcal{L}$ as a Feistel function (Figure 7.2c). The general structure of a step of Sparx-64/128 is provided in Fig. 7.2b. The 128-bit permutation used in the key schedule has a simple definition summarized in Fig. 7.3, where the counter $r$ is initialized to 0. It corresponds to the pseudo-code given in Algorithm 7.2, where $(z)_L$ and $(z)_R$ are the 16-bit left and right halves of the 32-bit word $z$.



(a) $A_k^r$.

(b) Step structure.

(c) $\mathcal{L}$.

Figure 7.2: A high level view of Sparx-64/128.

The $\mathcal{L}$ function is borrowed from Noekeon [DPVAR00] and can be defined using 16- or 32-bit rotations. It is defined as a Lai-Massey structure mapping a 32-bit value $x||y$ to $x \oplus \big((x \oplus y) \lll 8\big)||y \oplus \big((x \oplus y) \lll 8\big)$. Alternatively, it can be seen as a mapping of a 32-bit value $z$ to $z \oplus (z \lll^{32} 8) \oplus (z \ggg^{32} 8)$ where the rotations are over 32 bits.

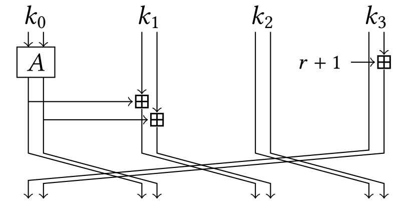


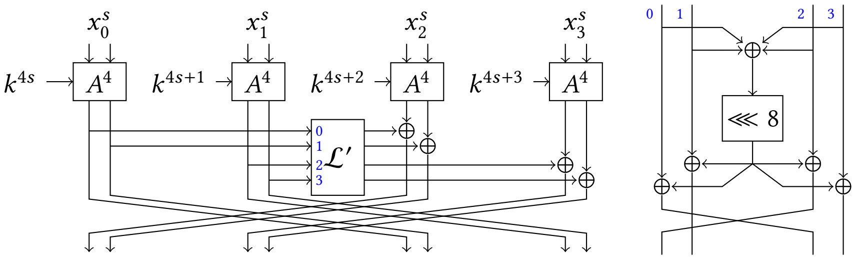
Figure 7.3: $K_4^{64}$ (used in Sparx-64/128).

```
$r \leftarrow r + 1$
$k_0 \leftarrow A(k_0)$
$(k_1)_L \leftarrow (k_1)_L + (k_0)_L \mod 2^{16}$
$(k_1)_R \leftarrow (k_1)_R + (k_0)_R \mod 2^{16}$
$(k_3)_R \leftarrow (k_3)_R + r \mod 2^{16}$
$k_0, k_1, k_2, k_3 \leftarrow k_3, k_0, k_1, k_2$
```

Algorithm 7.2: Pseudo-code of $K_4^{64}$

### 7.2.2 Sparx-128/128 and Sparx-128/256

For use cases in which a larger block size can be afforded, we provide Sparx instances with a 128-bit block size and 128- or 256-bit keys. They share an identical step structure which is fairly similar to Sparx-64/128. Indeed, the linear layer relies again on a Feistel function except that $\mathcal{L}$ is replaced by $\mathcal{L}'$, a permutation of $\{0,1\}^{64}$. Both Sparx-128/128 and Sparx-128/256 use 4 rounds per step but the first uses 8 steps while the last uses 10.



(a) Step structure.

(b) $\mathcal{L}'$.

Figure 7.4: The step structure of both Sparx-128/128 and Sparx-128/256.

The Feistel function $\mathcal{L}'$ can be defined as follows. Let $a||b||c||d$ be a 64-bit word where each word $a, \dots, d$ is 16-bit long. Let $t = (a \oplus b \oplus c \oplus d) \lll 8$. Then $\mathcal{L}'(a||b||c||d) = c \oplus t \;||\; b \oplus t \;||\; a \oplus t \;||\; d \oplus t$. This function can also be expressed using 32-bit rotations. Let $x||y$ be the concatenation of two 32-bit words and $\mathcal{L}'_b$ denote $\mathcal{L}'$ without its final branch swap. Let $t = \big((x \oplus y) \ggg^{32} 8\big) \oplus \big((x \oplus y) \lll^{32} 8\big)$, then $\mathcal{L}'_b(x||y) = x \oplus t||y \oplus t$. Alternatively, we can use $\mathcal{L}$ to compute $\mathcal{L}'_b$ as follows: $\mathcal{L}'_b(x||y) = y \oplus \mathcal{L}(x \oplus y)||x \oplus \mathcal{L}(x \oplus y)$.

The ciphers Sparx-128/128 and Sparx-128/256 differ only by their number of steps and by their key schedule. The key schedule of Sparx-128/128 needs a 128-bit permutation $K_4^{128}$ described in Fig. 7.5 and Algorithm 7.3 while Sparx-128/256 uses a 256-bit permutation $K_4^{256}$, which is presented in both Fig. 7.6 and Algorithm 7.4.

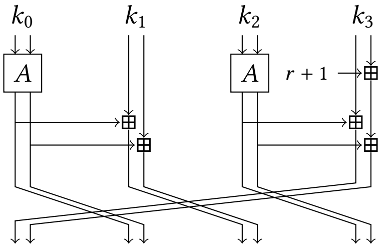


Figure 7.5: $K_4^{128}$ (used in Sparx-128/128).

```
$r \leftarrow r + 1$
$k_0 \leftarrow A(k_0)$
$(k_1)_L \leftarrow (k_1)_L + (k_0)_L \mod 2^{16}$
$(k_1)^R \leftarrow (k_1)_R + (k_0)_R \mod 2^{16}$
$k_2 \leftarrow A(k_2)$
$(k_3)_L \leftarrow (k_3)_L + (k_2)_L \mod 2^{16}$
$(k_3)_R \leftarrow (k_3)_R + (k_2)_R + r \mod 2^{16}$
$k_0, k_1, k_2, k_3 \leftarrow k_3, k_0, k_1, k_2$
```

Algorithm 7.3: Pseudo-code of $K_4^{128}$

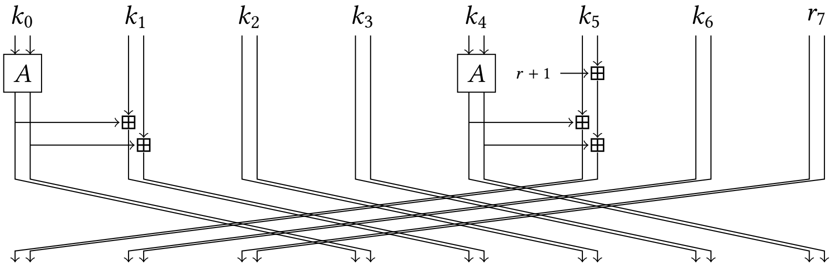


Figure 7.6: $K_8^{256}$ (used in Sparx-128/256).

**Algorithm 7.4** Sparx-128/256 key schedule permutation.

```
$r \leftarrow r + 1$
$k_0 \leftarrow A(k_0)$
$(k_1)_L \leftarrow (k_1)_L + (k_0)_L \mod 2^{16}$
$(k_1)_R \leftarrow (k_1)_R + (k_0)_R \mod 2^{16}$
$k_4 \leftarrow A(k_4)$
$(k_5)_L \leftarrow (k_5)_L + (k_4)_L \mod 2^{16}$
$(k_5)_R \leftarrow (k_5)_R + (k_4)_R + r \mod 2^{16}$
$k_0, k_1, k_2, k_3, k_4, k_5, k_6, k_7 \leftarrow k_5, k_6, k_7, k_0, k_1, k_2, k_3, k_4$
```

Test vectors for all instances of Sparx are provided in Table 7.2 as 16-bit words in hexadecimal notation.

## 7.3 Design Rationale

There are four main components in the different instances of Sparx. The arx-box is discussed in Section 7.3.1, the high level mixing layer in Section 7.3.2, the linear Feistel function in Section 7.3.3 and the key schedule in Section 7.3.4.

### 7.3.1 Choosing the arx-box

We chose the round function of Speckey/Speck32 over Marx-2 because of its superior implementation properties. Indeed, its smaller total number of operations means that a cipher using it needs to do fewer operations when implemented on a 16-bit

| | | | | |
|---|---|---|---|---|
| **Sparx-64/128** | | | | |
| key | 0011 2233 | 4455 6677 | 8899 aabb | ccdd eeff |
| plaintext | 0123 4567 | 89ab cdef | | |
| ciphertext | 2bbe f152 | 01f5 5f98 | | |
| **Sparx-128/128** | | | | |
| key | 0011 2233 | 4455 6677 | 8899 aabb | ccdd eeff |
| plaintext | 0123 4567 | 89ab cdef | fedc ba98 | 7654 3210 |
| ciphertext | 1cee 7540 | 7dbf 23d8 | e0ee 1597 | f428 52d8 |
| **Sparx-128/256** | | | | |
| key | 0011 2233 | 4455 6677 | 8899 aabb | ccdd eeff |
| | ffee ddcc | bbaa 9988 | 7766 5544 | 3322 1100 |
| plaintext | 0123 4567 | 89ab cdef | fedc ba98 | 7654 3210 |
| ciphertext | 3328 e637 | 14c7 6ce6 | 32d1 5a54 | e4b0 c820 |

Table 7.2: Test vectors for the different instances of Sparx.

platform. Ideally, we would have used an ARX-box with 32-bit operations but, at the time of writing, no such function has known differential and linear bounds (cf. Table 6.1) for sufficiently many rounds.

We chose to evaluate the iterations of the ARX-box over each branch rather than in parallel because such an order decreases the number of times each 32-bit branch must be loaded in CPU registers. This matters when the number of registers is too small to contain both the full key and the full internal state of the cipher and does not change anything if it is not the case as far as software implementations are concerned.

### 7.3.2 Mixing Layer, Number of Steps and Rounds per Step.

#### 7.3.2.1 Overall Design Strategy

Our main approach for choosing the mixing layer was exhaustive enumeration of all matrices suitable for our long trail bounding algorithm from Section 6.2.4.1 and selecting the final matrix according to various criteria, which we will discuss later.

For Sparx-64/128, there is only one linear layer fulfilling our design criteria: one corresponding to a Feistel round. For such a structure, we found that the best integral covers 4 steps (without the last linear layer) and that, with 3 rounds per step, the MEDCP and MELCC are bounded by $2^{-75}$ and $2^{-38}$. These quantities imply that no single trail differential or linear distinguisher exists for 5 or more steps of Sparx-64/128.

For Sparx instances with 128-bit block we implemented an exhaustive search on a large subset of all possible linear layers. After some filtering, we arrived at roughly 3000 matrices. For each matrix we ran our algorithm from Section 6.2.4.1 to obtain bounds on MEDCP and MELCC for different values of the number of rounds per step ($r_a$). We also ran the algorithm for searching integral characteristics described in Section 6.2.4.2.

Then, we analyzed the best matrices and found that there is a matrix which corresponds to a Feistel-like linear layer with the best differential/linear bound for $r_a = 4$. This choice also offered good compromise between other parameters, such as diffusion, strength of the ARX-box, simplicity and easiness/efficiency of implementation and symmetry between encryption and decryption. It also generalizes elegantly the linear layer of Sparx-64/128. We thus settled for this Feistel-like function.

For more details on the selection procedure and other interesting candidates for the linear layer, see [DPU+16b].

### 7.3.3 The Linear Feistel Functions

The linear layer obtained using the steps described above is only specified at a high level, it remains to define the linear Feistel functions $\mathcal{L}$ and $\mathcal{L}'$. The function $\mathcal{L}$ that we have chosen has been used in the Lai-Massey round constituting the linear layer of Noekeon [DPVAR00]. We reuse it here because it is cheap on lightweight processors as it only necessitates one rotation by 8 bits and 3 XORs. It also provides some diffusion as it has branching number 3. Its alternative representation using 32-bit rotations allows an optimized implementation on 32-bit processors.

The Feistel function $\mathcal{L}'$, used for a larger block size, is a generalization of $\mathcal{L}$: it also relies on a Lai-Massey structure as well as a rotation by 8 bits. The reason behind these choices are the same as before: efficiency and diffusion. Furthermore, $\mathcal{L}'$ must also provide diffusion between the branches. While this is achieved by the XORs, we further added a branch swap in the bits of highest weight. This ensures that if only one 32-bit branch is active at the input of $\mathcal{L}'$ then two branches are active in its output. Indeed, there are two possibilities: either the output of the rotation is non-zero, in which case it gets added to the other branch and spreads to the whole state through the branch swap. Otherwise, the output is equal to 0, which means that the two 16-bit branches constituting the non-zero 32-bit branch hold the same non-zero value. These will then be spread over the two output 32-bit branches by the branch swap. The permutation $\mathcal{L}'$ also breaks the 32-bit word structure, which can help prevent the spread of integral patterns.

### 7.3.4 Key Schedule

The key schedules of the different versions of Sparx have been designed using the following general guidelines.

First, we look at criteria related to the implementation. To limit code size, components from the round function of Sparx are re-used in the key-schedule itself. To accommodate cases where the memory requirements are particularly stringent, we allow an efficient on-the-fly computation of the key.

We also consider cryptographic criteria. For example, we need to ensure that the keys used within each chain of 3 or 4 arx-boxes are independent of one another. As we do not have enough entropy from the master key to generate truly independent round keys, we must also ensure that the round-keys are as different as possible from one another. This implies a fast mixing of the master key bits in the key schedule. Furthermore, in order to prevent slide attacks [BW99], we chose to have the round keys depend on the round index. Finally, since the subkeys are XOR-ed in the key state, we want to limit the presence of high probability differential pattern in the key update. Diffusion in the key state is thus provided by additions modulo $2^{16}$ rather than exclusive-or. While there may be high probability patterns for additive differences, these would be of little use because the key is added by an XOR to the state.

As with most engineering tasks, some of these requirements are at odds against each other. For example, it is impossible to provide extremely fast diffusion while also being extremely lightweight. Our designs are the most satisfying compromises we could find.

## 7.4 Security Analysis

### 7.4.1 Single Trail Differential/Linear Attack

By design and thanks to the Long Trail argument, we know that there is no differential or linear trail covering 5 steps (or more) with a useful probability for any instance of Sparx. Therefore, the 8 steps used by Sparx-64/128 and Sparx-128/128 and the 10 used by Sparx-128/256 are sufficient to ensure resilience against such attacks.

### 7.4.2 Attacks Exploiting a Slow Diffusion

We consider several attacks in this category, namely impossible and truncated differential attacks, meet-in-the middle attacks as well as integral attacks.

When we chose the linear layers, we ensured that they prevented integral attacks based on the division-property. This implies that they provide good diffusion. Furthermore, the Feistel structure of the linear layer makes it easy to analyze and increases our confidence in our designs. In the case of 128-bit block sizes, the Feistel function $\mathcal{L}'$ has branching number 3 in the sense that if only one 32-bit branch is active then the two output branches are active. This prevents attacks trying to exploit patterns at the branch level. Finally, this Feistel function also breaks the 32-bit word structure through a 16-bit branch swap which frustrates the propagation of integral characteristics.

Meet-in-the-middle attacks are further hindered by the large number of key additions. This liberal use of the key material also makes it harder for an attacker to guess parts of it to add rounds at the top or at the bottom of, say, a differential distinguisher.

### 7.4.3 Best Attacks Against Sparx

The best attacks we could find are integral attacks based on Todo's division property. For 22-round Sparx-128/128, we can recover the key in time $2^{105}$ using $2^{102}$ chosen plaintexts and $2^{72}$ blocks of memory. We attack 24-round Sparx-128/256 in time $2^{233}$, using $2^{104}$ chosen plaintexts and $2^{202}$ blocks of memory. Finally, the attack against Sparx-64/128 covers 15/24 rounds and recovers the key in time $2^{101}$ using $2^{37}$ chosen plaintexts and $2^{64}$ blocks of memory. The integral distinguisher we use to attack Sparx128/k is described in Section 7.4.3.1 (p. 124). Then, the attacks against Sparx128/k are described in Section 7.4.3.2 (p. 126) and the one targeting Sparx64/128 is in Section 7.4.3.3 (p. 127).

#### 7.4.3.1 An Integral Distinguisher for 128-bit Blocks

Consider an instance of Sparx operating on 128-bit blocks. Using Todo's division property, we found that if we fix the left-most 32-bit word of the plaintext and let the other 3 take all possible $2^{96}$ values, then the output of the two right-most word at the end of the last S-Box layer of the 5th step are balanced. This pattern is destroyed by the linear layer of the 5th step.

However, because half of the $A$ function merely undergoes linear transformations, and because the lsb of modular addition is a linear function, it is possible to

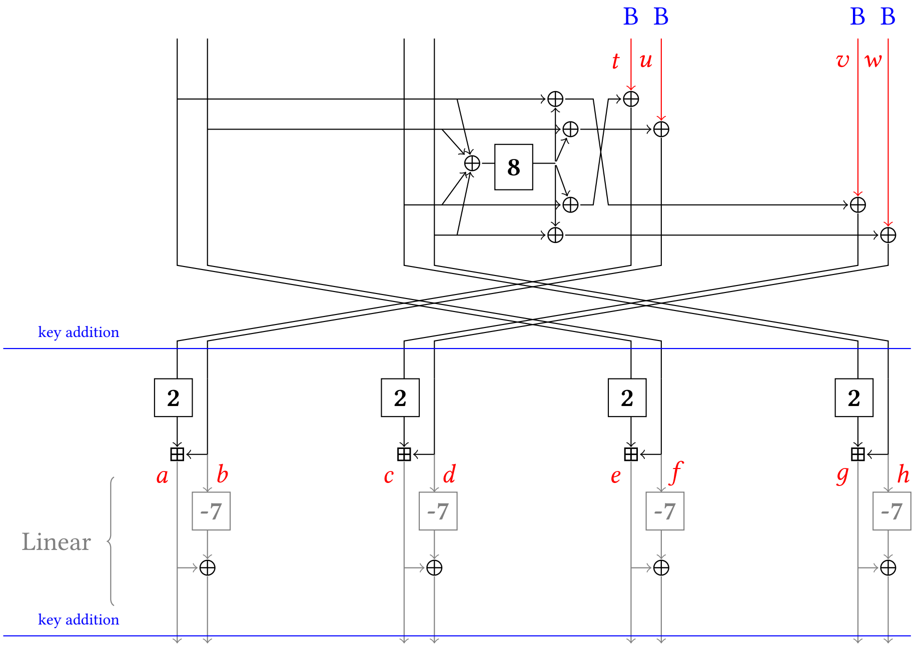


Figure 7.7: Principle of the integral attack against Sparx instances operating on 128 bits. Bold numbers denote bit rotations to the left. The letter B denotes balanced 16-bit words at the end of the S-Box layer of the fifth step.

extend this distinguisher by one more round. First, the last operations of $A$ can be inverted for free because they are linear, as summarized in Figure 7.7, to obtain new 16-bit values $a, b, \dots, h$, as defined in the same picture where we also define the quantities $t, u, v$ and $w$.

Consider a structure of size $2^{96}$ obtained by encrypting $2^{96}$ plaintexts where the left-most 32-bit word is fixed and the other take all possible values. We use $z[j]$ to denote the $j$-th bit of a 16-bit word $z$ where the ordering is from LSB to MSB. Let $i$ be the index of the ciphertexts in our structure. Then the following equation holds with probability 1:

$$\bigoplus_{i=0}^{2^{96}-1} \Big( (e^i[0] \oplus f^i[0]) \oplus f^i[14] \oplus (g^i[0] \oplus h^i[0]) \oplus h^i[14] \Big) \oplus (b^i[10] \oplus f^i[10]) \;=\; 0. \quad (7.1)$$

Indeed, the sum of these values and the corresponding key bits yields the value of $u^i[10]$ which sums to 0 over the structure, regardless of the key bits. If we look at $w^i[10]$ instead of $u^i[10]$, we can derive another equation:

$$\bigoplus_{i=0}^{2^{96}-1} \Big( (e^i[0] \oplus f^i[0]) \oplus f^i[14] \oplus (g^i[0] \oplus h^i[0]) \oplus h^i[14] \Big) \oplus (d^i[10] \oplus h^i[10]) \;=\; 0. \quad (7.2)$$

Equations (7.1) and (7.2) hold for any key and for any such structure of $2^{96}$ ciphertexts. They can be used to attack both Sparx-128/128 and Sparx-128/256.

#### 7.4.3.2 Integral Attack Against 22/(24)-round Sparx-128/128(256)

We use this distinguisher to attack 22-round Sparx-128/128. It is naturally extended to attack 6 steps (24 rounds) of Sparx-128/256 by guessing keys for two more rounds. We will use the partial sums technique introduced by Ferguson *et al.*in [FKL+01]. The idea of the attack is to encrypt several structures of plaintexts and then by using partial sums to split the key guessing work into two parts and finally to filter out wrong keys. An outline of the attack follows.

1. Encrypt 64 structures of $2^{96}$ plaintexts such that the left-most word is fixed inside each structure and the other three take all possible values.

2. For each structure and each word position:

   a) Store all word values which occur an odd number of times in this position in all ciphertexts from the structure. On average there will be $2^{31}$ such values per structure per position.

3. Initialize a hash table indexed by 128-bit blocks.

4. For all $2^{64}$ possible keys $K_0, K_1$:

   a) For each structure, decrypt one round of Speckey on all stored words for positions 0 and 1 using keys $K_0$ and $K_1$ respectively. Compute the contribution of decrypted bits to sums from Equations (7.1) and (7.2). Such contributions form a 128-bit string $s$ (two bits per structure).

   b) Add $K_0||K_1$ to the hash table with index $s$.

5. For all $2^{64}$ possible keys $K_2, K_3$:

   a) Decrypt one round of Speckey on all stored words for positions 2 and 3 using keys $K_2, K_3$ and compute the contributions similarly to the previous step. Since XOR of the contributions from left and right halves must be equal to zero for each structure, the contributions must be equal.

   b) Check if the 128-bit contribution string is in the hash table. If it is there, get the corresponding key $K_0||K_1$ and save $K_0||K_1||K_2||K_3$ as a key candidate.

6. On average we will obtain 1 key candidate for the last round. Then we can decrypt the last round and run another attack or we can exploit the key schedule and reconstruct all round keys.

Step 2 requires $64 \cdot 2^{96} \cdot 4 = 2^{104}$ simple operations. Complexity of steps 4 and 5 is equal to $2^{64} \cdot 64 \cdot 2 \cdot 2^{31} = 2^{102}$ Speckey round decryptions. Therefore full attack complexity can be bounded by $2^{105}$ operations. The data complexity is equal to $64 \cdot 2^{96} = 2^{102}$. To store the hashtable we need around $2^{64} \cdot (2 \cdot 64 + 64) < 2^{72}$ memory blocks.

The 24-round attack on Sparx-128/256 is very similar. We encrypt 192 structures and guess keys required to decrypt three Speckey rounds instead of one. The complexity then is dominated by steps 4 and 5 and is equal to $2 \cdot 2^{192} \cdot 192 \cdot 2 \cdot 2^{31} < 2^{233}$ operations. The data complexity is $192 \cdot 2^{96} < 2^{104}$ chosen plaintexts. The memory complexity is around $2^{192} \cdot (2 \cdot 192 + 192) < 2^{202}$ blocks.

We note that by exploiting the key schedule we can reduce the complexities by not guessing a repeating key material. However, we did not manage to attack more rounds.

#### 7.4.3.3 Integral Attack Against 15-round Sparx-64/128

If we encrypt a structure built by setting the left side to a constant and letting the right side take all possible values using Sparx-64/128 then, with probability 1, the right side after 3 steps (minus the last linear layer) must take all $2^{32}$ possible values. Using multi-set terminology, a permutation on the right side of the input becomes a permutation on the right side of the output. We can use this property as a distinguisher for 12 rounds. It turns out that the division property does not yield any stronger result in this case.

As for the attacks against 128-bit block variants, we can append one round after the end of the 4th step and derive a linear equation linking the lsb of the modular addition on the right with 2 other bits from the right hand side and 1-bit from the left hand side. The sum of these bits over the ciphertexts in a structure must be equal to 0.

We append two rounds at the end of this distinguisher to attack 5 steps of this block cipher. To do this, we repeat the following procedure several times.

1. Encrypt a structure of $2^{32}$ plaintexts and invert the last linear linear.

2. For all 64-bit key $k_L$, partially decrypt 2 rounds for the left-hand side and store $k_L$ in a list indexed by the xor of the 3 interesting bits from this side.

3. For all 64-bit key $k_R$, partially decrypt 2 rounds for the right-hand side and store $k_R$ in list indexed by the unique interesting bit from this side.

Each time we repeat this procedure, we re-distribute the key guesses in list indexed by the concatenation of the indices in the successive structures. For example, if $k^R$ is in list 1 for the first structure and list 0 for the second one, we place it in a general list with index $01 = 1$. If it is placed in list 1 by the third structure, we move it to the general list $101 = 5$, and so on.

If we repeat this procedure $u$ times, we obtain $2^{128-u}$ key candidates of 128 bits from which we deduce full key candidates that are tested with trial encryptions. We thus have a time/data tradeoff: with only one structure, we can only recover 1 bit from the key. Conversely, with all 128 structures, we recover the full key.

The treatment of each structure requires $2^{32+64}$ operations and the storage of $2^{32}$ ciphertexts. At all times, we must also store all $2^{64}$ candidates for both $k_L$ and $k_R$. The complexity of the final brute-force is $2^{128-u}$. If $u \geq 32$, the bottle-neck in terms of time is the treatment of all $u$ structures.

In the end, we attack 5 steps (i.e. 15 rounds) of Sparx-64/128 using about $2^{64}$ blocks of memory, $2^{32} \times u$ chosen plaintexts and $u2^{96} = 2^{128-u}$ operations. In particular, if we use 32 structure, we can break this cipher in time $2^{101}$ using $2^{37}$ chosen plaintexts.

## 7.5 Software Implementation

In [DLCK+15], Dinu *et al.* presented the FELICS[2] framework. It is a benchmarking tool providing a unified method for benchmarking lightweight block and stream ciphers on several micro-controllers: the 8-bit Atmel ATmega128, the 16-bit TI MSP430, and the 32-bit ARM Cortex-M3.

The information extracted using this framework allowed to highlight some properties of SPARX. These are detailed in Section 7.5.1. Our cipher is compared with other lightweight ciphers in Section 7.5.2.

### 7.5.1 Implementation Properties

An encryption using either version of SPARX has the same overall structure consisting of three interleaved loops. At the highest level there is a loop over all steps. During each step, there is a loop over all branches. Finally, over each branch, there is a loop over rounds per steps.

By (partially) rolling or unrolling these loops, we obtain very different implementations, as shown in Table 7.3. This allows SPARX to cover a large subset of the performance space: it can be small, using only 166 bytes to be implemented on MSP, or it can be fast so as to use only 932 cycles to encrypt one block on ARM.

Our choice to leave the final linear layer despite the fact that it plays no role as far as security is concerned was motivated by this analysis. Indeed, it turns out that removing it can marginally speed up a fully unrolled implementation and is actually harmful in other cases. This behavior is caused by the extra logic needed to implement a special last round.

| | | AVR | | | MSP | | | ARM | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Version | Implementation | Time | Code | RAM | Time | Code | RAM | Time | Code | RAM |
| | | [cyc.] | [B] | [B] | [cyc.] | [B] | [B] | [cyc.] | [B] | [B] |
| SPARX-64/128 | 1-step rolled | 1789 | **248** | 2 | 1088 | **166** | 14 | 1370 | **176** | 28 |
| | 1-step unrolled | 1641 | 424 | **1** | 907 | 250 | 12 | 1100 | 348 | **24** |
| | 2-steps rolled | 1677 | 356 | 2 | 1034 | 232 | 10 | 1331 | 304 | 28 |
| | 2-steps unrolled | **1529** | 712 | **1** | **853** | 404 | **8** | **932** | 644 | **24** |
| SPARX-128/128 | 1-step rolled | 4553 | **504** | 11 | 2809 | **300** | 26 | 3463 | **348** | 44 |
| | 1-step unrolled | 4165 | 1052 | **10** | 2353 | 584 | 24 | 2784 | 884 | 40 |
| | 2-steps rolled | 4345 | 720 | 11 | 2593 | 432 | 18 | 3399 | 620 | 40 |
| | 2-steps unrolled | **3957** | 1820 | **10** | **2157** | 1004 | **16** | **2377** | 1692 | **36** |

Table 7.3: Different trade-offs between the execution time and code size for the encryption of a block using SPARX-64/128 and SPARX-128/128.

[2] FELICS stands for Fair Evaluation of LIghtweight Cryptographic Systems. Up to date results are provided on the web page of the framework: https://www.cryptolux.org/index.php/FELICS.

### 7.5.2 Comparison with Other Ciphers

Felics takes as input several implementations of a block cipher, written in C or in assembly, and returns their RAM consumption, the number of cycles they need to and their code size. These are extracted in different settings emulating real-life scenarios such as the encryption in CTR mode of 128 bits of data.

The final result is expressed using a single quantity called *Figure Of Merit (FOM)* which is better when lower. It aggregates the time-, RAM- and code-efficiency of the different implementations of a given algorithm.

As we can see in Table 7.4, Sparx ranks favorably among micro-controllers-oriented lightweight block ciphers. Furthermore, if we restrict the comparison to algorithms designed to provide provable bounds for differential and linear probabilities, Sparx and Rectangle are the best two algorithms: among the top ten algorithms, only Rectangle, Sparx, the Aes and Fantomas have such bounds.

| Rank | Cipher | Reference | Block size | Key size | Scenario 1 FOM |
|---|---|---|---|---|---|
| 1 | Speck | [BSS+13] | 64 | 128 | 5.0 |
| 2 | Chaskey-LTS | [MMH+14] | 128 | 128 | 5.0 |
| 3 | Simon | [BSS+13] | 64 | 128 | 6.9 |
| 4 | Rectangle | [ZBL+15] | 64 | 128 | 7.8 |
| 5 | Lea | [HLK+14] | 128 | 128 | 8.0 |
| **6** | **Sparx** | This chapter | **64** | **128** | **8.6** |
| **7** | **Sparx** | This chapter | **128** | **128** | **12.9** |
| 8 | Hight | [HSH+06] | 64 | 128 | 14.1 |
| 9 | Aes | [DR02] | 128 | 128 | 15.3 |
| 10 | Fantomas | [GLSV15] | 128 | 128 | 17.2 |

Table 7.4: Top 10 best implementations in Scenario 1 (encryption key schedule + encryption and decryption of 128 bytes of data using CBC mode) ranked by the Figure of Merit (FOM) defined in Felics.

The FOM of Sparx indicates that it is, overall, suitable for use on micro-controllers. We can also give more specific figures. Of all the algorithms implemented in Felics, it can use the least amount of RAM on both AVR and MSP. It is also one of the fastest algorithm on MSP, which should come as no surprise given that it operates on 16-bit words.

Part II

# On S-Box Reverse-Engineering

# Contents of this Part

Chapter 8

# Definitions and Literature Survey

A symmetric primitive has to be non-linear. There are two different strategies for achieving a high non-linearity. The first consists of using modular addition, usually within the framework of the ARX design paradigm which stands for Addition, Rotation, XOR. For example, the Sparx family of lightweight block ciphers presented in Chapter 7 (p. 117) is built in this fashion. The other method consists of using so-called *S-Boxes*.

**Definition 8.0.1** (S-Box)**.** *An* S-Box *is a function mapping a small number of bits $m$ to $n$ bits. The input length $m$ must be small. Thus, they are usually specified through their look-up tables.*

S-Boxes have many advantages. First of all, as they are usually implemented through table look-ups, they are quite fast in software.[1] Should side-channel resistance be needed, they can also be implemented/specified using a small electronic circuit or a bit-sliced implementation. Finally, due to their small size, it is possible to precisely evaluate their cryptographic properties. In fact, both the *wide trail strategy* [DR01] which was used to design the Aes [DR02] and the *long trail strategy* which we proposed in [DPU+16a] (see Chapters 6 and 7) are methods which allow the designer of a block cipher to prove the security of their algorithm against basic differential [BS91] and linear [Mat94] attacks using the properties of the S-Boxes involved. Therefore, the study of the cryptographic properties of these functions is of paramount importance.

In this chapter, I first introduce the basic mathematical concepts needed to describe and analyze S-Boxes in Section 8.1. The cryptographic properties of S-Boxes are presented in Section 8.2 (p. 139), namely their differential, linear and algebraic properties. Then, I present an extensive analysis of the literature in terms of S-Box design in Section 8.3 (p. 146): which design strategies have been used by cryptographers in practice, and why where those chosen?

## 8.1 Representation and Basic Concepts

Recall that $\{0, 1\} = \mathbb{F}_2$. The set $\mathbb{F}_2^n$ consisting of elements $x = (x_0, ..., x_{n-1})$ can be given different structures:

[1] This speed has however a price: a table look-up is an operation which leaks some information about its input, meaning that they are particularly vulnerable to side-channel attacks. As discussed in Chapter 7 (p. 117), this is one of the reasons why the ARX paradigm is preferable in some cases.

- it can be interpreted as a finite field of size $2^n$ defined as $\mathbb{F}_{2^n} = \mathbb{F}_2[x]/p(x)$ for some irreducible polynomial $p$ of degree $n$, or
- alternatively, $\mathbb{F}_2^n$ can be viewed as $\mathbb{Z}/2^n\mathbb{Z}$, in which case $x \in \mathbb{F}_2^n$ is identified with $\overline{x} = \sum_{i=0}^{n-1} x_i 2^i$.

We usually represent S-Boxes using the hexadecimal representation of their look-up tables (LUT), either as lists if they are small or as tables if they are bigger. For instance, the 3-bit permutation $z : x \mapsto (3 \times x + 1) \mod 8$ is represented as $z = [1, 4, 7, 2, 5, 0, 3, 6]$. The 8-bit S-Box $S_{\text{AES}}$ of the AES is represented in Table 8.1. As we can see, $S_{\text{AES}}$(0x7a) = 0xda.

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

Table 8.1: The S-Box of the AES.

There are different equivalence classes partitioning the space of the S-Boxes mapping $\mathbb{F}_2^m$ to $\mathbb{F}_2^n$. Linear- and affine-equivalence are described below. Another more complex equivalence called CCZ-equivalence is introduced in Chapter 14 (p. 267).

**Definition 8.1.1** (Linear Equivalence)**.** *Let $f : \mathbb{F}_2^m \rightarrow \mathbb{F}_2^n$ and $g : \mathbb{F}_2^m \rightarrow \mathbb{F}_2^n$ be two S-Boxes. They are* linear-equivalent *if and only if there exists two linear permutations $\eta$ and $\mu$ such that $g = \eta \circ f \circ \mu$.*

**Definition 8.1.2** (Affine Equivalence)**.** *Let $f : \mathbb{F}_2^m \rightarrow \mathbb{F}_2^n$ and $g\mathbb{F}_2^m \rightarrow \mathbb{F}_2^n$ be two S-Boxes. They are* affine-equivalent *if and only if there exists two affine permutations $\eta$ and $\mu$ such that $g = \eta \circ f \circ \mu$.*

We call *affine whitening* the action of composing a function with two affine permutations in order to hide its structure.

The *scalar product* of the two vectors $u = (u_{m-1}, ..., u_0)$ and $v = (v_{m-1}, ..., v_0)$ is denoted $u \cdot v$ and is equal to $\bigoplus_{i=0}^{m-1} u_i \wedge v_i$, where $\wedge$ is the logical AND. Using this scalar product, I can introduce the distinction between *coordinate* and *component* of an S-Box.

**Definition 8.1.3** (Coordinates and Components). *An S-Box $s$ with $n$ output bits has $n$* coordinates*, denoted $S_i$, where $s_i(x)$ is the $i$-th output bit of $s(x)$. It also has $2^n - 1$* components *$x \mapsto a \cdot s(x)$, one for each non-zero $a$ in $\mathbb{F}_2^n$. In particular, $s_i(x) = e_i \cdot s(x)$, where $\{e_i\}_{i<n}$ is the canonical basis of $\mathbb{F}_2^n$.*

Finally, I recall the well-known concept of *balanced function.*

**Definition 8.1.4** (Balanced function). *A function $f : \mathbb{F}_2^m \rightarrow \mathbb{F}_2^n$ is* balanced *if and only if all preimage sets $\{x \in \mathbb{F}_2^m, f(x) = y\}$ have the same size for all $y$ of $\mathbb{F}_2^n$.*

## 8.2 Cryptographic Properties of S-Boxes

In this Section, I summarize some of the properties of an S-Box that are relevant in cryptography. Its content is basically folklore nowadays: these notions are well understood and a significant amount of papers have been devoted to these topics, either directly or through their connection with e.g. coding theory. The differential, linear and algebraic properties of S-Boxes are introduced in Sections 8.2.1, 8.2.2 and 8.2.3 respectively.

### 8.2.1 Differential Properties

The differential properties of an S-Box are fully described by its *Difference Distribution Table.*

**Definition 8.2.1** (Difference Distribution Table). *The* Difference Distribution Table (DDT) *of an S-Box $s : \mathbb{F}_2^m \rightarrow \mathbb{F}_2^n$ is the $2^m \times 2^n$ table* DDT *where the entry* DDT$[i, j]$ *is equal to the number of solutions $x$ of*

$$s(x \oplus i) \oplus s(x) = j.$$

For example, the 3-bit S-Box $r = [6, 0, 7, 3, 2, 5, 1, 4]$ obtained with a Knuth shuffle has the DDT shown in Table 8.2.

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| 0 | 8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 2 | 2 | 2 | 2 |
| 2 | 0 | 4 | 0 | 4 | 0 | 0 | 0 | 0 |
| 3 | 0 | 0 | 0 | 0 | 2 | 2 | 2 | 2 |
| 4 | 0 | 0 | 0 | 0 | 2 | 2 | 2 | 2 |
| 5 | 0 | 0 | 4 | 4 | 0 | 0 | 0 | 0 |
| 6 | 0 | 0 | 0 | 0 | 2 | 2 | 2 | 2 |
| 7 | 0 | 4 | 4 | 0 | 0 | 0 | 0 | 0 |

Table 8.2: The DDT of $r = [6, 0, 7, 3, 2, 5, 1, 4]$.

Several distinct S-Boxes can have the same DDT. For example, $s : \mathbb{F}_2^m \rightarrow \mathbb{F}_2^n$ and $s_{a,b} : \mathbb{F}_2^m \rightarrow \mathbb{F}_2^n$ where $s_{a,b}(x) = s(x \oplus b) \oplus a$, $a \in \mathbb{F}_2^n$ and $b \in \mathbb{F}_2^m$ have the same DDT. To the best of my knowledge, there is no efficient algorithm allowing to recover an S-Box from its DDT.

The maximum value of DDT$[i, j]$ for $i \neq 0$ is called differential uniformity of $s$ [Nyb94] and is denoted $\Delta_s$, e.g. $\Delta_r = 4$. The lower it is, the better an S-Box is at preventing differential attacks. Indeed, the highest probability for a differential pattern to go through this S-Box successfully is equal to $\Delta_s/2^n$ which is lower when $\Delta_s$ is. The wide trail argument relies on this bound on the differential probability at the S-Box level.

The optimal value for $\Delta_s$ is 2. A function $f$ such that $\Delta_s = 2$ is called *Almost Perfect Non-linear (APN)* but, unfortunately, the very existence of APN permutation is not really understood. On finite fields of odd degree, the cube function $x \mapsto x^3$ and the inverse function $x \mapsto x^{2^n-2}$ are for example known to be APN. However, on finite fields of even degree, the mere existence of APN permutations is an open problem. The only exception is for $n = 6$: Dillon *et al.* found an APN permutation in $\mathbb{F}_{2^6}$. Chapter 14 (p. 267) is devoted to an analysis of this permutation.

The expected distribution of the coefficients in the DDT of a random S-Box is known. In fact, it can be used as a criteria to figure out if an S-Box may have been generated at random or not, as explained in Chapter 9 (p. 159).

For a given S-Box, the maximum value may not be sufficient to assess the security of a cipher using it because of the *differential effect*. This phenomenon corresponds to the clustering of several distinct differential trails covering several rounds which share the same input and output differences. For example, we showed in Section 5.4 (p. 100) that the block ciphers Twine, LBlock and LBlock-s exhibit such an effect. The probability of a single trail covering several rounds of one of these ciphers is bounded using the number of active S-Boxes and the differential uniformity of the S-Boxes. However, the probability of the differential itself — thus taking into account all differential trail with the same input and output difference — is higher than this bound. The exact probability depends on the distribution of the coefficients in the DDT even if the differential uniformity is the same.

Finite field monomials are particularly easy to study because the rows of the DDT of such functions all share the same coefficient distribution. This insight was first presented in [BCC10]. The coefficients in line $a$ correspond to the number of solution $x$ to the following equation, for each value of $b$:

$$(x+a)^e + x^e = b \text{ is equivalent to } \left(\frac{x}{a}+1\right) + \left(\frac{x}{a}\right)^e = \frac{b}{a^e}.$$

As we can see, for $a \neq 0$, this number does not depend on $a$. In particular, the DDT of the multiplicative inversion over fields of even degree has lines with one 4, $2^{n-1}-2$ occurrences of 2 and $2^{n-1}+1$ zeroes. Over fields of odd degree, this function is APN so that all lines with non-zero indices contain $2^{n-1}$ zeroes and an equal amount of 2.

To capture the idea that the inverse function over fields of even degree is in some sense “almost” APN, the concept of *local-uniformity* was introduced in [BCC11] for the APN case and generalized in [BP14b]: a monomial such that $\text{DDT}[1,b] \leq \delta$ for $b \neq 0,1$ is said to be *locally differentially $\delta$-uniform.* In particular, if $\delta = 2$, the monomial is *locally-APN*, so that the inverse function in $\mathbb{F}_{2^8}$ is locally-APN. I used the fact that the inverse is locally-APN to efficiently reverse-engineer the S-Box of the German block cipher Chiasmus in Section 9.3 (p. 177).

Below, we list the effect of some simple operations on the DDT.

**Lemma 8.2.1.** *The DDT of the inverse of a permutation $s$ is the transpose of the DDT of $s$. In other words, if $D$ is the DDT of $s$ and $D'$ that of $s^{-1}$, then $D[i,j] = D'[j,i]$ for all $i,j$.*

**Lemma 8.2.2.** *Let $f$ be an $n$-bit permutation and let $D$ be its DDT. Let $g = \eta \circ f \circ \mu$ where $\eta$ and $\mu$ are affine permutations and let $D'$ be its DDT. Then $D'$ and $D$ are related* *as follows:*

$$D'[a,b] = D[\mu(a), \eta^{-1}(b)].$$

### 8.2.2 Linear Properties

Just like the differential properties, the linear properties of an S-Box are fully described by its *Linear Approximation Table.*

**Definition 8.2.2** (Linear Approximation Table). *The* Linear Approximation Table (LAT) *of an S-Box* $s : \mathbb{F}_2^m \to \mathbb{F}_2^n$ *is the* $2^m \times 2^n$ *table* LAT *where* $\mathrm{LAT}[a,b] = (1/2)W_{a,b}$*. The coefficients* $W_{a,b}$ *are the* Walsh coefficients *of* $s$ *and are given by*

$$W_{a,b} = \sum_{x \in \mathbb{F}_2^m} (-1)^{a \cdot x \oplus b \cdot s(x)},$$

*where* $u \cdot v$ *denotes the scalar product of the two vectors* $u = (u_{m-1}, \ldots, u_0)$ *and* $v = (v_{m-1}, \ldots, v_0)$*.*

For example, the 3-bit S-Box $r = [6,0,7,3,2,5,1,4]$ used previously has the LAT shown in Table 8.3.

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| 0 | 4 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | -2 | -2 | 0 | 0 | 2 | -2 |
| 2 | 0 | 2 | 0 | -2 | 0 | 2 | 0 | 2 |
| 3 | 0 | 2 | -2 | 0 | 0 | -2 | -2 | 0 |
| 4 | 0 | 0 | -2 | 2 | 0 | 0 | 2 | 2 |
| 5 | 0 | 0 | 0 | 0 | -4 | 0 | 0 | 0 |
| 6 | 0 | 2 | 2 | 0 | 0 | -2 | 2 | 0 |
| 7 | 0 | -2 | 0 | -2 | 0 | -2 | 0 | 2 |

Table 8.3: The LAT of $r = [6,0,7,3,2,5,1,4]$.

Unlike the DDT, an LAT is uniquely associated to its S-Box. In fact, it is possible to recover the S-Box from its LAT using the following lemma.

**Lemma 8.2.3.** *Let* $s_i$ *be the i-th coordinate of* $s : \mathbb{F}_2^m \to \mathbb{F}_2^n$ *and let* $\mathrm{LAT}[a,b]$ *be the* LAT *coefficients of* $s$*. Then* $s_i$ *can be recovered using:*

$$s_i(x) = \frac{1}{2} - \frac{1}{2^m} \sum_{a \in \mathbb{F}_2^m} \mathrm{LAT}[a, 2^i](-1)^{a \cdot x}. \tag{8.1}$$

*Proof.* This formula is easily derived using the relationship between the LAT and the Fourier transform. It is also straightforward to check it:

$$\begin{aligned} \sum_{a \in \mathbb{F}_2^m} \mathrm{LAT}[a, 2^i](-1)^{a \cdot x} &= \frac{1}{2} \sum_{a \in \mathbb{F}_2^m} \sum_{y \in \mathbb{F}_2^m} (-1)^{a \cdot y \,\oplus\, (2^i) \cdot s(y) \,\oplus\, a \cdot x} \\ &= \frac{1}{2} \sum_{z \in \mathbb{F}_2^m} (-1)^{s_i(z \oplus x)} \sum_{a \in \mathbb{F}_2^m} (-1)^{a \cdot z}, \end{aligned}$$

where $z = x \oplus y$. The sum $\sum_{a \in \mathbb{F}_2^m} (-1)^{a \cdot z}$ is always equal to 0 unless $z = 0$, in which case it equals $2^m$. The right hand side of Equation (8.1) is thus equal to $(1 - (-1)^{s_i(x \oplus 0)})/2$ which is indeed equal to $s_i(x)$. □

In a linear attack, the data complexity depends on the square of the LAT coefficients of the S-Box present in the linear trail. The higher these coefficients, the more efficient the attack. Thus, the main design criteria for an S-Box in terms of LAT is to have a low maximum coefficient absolute value. This is measured using the *linearity*: the linearity $\mathcal{L}_s$ of an S-Box $s : \mathbb{F}_2^m \to \mathbb{F}_2^n$ is equal to $2 \times \max_{a \neq 0, b} (|\text{LAT}[a,b]|)$. Alternatively, authors sometimes use the *non-linearity* which, for an S-Box $s : \mathbb{F}_2^m \to \mathbb{F}_2^n$, is defined as $\mathcal{NL}(s) = 2^{m-1} - \mathcal{L}_s/2 = 2^{m-1} - \max_{a \neq 0, b} (|\text{LAT}[a,b]|)$. The non-linearity measures the distance between the S-Box and the set of all affine functions. Therefore, the higher it is, the better the S-Box.

Much like for the DDT, only the maximum coefficient of the LAT of the S-Box is used to prove that a block cipher is secure against linear attacks. However, e.g. in a multi-dimensional linear attack [BDQ04, KR94] or in zero-correlation attacks [BR11, BW12], we need to look at other coefficients.

The similarities between the roles of the LAT in linear attacks and its generalizations and that of the DDT in differential attacks and its generalizations is well studied, see for instance [BN13, BN14]. In fact, the following relation between the DDT and the LAT was observed much earlier by Chabaud and Vaudenay in [CV95]: the DDT and the LAT of an S-Box $s : \mathbb{F}_2^m \to \mathbb{F}_2^n$ are such that

$$\text{DDT}[\delta, \Delta] = \frac{1}{2^{n+m-2}} \sum_{a \in \mathbb{F}_2^m} \sum_{b \in \mathbb{F}_2^n} (\text{LAT}[a,b])^2 (-1)^{a \cdot \delta \oplus b \cdot \Delta}.$$

The LAT of such an S-Box $s : \mathbb{F}_2^m \to \mathbb{F}_2^n$ must also satisfy the following equations. Recall that its coefficients are $\text{LAT}[a,b] = W_{a,b}/2$.

- **Parseval's equality:** for all $b$, $\sum_{a \in \mathbb{F}_2^m} W_{a,b}^2 = 2^{2m}$.
- **Linearity bound :** $\mathcal{L}_s \geq 2^{m/2}$ — if the equality holds, the function is *bent*.
- If $s$ is a permutation of $\mathbb{F}_2^n$, then the LAT of $f^{-1}$ is the transpose of that of $f$.

The coefficients in an LAT are merely shuffled by affine-whitening, as formalized in the following lemma.

**Lemma 8.2.4.** *Let $f$ be an $n$-bit permutation and let $\mathcal{L}$ be its* LAT. *Let $\mathcal{L}'$ be a table defined by $\mathcal{L}'[u,v] = \mathcal{L}[\mu(u), \eta(v)]$ for some linear permutations $\mu$ and $\eta$. Then the function $f'$ has* LAT *$\mathcal{L}'$, where*

$$f' = \eta^t \circ f \circ (\mu^{-1})^t.$$

### 8.2.3 Algebraic Properties

S-Boxes can be represented in different ways. So far, we have only used their look-up tables but other representations exist. Furthermore, those have some relevance when assessing the security provided by the S-Box.

**Definition 8.2.3** (Algebraic Normal Form)**.** *The* Algebraic Normal Form (ANF) *of a Boolean function $f : \mathbb{F}_2^m \to \mathbb{F}_2$ is*

$$f(x) = \bigoplus_{u \in \mathbb{F}_2^m} a_u^s x^u,$$

*where $a_u^f \in \mathbb{F}_2$ and $x^u$ is defined as*

$$x^u = \prod_{i=0}^{m-1} x_i^{u_i}.$$

*The* ANF *coefficients $a_u^f$ can be found using the* Möbius transform*:*

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

*where $a \preccurlyeq b$ if $a_i \leq b_i$ for every $i$. For an S-Box $s : \mathbb{F}_2^m \rightarrow \mathbb{F}_2^n$, the* ANF *consists of the* ANF *of each of its coordinates $s_i$ for $i \in \{0, \ldots, n-1\}$.*

For example, the ANF of $r = [6,0,7,3,2,5,1,4]$ is

$$\text{ANF}(r) = \begin{cases} r_0 = x_0 x_2 + x_1 \\ r_1 = x_0 x_1 + x_1 x_2 + x_0 + 1 \\ r_2 = x_0 + x_2 + 1 \,, \end{cases}$$

**Definition 8.2.4** (Algebraic degree)**.** *The* Algebraic Degree *of a Boolean function is the maximum number of variables in a term of its* ANF*. For an S-Box $s : \mathbb{F}_2^m \rightarrow \mathbb{F}_2^n$, it is the maximum algebraic degree of its coordinates.*

For example, the algebraic degree of $r$ is equal to 2 because $\deg(r_0) = \deg(r_1) = 2$, even though $\deg(r_2) = 1$.

The algebraic degree and the LAT coefficients are connected by the following well known lemma (see e.g. [Lan90, Prop. 1.5]).

**Lemma 8.2.5** (LAT coefficient divisibility)**.** *If all Walsh coefficients of an S-Box with $m$ input bits are divisible by $2^\ell$ then its algebraic degree is at most $m + 1 - \ell$.*

*Identically, if all* LAT *coefficients are divisible by $2^\ell$, the algebraic degree of the S-Box is at most $m - \ell$.*

A cryptographic primitive such as a block cipher $E_k$ must have a high algebraic degree. Otherwise, it is easy to find e.g. zero-sum distinguisher, that is, sets of inputs $\{x_0, \ldots, x_{M-1}\}$ such that $\bigoplus_{i=0}^{M-1} E_k(x_i) = 0$. In fact, Chapter 11 (p. 207) is devoted to the analysis of the algebraic degree of SPNs and to its use in cryptanalysis.

While the ANF represents a Boolean function as a multivariate polynomial with variables in $\mathbb{F}_2$, it is also possible to use the field structure of $\mathbb{F}_2^n$ by defining a polynomial representation in $\mathbb{F}_{2^n}$.

**Definition 8.2.5** (Univariate polynomial representation)**.** *Let $s : \mathbb{F}_2^n \rightarrow \mathbb{F}_2^n$ be an S-Box with identical input and output length. We can interpret the vectors of $\mathbb{F}_2^n$ as elements of a finite field $\mathbb{F}_{2^n}$, and, further, $s$ can be written as a* univariate polynomial *of $\mathbb{F}_{2^n}[X]$ in a unique fashion:*

$$s(X) = \sum_{i=0}^{2^n - 1} v_i X^i.$$

*This representation can be obtained using for example* Lagrange interpolation*. The* univariate degree *of $s$ is the maximum value of $i$ for which $v_i \neq 0$.*

The univariate representation of $r$ in $\mathbb{F}_{2^3} = \mathbb{F}_2[X]/(X^3 + X + 1)$ is

$$\text{Univ}(r) = (a+1)x^6 + x^5 + (a+1)x^4 + x^2 + (a^2+a)x + a^2 + a ,$$

where $a$ is a generator of the multiplicative group of $\mathbb{F}_{2^3}$.

The univariate degree and the algebraic degree are two different notions that should not be mistaken for one another. However, the two are related as follows.

**Lemma 8.2.6.** *Let $s : X \mapsto \sum_{i=0}^{2^n-1} v_i X^i$ be an n-bit S-Box. Its univariate degree is, by definition,* $\max(\{i, v_i \neq 0\})$. *Its algebraic degree $d_a$ is equal to*

$$d_a = \max(\{hw(i), v_i \neq 0\}) .$$

In particular, for a monomial $x \mapsto x^e$ of $\mathbb{F}_{2^n}$, the algebraic degree is the Hamming weight of $e$.

If the univariate degree of a function is too low, it may first lead to its algebraic degree being to low, as a consequence of Lemma 8.2.6. It may also lead to interpolation attacks such as the one [JK01] directed at the KN-cipher [NK95].

Finally, I mention the notion of High-Degree Indicator Matrix. It was introduced by Aleksei Udovenko and myself in [PU16].

**Definition 8.2.6** (High-Degree Indicator Matrix (HDIM))**.** *Let $S$ be an n-bit permutation. We define the High-Degree Indicator Matrix $\hat{H}(F)$ of $F$ to be the $n \times n$ matrix such that $\hat{H}(F)[i,j] = 1$ if and only if the* ANF *of $F_i$ contains the monomial $\prod_{k \neq j} x_k$ (which has degree $n-1$).*

For example, the HDIM of $r$ is equal to

$$\hat{H}(r) = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} .$$

The HDIM has some interesting properties. For starters, it is related to the LAT: the LAT coefficients of a permutation are always divisible by 2 but their congruence modulo 4 is determined by the HDIM. In order to establish this relation, we first derive an expression of the congruence modulo 4 of LAT coefficients.

**Lemma 8.2.7** (LAT modulo 4)**.** *Let $F$ be an n-bit permutation ($n > 2$) and let $\mathcal{L}$ be its* LAT*. Then $\mathcal{L}[a,b]$ is such that* $\mathcal{L}[a,b] \equiv 2 \times \left( \bigoplus_{x \in \mathbb{F}_2^n} \big(b \cdot F(x)\big)\big(a \cdot x\big) \right) \mod 4$ *or, equivalently,*

$$\frac{\mathcal{L}[a,b]}{2} \equiv \bigoplus_{x \in \mathbb{F}_2^n} \big(b \cdot F(x)\big)\big(a \cdot x\big) \mod 2.$$

*Proof.* First of all, we remark that $(-1)^z = 1 - 2z$ for $z$ in $\{0,1\}$. Using this equality, we can rewrite $\mathcal{L}[a,b]$ as

$$\begin{aligned} \mathcal{L}[a,b] &= \frac{1}{2} \sum_{x \in \mathbb{F}_2^n} (-1)^{a \cdot x + b \cdot F(x)} \\ &= \frac{1}{2} \sum_{x \in \mathbb{F}_2^n} (-1)^{a \cdot x} - \sum_{x \in \mathbb{F}_2^n} \big(b \cdot F(x)\big)(-1)^{a \cdot x} \end{aligned}$$

For $a \neq 0$, it holds that $\sum_{x \in \mathbb{F}_2^n} (-1)^{a \cdot x} = 0$. We deduce that the coefficient $\mathcal{L}[a,b]$ is equal to

$$\begin{aligned}\mathcal{L}[a,b] &= -\sum_{x \in \mathbb{F}_2^n} \big(b \cdot F(x)\big)(-1)^{a \cdot x} \\ &= -\sum_{x \in \mathbb{F}_2^n} \big(b \cdot F(x)\big) + 2\Big(\sum_{x \in \mathbb{F}_2^n} \big(b \cdot F(x)\big)\big(a \cdot x\big)\Big).\end{aligned}$$

The first term in this sum is equal to $2^{n-1}$ as every component of a permutation is balanced.[2] Thus, if we look at the congruence modulo 4 of $\mathcal{L}[a,b]$, we obtain the following (for any $n > 2$):

$$\mathcal{L}[a,b] \equiv 2\Big(\sum_{x \in \mathbb{F}_2^n} \big(b \cdot F(x)\big)\big(a \cdot x\big)\Big) \mod 4,$$

from which we deduce that

$$\frac{\mathcal{L}[a,b]}{2} \equiv \sum_{x \in \mathbb{F}_2^n} \big(b \cdot F(x)\big)\big(a \cdot x\big) \mod 2$$

As sum and XOR are equivalent modulo 2, this proves the lemma. □

Let $B(\mathcal{L})$ be a $2^n \times 2^n$ Boolean matrix with

$$B(\mathcal{L})[a,b] = \frac{\mathcal{L}[a,b] \mod 4}{2} ,$$

that is, $B(\mathcal{L})[a,b] = 0$ if and only if $\mathcal{L}[a,b] \equiv 0 \mod 4$. Because of Lemma 8.2.7 and because the scalar product is bilinear, this matrix has the following property:

$$B(\mathcal{L})[a \oplus a', b \oplus b'] = B(\mathcal{L})[a,b] \oplus B(\mathcal{L})[a,b'] \oplus B(\mathcal{L})[a',b] \oplus B(\mathcal{L})[a',b']. \quad (8.2)$$

In other words, the function $(a,b) \mapsto B(\mathcal{L})[a,b]$ is a bilinear form. In fact, this bilinear form is entirely defined by the HDIM. Indeed, Lemma 8.2.7 implies that

$$B(\mathcal{L})[a,b] = \bigoplus_{x \in \mathbb{F}_2^n} (a \cdot x)\,(b \cdot F(x)) ,$$

which, when $a$ and $b$ are elements of the canonical basis $(e_0, ... e_{n-1})$ of $\mathbb{F}_2^n$, is equal to

$$B(\mathcal{L})[e_i, e_j] = \bigoplus_{x \in \mathbb{F}_2^n} (e_i \cdot x)\,\big(e_j \cdot F(x)\big) . \quad (8.3)$$

The Boolean function $x \mapsto e_j \cdot F(x)$ has algebraic degree at most $n-1$ because $F$ is a permutation. Therefore, the only way the sum in Equation (8.3) can be equal to 1 is if the monomial $\prod_{j \neq i} x_j$ appears in the ANF of $x \mapsto e_j \cdot F(x)$, namely $F_j$. By definition of the HDIM, we immediately derive that $B(\mathcal{L})[e_i, e_j] = \hat{H}(F)[i,j]$. We deduce the following lemma.

[2] If $F$ is not a permutation but some function with degree at most $n-1$, then this term *a priori* does not go away when taking the modulo 4 of the expression.

**Lemma 8.2.8** (HDIM alternative definition)**.** *Let $F$ be a permutation of $\mathbb{F}_2^n$ and let $\hat{H}(F)$ be its HDIM. Then its HDIM coefficients are equal to*

$$\hat{H}(F)[i,j] = \bigoplus_{x \in \mathbb{F}_2^n} (e_i \cdot x) \left(e_j \cdot F(x)\right) .$$

Using Equation (8.2), we can express a connection between the HDIM and the LAT of a permutation which is stated in the following lemma.

**Lemma 8.2.9** (LAT and HDIM)**.** *Let $F$ be an $n$-bit permutation, let $\hat{H}(F)$ be its HDIM and $\mathcal{L}$ be its LAT. It holds that*

$$\mathcal{L}[a,b] \mod 4 \;=\; 2 \times \left(b^t \times \hat{H}(F) \times a\right).$$

Due to the fact that the LAT of the inverse of a permutation $F$ is the transpose of the LAT of $F$, we have the following lemma.

**Lemma 8.2.10** (HDIM of the inverse)**.** *Let $F$ be an $n$-bit permutation. The HDIM of $F^{-1}$ is the transpose of the HDIM of $F$.*

We now show that the HDIM of $\eta \circ f \circ \mu$ can easily be deduced from that of $f$ when $\eta$ and $\mu$ are $n$-bit linear permutations. The corresponding theorem will be used in Section 10.5.2 (p. 201) to attack Feistel Networks whitened using affine layers.

**Lemma 8.2.11.** *Let $\mu, \eta$ be linear $n$-bit mappings, $F$ be an $n$-bit permutation and let $G = \eta \circ F \circ \mu$. Furthermore, let $\hat{H}(F)$ be the HDIM of $f$ and $\hat{H}(G)$ be that of $G$. Then it holds that*

$$\hat{H}(G) = \eta \times \hat{H}(F) \times (\mu^t)^{-1}.$$

*Proof.* We prove this result in two steps. First, the fact that $\hat{H}(F \circ \mu) = \hat{H}(F) \times (\mu^{-1})^t$ can be derived as follows:

$$\begin{aligned}\hat{H}(F \circ \mu)[i,j] &= \bigoplus_{x \in \mathbb{F}_2^n} \left(e_i \cdot F(\mu(x))\right)\left(e_j \cdot x\right) = \bigoplus_{y \in \mathbb{F}_2^n} \left(e_i \cdot F(y)\right)\left(e_j \cdot \mu^{-1}(y)\right) \\ &= \bigoplus_{y \in \mathbb{F}_2^n} \left(e_i \cdot F(y)\right)\left((\mu^t)^{-1}(e_j) \cdot y\right).\end{aligned}$$

We then note that $\hat{H}(\eta \circ F) = \hat{H}(F^{-1} \circ \eta^{-1})^t$ which, using what we just found, is equal to $(\hat{H}(F^{-1}) \times \eta^t)^t = (\hat{H}(F)^t \times \eta^t)^t$, so that $\hat{H}(\eta \circ F) = \eta \times \hat{H}(F)$. This concludes the proof. □

## 8.3 S-Boxes in the Wild

In this section, I survey the literature on symmetric cryptography and collect all S-Boxes used along with their design criteria and structures (if known). I first go over S-Boxes defined using a mathematical structure in Section 8.3.1 (p. 147). Then, I describe S-Boxes built like small block ciphers in Section 8.3.2 (p. 152). These components can also be specified using electronic circuits or bit-sliced implementation, e.g. in the context of lightweight cryptography. Such S-Boxes are presented in Section 8.3.3 (p. 154). As explained in Section 8.3.5 (p. 155), hill climbing can also be used. Finally, it is also possible to construct S-Boxes as random functions or permutations using some nothing-up-my-sleeve quantity as a randomness source, as detailed in

Section 8.3.4 (p. 155). The popularity of the different structures found is summarized in Section 8.4 (p. 156).

Several rarer constructions are not discussed in this section. For example, injective S-Boxes mapping $m$ bits to $n$ with $m \ll n$ such as those used in Cast-128 [Ada97], Blowfish [Sch94], Dfc [GGH+98] and Coconut98 [Vau98] are not described here.

Even fewer constructions rely on data dependent S-Boxes. That is, the generation of the S-Box is part of the key schedule. This strategy is used by Twofish [SKW+98], Polar bear [HN+05], Oryx [WSD+99], and Cmea [WSK97]. Both Twofish and Cmea use a fixed S-Box to generate the data-dependent one. These are discussed in Sections 8.3.2.2 (p. 153) and 12.3.2 (p. 241) respectively.

## 8.3.1 Mathematical Objects

Some S-Boxes are built from mathematical objects. A fairly comprehensive survey of such design strategies can be found in [BN15]. By far the most popular is the finite field multiplicative inversion, used most notably by the Aes. This function is differentially 4-uniform in fields of even degree, in particular in $\mathbb{F}_{2^8}$. It is even apn in fields of degree 7 and 9, a property which is actually used e.g. in the block cipher Misty [Mat97]. The only known apn permutation operating on an even number of bits is defined over 6 bits and is in fact used by an algorithm. This 6-bit permutation and the structure we identified in it are discussed in Chapter 14 (p. 267).

### 8.3.1.1 Finite Field Inversion

The finite field inversion is a very popular choice due to its ideal differential uniformity, non-linearity and algebraic degree. These properties were first described by Nyberg in [Nyb94]. Here is a list of algorithms using a finite field inversion for their S-Box. If the algorithm uses several distinct S-Box, we append the number of S-Boxes based on the inverse over the total number of S-Boxes. First, the 8-bit S-Boxes are listed below. Algorithms using the exact same S-Box as the Aes are marked with a "†" symbol.

- Asc-1 [JK12] †
- Aes [DR02] †
- Ale [BMR+14] †
- Aria [KKP+04] (2/2) †
- Bksq [DR00] †
- Camellia [AIK+01] (4/4) †
- Chiasmus [STW13]
- Clefia [SSA+07] (1/2)
- Grøstl [GKM+11] †
- Hierocrypt 3 [OMSK01]
- Mugi [WFY+02] †
- Photon [GPP11] †
- Seed [LLY+05] (2/2)
- Shark [RDP+96]
- Sms4 [Ltd06]
- Snow 2.0 [EJ03] †
- Snow 3G [ETS06a] (1/2) †
- Square [DKR97]
- Zuc [ETS11] (1/2)

S-Boxes based on the inverse are used for other word sizes. Those are listed below.

- mCrypton [LK06] $n = 4$, (2/2)
- Mibs [ISSK09] $n = 4$
- Misty [Mat97] $n = 7, 9$, (2/2)
- Panda [YWH+14] $n = 4$,
- SC2000 [SYY+02] $n = 5, 6$, (2/3)
- Twine [SMMK13] $n = 4$
- Whirlwind [BNN+10] $n = 16$

The Seed block cipher is a standard in South Korea as well as an RFC standard [LLY⁺05] which, according to said RFC, “has been adopted by most of the security systems in the Republic of Korea”. It is also described in a document published by its authors[3] which shows that the two S-Boxes are affine equivalent to $x \mapsto x^{247}$ and $x \mapsto x^{251}$ respectively, both of which are in the cyclotomic class of $x \mapsto x^{-1}$ as $4 \times (2^8 - 2) \equiv 251 \mod (2^8 - 1)$ and $8 \times (2^8 - 2) \equiv 247 \mod (2^8 - 1)$.

For Sms4, the structure of the S-Box used was not made public. However, it was found to be affine-equivalent to the inverse function by Liu *et al.* in [LJH⁺07].

For Chiasmus, a thorough analysis is provided in Section 9.3 (p. 177). Monomials in general and the multiplicative inverse in particular are easily recovered from an unknown S-Box built by composing a monomial with affine functions, as shown in Section 9.1.2.4 (p. 163).

#### 8.3.1.2 Apn Functions

Apn permutations are harder to use in cryptography as those either operate on an odd number of bits or on 6 bits, neither of which are as convenient as 8 bits. Still, the following algorithm use APN function on $n$ bits:

- 3-WAY [DGV94] $n = 3$
- Fides [BBK⁺13] $n = 5, 6$
- Misty [Mat97] $n = 7, 9$
- SC2000 [SYY⁺02] $n = 5$

#### 8.3.1.3 Finite Field Exponentiation

Another mathematical structure received significant attention from cryptographers, both from the academic community and some government agencies: exponential S-Boxes. These are built from the exponentiation in finite field or its inverse, the discrete logarithm. However, the exponential has no preimage for 0. It must therefore be chosen arbitrarily and, unfortunately, not all papers on this type of structure “agree” on what this position should be.

S-Boxes built from a particular type of linear recurrence equivalent to a finite field exponentiation and where 0 maps to 0 are called *exponential substitution.* They were introduced and some of their properties were studied in papers by some of the BelT designers, BelT [Bel11] being the current standard block cipher of Belarus. The first one is a paper published in “Вести НАН Беларуси” [News of the National Academy of Sciences of Belarus] [AA05] while the second is a translation which the authors published on eprint [AA04].

**Definition 8.3.1** (Exponential substitution [AA04, AA05])**.** *Let $\alpha$ be a primitive element of the finite field $\mathbb{F}_{2^n}$ with minimal polynomial $x^n + \sum_{i=1}^{n-1} m_i x^i + 1$. First, let $x$ be an element of $\mathbb{F}_2^n$ and denote $\overline{x} = \sum_{i=0}^{n-1} x_i 2^i$. An* exponential substitution *is a permutation $s$ such that*

$$s(x) = \begin{cases} 0 \text{ if } x = 0, \\ \alpha^{\overline{x}} \text{ otherwise.} \end{cases}$$

*In this case, the truth table of each coordinate $s_i$ of $s$ is such that, for all $x$ less than $2^n - n$,*

$$s_i(x+n) \oplus m_{n-1} s_i(x+n-1) \oplus \ldots \oplus m_1 s_i(x+1) \oplus s_i(x) = 0.$$

[3] Available online at http://seed.kisa.or.kr/html/egovframework/iwt/ds/ko/ref/%5B2%5D_SEED+128_Specification_english_M.pdf.

*In other words, each coordinate of $s$ is a segment of an* LFSR *sequence with linear recurrence given by the same irreducible polynomial which is used to define the finite field.*

In particular, it is shown in [AA04, AA05] that exponential substitutions have reasonably high non-linearity and high algebraic degree. It is also shown that the differential uniformity is low as well as the probability of hybrid differentials mixing modular addition and XOR such as $s(x \oplus a) = s(x) \boxplus b$. This type of differential is studied more thoroughly in Section 9.1.2.5 (p. 167).

To formalize the difference between such structures and other definitions of exponential/discrete logarithm-based S-Boxes which use a different preimage for 0, we introduce the following definition. While the impact of the difference between inserting 0 at position 0 or at position $2^n - 1$ is limited as it merely rotates the truth table, inserting it in the middle of the truth table of the S-Box has more consequences.

**Definition 8.3.2** (Pseudo-Exponential/Logarithm)**.** *We call* pseudo-exponential *of $n$ bits a permutation defined by an exponent $\lambda$ generating the multiplicative group of some field of size $2^n$ and a position $z$ at which 0 is inserted. It is denoted $exp_{\lambda,z}$, so that*

$$exp_{\lambda,z}(x) = \begin{cases} 0 \text{ if } \overline{x} = z, \\ \lambda^{x \boxminus 1} \text{ if } \overline{x} < z, \\ \lambda^{x} \text{ otherwise}. \end{cases}$$

*A* pseudo-logarithm *is the functional inverse of a pseudo-exponential. The functional inverse of $exp_{\lambda,z}$ is denoted $log_{\lambda,z}$.*

Both discrete logarithm [HN10] and exponential [BR00c, RBF08] have been discussed in the Western literature before but the definitions used differ slightly. As it is defined in [HN10], the discrete logarithm maps $x \neq 0$ to $\log_\alpha(x)$ for some primitive element $\alpha$ of $\mathbb{F}_{2^n}$ and it maps 0 to $2^n - 1$. Thus, its inverse is not an exponential substitution in the sense of Definition 8.3.1: it maps $x$ to $\alpha^x$ unless $x = 2^n - 1$ which is mapped to 0. In this case, 0 is not inserted at position 0 but at position $2^n - 1$. This is the structure used by the block cipher MAGENTA [JH98]. In [RBF08], rotational symmetries of such exponentials were studied. They were also used to build the small 4-bit S-Box $E$ used inside the Whirlpool hash function [BR00c]. Therefore, what is called "exponential" in those papers is here called "pseudo-exponential with a 0 at position $2^n - 1$", which is denoted $\exp_{\lambda,2^n-1}$.

It is possible to represent the S-Box $H$ of BelT using a pseudo-exponentiation. In order to find the decomposition described below, we brute-forced all possible bases for the exponentiation and all field representations to obtain pseudo exponentials $\exp_{\lambda,10}$. For each, we checked whether $H \circ \exp_{\lambda,10}^{-1}$ was linear and, if it were, computed its matrix representation. Only one of those matrices has a visible structure; it corresponds to the decomposition we kept. Let $\omega_H$ be the symmetric matrix defined by

$$\omega_H = \begin{bmatrix} 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 \end{bmatrix}$$

and let $\lambda = w^7 + w^3 + w$, where $w$ is the generator of the multiplicative group of $\mathbb{F}_2[x]/(x^8 + x^6 + x^5 + x + 1)$. Then $H(x)$ can be computed as follows:

$$H(x) = (\omega_H \circ \exp_{\lambda,10})(x).$$

Furthermore, as explained in [HN10], the coordinates of $\exp_{\lambda,2^n-1}$ are closely related to the functions introduced by Feng *et al.* in [FLY09] and studied in [CF09]. In fact, when the input and output are of the same size, the function of Feng *et al.* is the discrete logarithm except in $\{0, 1\}$: it maps 0 to 0 and 1 to $2^n - 1$.

This structure is not nearly as popular as the inverse function but is has been used by several algorithms listed below.

- Magenta [JH98] $n = 8$
- Whirlpool [BR00c] $n = 4$
- Whirlbob [SB15] $n = 4$
- BelT [Bel11] $n = 8$
- Stribog [Fed12]? $n = 8$
- Kuznyechik [Fed15]? $n = 8$

The S-Box of the hash function Whirlpool has a recursive structure detailed in Section 8.3.2.3 (p. 153). The 4-bit permutation it uses is a pseudo-exponential in $\mathbb{F}_{2^4}$ which maps $2^4 - 1$ to 0. The same 8-bit S-Box is used by the second round Caesar candidate Whirlbob.

The case of the S-Box shared by the hash function Stribog and the block cipher Kuznyechik, both of which are Russian standards, is quite peculiar. Indeed, this S-Box was only specified as a look-up table in the original specifications of these algorithms. However, we managed to recover some hidden structure in it. In fact, an entire chapter of this thesis is devoted to this topic, namely Chapter 13 (p. 245). In it, we describe two decompositions. The first is reminiscent of a 2-round Feistel Network with finite fields multiplications instead of xor, which is I we also list this S-Box among the Feistel-based in Section 8.3.2 (p. 152). Moreover, we also identified a close proximity of this S-Box to a pseudo-exponential permutation of $\mathbb{F}_{2^8}$: this S-Box is functionally equivalent to such a permutation composed with a very weak 8-bit permutation with a differential uniformity of 128.

#### 8.3.1.4 Other Constructions

Other mathematical constructions have been used, notably by the block ciphers Safer [Mas94], E2 [KMA⁺00] and Picaro [PRC12] as well as by the stream cipher Snow 3G [ETS06a].

**Modular Ring Exponentiation.** Safer is a block cipher which was first proposed in 1993 by Massey. It uses an 8-bit S-Box along with its inverse in the round function. It is a bijection based on exponentiation in the prime field $\mathbb{Z}/257\mathbb{Z}$ mapping $x \in \mathbb{Z}/256\mathbb{Z}$ to $45^x$. The rationale behind this choice is given by Massey in the original paper:

> Thus the mapping "$45^{(\cdot)}$" is an invertible mapping from one byte to one byte that is very nonlinear with respect to the arithmetic of $GF(257)$ as well as with respect to the vector space of 8-tuples over the binary field $GF(2)$ whose addition is bit-by-bit xor.

However, it must be noted that the highest LAT and DDT coefficients are respectively equal to 46 and 128, both of which are far higher than what could be expected from a random 8-bit permutation, as explained in Section 9.4 (p. 178).

**Dickson Polynomial.** Snow 3G is a stream cipher used by the 3GPP standard. The structures of its components are described in [ETS06b]. Its S-Box $S_2$ is based on a Dickson polynomial [Dic96]. Dickson polynomials are bivariate polynomials $D_i(x,y)$ given by the following induction in the case where the variables are in $\mathbb{F}_{2^n}$:

$$\begin{cases} D_0(x,y) = 0 \\ D_1(x,y) = x \\ D_i(x,y) = xD_{i-1}(x,y) + yD_{i-2}(x,y) \ . \end{cases}$$

The reason behind this choice is stated in Section 9.4.1.3 of [ETS06b]:

> There are few criteria for the selection of $S_2$:
>
> - it should have a higher Algebraic Immunity than [the S-Box of the Aes],
> - it should not lead to a design that is weaker than Snow 2.0, and
> - it should be relatively "cheap" to implement.

**Bivariate Polynomials.** Picaro was designed to allow an easy masking of its evaluation so as to thwart side-channel attacks. The key metric to minimize for this purpose is the number of finite field multiplications performed during encryption which in turn implies that the S-Box should use few multiplications as well. Because of its Feistel structure, the S-Box of Picaro does not need to be a bijection. It is evaluated as a pair of bivariate polynomial mapping $(\mathbb{F}_{2^4})^2$ to itself:

$$f : \begin{cases} \mathbb{F}_{2^4} \times \mathbb{F}_{2^4} \rightarrow \mathbb{F}_{2^4} \times \mathbb{F}_{2^4} \\ (x,y) \mapsto \left(xy, (x^3+\omega)(y^3+\omega')\right) \ , \end{cases}$$

where $\omega$ and $\omega'$ are some constants. This structure is derived from a previous paper by Carlet [Car11].

**Combining Finite Field and Modular Arithmetic.** E2 [KMA$^+$00] was a candidate for the Aes competition based on a Feistel network using a Feistel function with a SAS structure. It relies on a unique 8-bit bijective S-Box, denoted $S$. It was obtained by composing two functions $f$ and $g$, where $f$ is defined over a modular ring as

$$\begin{cases} f : & \mathbb{Z}/256\mathbb{Z} \mapsto \mathbb{Z}/256\mathbb{Z} \\ & x \mapsto 97 \times x + 225 \ , \end{cases}$$

and $g$ is a finite field monomial with an exponent in the cyclotomic class of the inverse:

$$\begin{cases} g & : \mathbb{F}_{2^8} \rightarrow \mathbb{F}_{2^8} \\ & x \mapsto x^{127} \ . \end{cases}$$

Composing functions operating on different groups was seen by the authors of E2 as a strategy to thwart algebraic attacks.

### 8.3.2 Block Cipher-Based

S-Boxes must have good cryptographic properties so that a block cipher using them has good cryptographic properties as well. This recursive phenomenon, whereby the strength of the smaller component is propagated upwards to the larger object, can be used backwards as well: the S-Box itself may be built from even smaller strong subcomponents. In this case, it is natural to use a small block cipher structure to build the S-Box from smaller ones, namely a Feistel, an SPN or a Lai-Massey structure. A remainder about these types of round function is provided in Section 1.2.1.1 (p. 4).

Such a recursive structures lend themselves well to efficient hardware implementations. Indeed, it is far easier to implement e.g. two layers of good 4-bit S-Boxes than a random 8-bit S-Box. In fact, the use of a block cipher structure to build an S-Box is very often justified by the lower circuit size it implies. Thus, most of the S-Boxes mentioned in this section also fit in Section 8.3.3 (p. 154).

As shown in Chapters 11 (p. 207) and 10 (p. 181), S-Boxes built using SPN and Feistel structures can be decomposed given their full LUT unless many rounds are used.

#### 8.3.2.1 Feistel Networks and Misty Structures

The Feistel and Misty structures have been used to build the S-Boxes of several algorithms. A list is provided below, the number between parenthesis being the number of rounds inside the S-Box. All of them are Feistel Networks, except for the S-Box of Fantomas (which is also used by Scream [GLS+14]). The S-Box first found by Canteaut *et al.* [CDL16] is used by the Scream [GLS+14] CAESAR candidate.

- Crypton 0.5 [LH98] (3)
- CS-cipher [SV00] (3)
- Kuznyechik [Fed15] (2;?)
- Fantomas [GLSV15] (3)
- iScream [GLS+14] (3)
- Scream [CDL16, GLS+14] (3)
- SKINNY-128 [BJK+16] (4)
- Zorro [GGNS13] (4)
- ZUC [ETS11] (3)

In all cases, lightweightness was a criteria. In the case of Zorro, the total number of multiplication in $\mathbb{F}_{2^4}$ was also minimized to aid the implementation of masking, much like in the S-Box of PICARO.

The Feistel structure may be tweaked slightly. For example, in the case of Zorro, the structure is a bit peculiar in that the branch swap is replaced by a more complex bit permutation. This leads to the existence of peculiar patterns in HDIM and LAT which are presented in Section 12.2.2 (p. 229). Similarly, the S-Box $s_0$ of ZUC is composed with a bit rotation.

The bigger variants of the lightweight block cipher Skinny [BJK+16] use an 8-bit S-Box built using a generalized Feistel Network reminiscent of the high-level structure of the Piccolo [SIH+11] block cipher. The internal state is divided into four 2-bit branches which go through 4 Feistel rounds. In each, a simple `nor` is used as a Feistel function which is called twice. The branches then undergo a sophisticated shuffling rather than a simple rotation.

Finally, the S-Box of Kuznyechik can be represented as a sort of 2-round Feistel Network where the usual XOR has been replaced with finite field multiplications in $\mathbb{F}_{2^4}$. However, it is still unclear whether this Feistel-like representation exists by design or as a side-effect of some other structure, as explained in Chapter 13 (p. 245).

#### 8.3.2.2 Substitution-Permutation Network

The SPN structure has been used to build S-Boxes from smaller ones. The layers used inside an SPN can be described using S to denoted a layer of parallel smaller S-Boxes and A to denoted an affine layer, as detailed in Chapter 11 (p. 207).

Some of the 8-bit S-Boxes of the following algorithms have an SPN structure:

- ASASA scheme [BBK14]
- Clefia [SSA+07] (SAS)
- Crypton 1.0 [DPV01] (SAS)
- Enocoro [WIK+08] (SASA)
- Iceberg [SPR+04] (SASAS)
- Khazad [BR00b] (SASAS)
- Midori-128 [BBI+15] (ASA)
- Qarma-128 [Ava17] (ASA)
- Twofish [SKW+98] (ASAS)

In all cases other than the ASASA white-box scheme of Biryukov *et al.*, the main aim is to facilitate the hardware implementation of the S-Box.

In fact, the design criteria for the S-Boxes of Iceberg and Khazad are identical. These have to be involutions with $\Delta_s \leq 8$, $\mathcal{L}_s/2 \leq 32$, a non-linearity order equal to 7 and no fixed points. The 4-bit involutions used, $s0$ and $s1$ for Iceberg, $P$ and $Q$ for Khazad, where chosen using some form of hill climbing among the set of the differentially 4-uniform permutation with best non-linearity and best nonlinear order.

For Twofish, the four 4-bit S-Boxes used inside each 8-bit S-Box have been generated independently at random, a process which was iterated until a good enough S-Box was found. Similarly, the two distinct 4-bit involutions used to build the four S-Boxes of Crypton v1.0 were chosen as those implying the best properties for the big 8-bit construction out of a set of 4-bit involutions.

For Midori-128, the four 8-bit S-Boxes are built by combining the 4-bit S-Box $\text{Sb}_1$, chosen for its low-latency and low energy consumption, with simple bit permutations — one for each of the four 8-bit S-Boxes. The bit permutation is applied first, then two calls to $\text{Sb}_1$ are made in parallel and, finally, the inverse of the bit-permutation is called. The aim was to reduce the overall latency and energy consumption. The 8-bit S-Box of the larger variant of Qarma, Qarma-128, is built similarly and for the same reason.

The ASASA scheme was intended for white-box cryptography, the aim being to force regular users to store big tables while privileged users could use the secret ASASA decomposition of these tables to save space. White-box cryptography is discussed more thoroughly in Part III of this thesis.

#### 8.3.2.3 Lai-Massey

This structure is not very common for block cipher design so it should come as no surprise that few S-Boxes use it. Still, the following algorithms rely on S-Boxes using a Lai-Massey round. The number between parenthesis is the number of rounds used.

- Fox (Idea NXT) [VJ04] (3)
- Whirlpool [BR00c] (1)
- Fly [KG16] (1)

In the case of Fox (also known as Idea NXT), the aim was to avoid the use of an algebraic structure. It uses 3 different 4-bit functions which were chosen to have optimal differential uniformity, linearity and algebraic degree. The exact instances used for these subfunctions were chosen randomly until a good candidate for the larger 8-bit S-Box was found. Therefore, this S-Box also fits in the category of the S-Boxes built *via* hill climbing described in Section 8.3.5 (p. 155).

The S-Box of Whirlpool uses only one Lai-Massey round sandwiched between two 4-bit S-Box layers. These 4-bit S-Boxes are based on a pseudo-exponential function, as explained in Section 8.3.1.3 (p. 148).

Finally, the S-Box of Fly uses only one Lai-Massey round followed by an S-Box layer. The 4-bit S-Box used within the Lai-Massey round and the S-Box layer was chosen for its efficient bitsliced implementation — it is in fact the one of [UDCI+11]. In this case, this overall structure was chosen for its lightweightness and the fact that it has both a differential and a linear branching number equal to 3, meaning that no differential/linear pattern exist that connect a 1-bit input with a 1-bit output with non-trivial probability.

### 8.3.3 Simple Circuit

A very popular design criteria, especially for small 4- or 5-bit S-Boxes, is the existence of an efficient hardware and/or bit-sliced implementation. For instance, [UDCI+11] provides a comprehensive search for a 4-bit S-Box with optimal cryptographic properties (differential uniformity, linearity and algebraic degree) with a bit-sliced implementation as small as possible. The final result is the *Class 13* set of S-Boxes. Members of this class have been used directly, e.g. in RoadRunner and Mysterion. They have also been used as sub-components of some 8-bit S-Boxes, namely those of Fly, and Zorro. A bit-sliced implementation operating on four words $a, b, c, d$ is provided in [KG16]; we reproduce it in Algorithm 8.1.

**Algorithm 8.1** The bit-sliced implementation of the Class 13 member used in Fly.
**Inputs:** words a, b, c, d;
**Outputs:** updated words a, b, c, d.

```
t  = b;   b |= a;   b ^= c; // (B): c ^ (a | b)
c &= t;   c ^= d;           // (C): d ^ (c & b)
d &= b;   d ^= a;           // (D): a ^ (d & B)
a |= c;   a ^= t;           // (A): b ^ (a | C)
```

Several S-Boxes for which a small circuit and/or a small bit-sliced implementation was a design criteria are listed below. All of them are 4-bit permutations, except for the S-Box of Ascon [DEMS16] which permutes $\mathbb{F}_2^5$. When an 8-bit S-Box is built out of smaller 4-bit S-Boxes for efficiency reasons, it is instead listed in the category corresponding to the high-level structure used in Section 8.3.2 (p. 152). The S-Box of Present has been reused by multiple algorithms indicated by a "†" mark. Joltik borrows the S-Box of Piccolo and Skinny-64 uses a variant of it.

- Ascon [DEMS16]
- Epcbc [YKPH11] †
- Gost rev. [PLW10] †
- ITUbee [KDH13]
- Joltik [JNP14a]
- Hummingbird-2 [ESSS12]
- LBlock [WZ11]
- Led [BKL+07] †
- Lilliput [BFMT15]
- Noekeon [DPVAR00]
- Midori-64 [BBI+15]
- Mysterion [JSV17]
- Photon [GPP11] †
- Piccolo [SIH+11]
- Present [BKL+07] †
- Pride [ADK+14]
- Prince [BCG+12]
- Prøst [KLL+14]
- Qarma-64 [Ava17]
- Rectangle [ZBL+15]
- RoadRunner [BS15]
- SC2000 [SYY+02]
- Serpent [BAK98]
- Skinny-64 [BJK+16]
- Spongent [BKL+11] †

For Rectangle, Serpent, Hummingbird-2, Present, and many of the algorithms sharing its S-Box, another additional criteria was added: the minimization of the number of 1-bit to 1-bit entries in both the ddt and the lat. Still, an efficient hardware or bitsliced implementation was an important requirement in their design.

### 8.3.4 Hill Climbing

Another method for building S-Boxes with good differential/linear properties is the use of a form of hill-climbing. The general idea is to swap some entries and check if the properties were improved. If so, we keep iterating this transformation. If not, we backtrack. Below, I list some S-Boxes that have been built in this fashion. All of them are 8-bit permutations.

- Anubis [BR00a]
- Fox (Idea nxt) [VJ04]
- Kalyna [OGK+15b]
- Skipjack [U.S98] (?)

The S-Box of Anubis was picked from a set of about 600 millions random 8-bit involutions. It had to have $\Delta_s \leq 8$, $\mathcal{L}_s/2 \leq 32$[4], and no fixed points. Finally, every value of $S(x) \oplus x$ occurs exactly twice.

As described before, Fox uses a Lai-Massey structure where the 4-bit sub-functions have been chosen through hill-climbing.

Kalyna is the current standard block cipher in Ukraine. The standard hash function, Kupyna [OGK+15a], uses the same S-Boxes. Their 4 distinct S-Boxes were all generated in the same way: starting from a random S-Box, random entries were swapped. If the result has better differential or linear properties, then try again using the new swapped S-Box as a starting point. This method is described in more details in [KKO13]. A significant amount of resources was devoted to this computation as it took several hours on a cluster of 4096 processors.

We show in the next chapter, Section 9.2.3 (p. 172) that the peculiar linear properties of the S-Box of Skipjack can be accurately imitated using a hill-climbing algorithm with a specific optimization criteria. However, the nsa has yet to release the actual method they used.

### 8.3.5 Random Generation

In other designs, the authors deemed a random permutation to offer a sufficient level of security. Some algorithms using such 8-bit S-Boxes are listed below.

- MD2 [Kal92]
- newDES [Sco85]
- Turing [RH03]

Generating an S-Box randomly is very easy. The challenge in this case is to generate the S-Box in an way that is easy to duplicate and using a process simple enough that no hidden weakness could have been hidden. For example, the S-Box of the stream cipher Turing was generated using the internal state of the stream cipher RC4: it was first seeded with the string “Alan Turing” and then the internal state

[4] It is not the case as $\mathcal{L}_s/2 = 34$. This is due to a simple bug in the authors program which they acknowledge in the specification.

was updated 10,000 times. The S-Box with the best properties among those 10,000 permutations was kept.

Similarly, the S-Box of newDES was derived using a simple algorithm along with the American declaration of independence as an external source of entropy. That of the hash function MD2 was obtained[5] from the digits of the constant $\pi$.

## 8.4 Summary of the Structures Found in the Literature

Figure 8.1 summarizes the structure used to build all the 8-bit S-Boxes mentioned in this chapter. If a cipher uses several S-Boxes built in the same way, it counts as 1. If it uses different structures, as say Clefia, it counts as 1 in each category. The S-Box of Fox was built using hill climbing but, most importantly, it has a Lai-Massey structure. Thus, it is in the latter category. The numbers for each structure are as follows:

- Inverse: 19 (incl. 11 Aes),
- Exponential: 2,
- Other math: 4
- SPN: 9,
- Feistel: 7,
- Misty: 1,
- Lai-Massey: 3,
- Hill climbing: 2,
- Pseudo-random: 3,
- Unknown: 3.

These are summarized in Figure 8.1 where mathematical structures are in shades of blue, block ciphers in warm colors and those built with heuristic or unknown methods are in shades of gray. The algorithms which merely re-use the Aes S-Box are indicated as such within the inverse category.

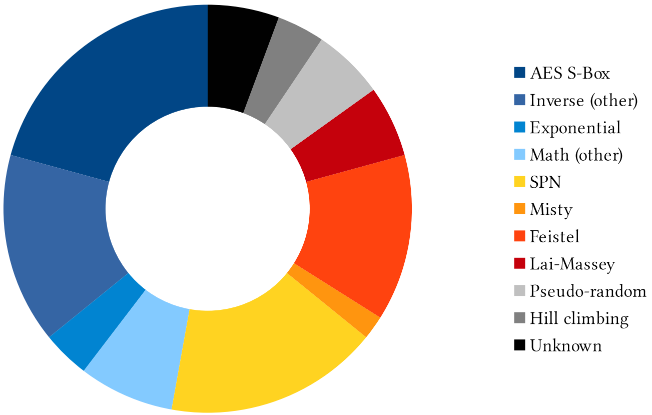


Figure 8.1: Types of structure used to build 8-bit S-Boxes by 53 different algorithms from the literature.

A summary of the cryptographic properties of the 8-bit S-Boxes mentioned above along with additional criteria is provided at the end of the next chapter, in Section 9.4 (p. 178).

[5] The details of the algorithm used are available on `crypto.stackexchange`. One of its users asked Rivest for his generation algorithm and published his answer (https://crypto.stackexchange.com/questions/11935/how-is-the-md2-hash-function-s-table-constructed-from-pi).

## 8.5 The Need for S-Box Decomposition

As we have seen, many different methods have been used to design the S-Boxes actually used to build block ciphers and hash functions. However, the method used and even the very design criteria considered have been kept secret by some designers, most notably the American and Russian secret service (NSA and FSB respectively). This observation raises an obvious question.

> Using only its look-up table, how can we recover the hidden structure and/or the design criteria used to build an S-Box?

The field of *S-Box reverse-engineering* aims at answering this question. The remainder of this part of my thesis is devoted to this topic. It is based on papers I was fortunate to co-author with the following cryptographers: Alex Biryukov, Anne Canteaut, Sébastien Duval, Dmitry Khovratovich, Gaëtan Leurent and Aleskei Udovenko.

The problem of recovering the hidden structure of an S-Box is related to the idea of attacking so-called *white-box* implementations of block ciphers. The aim of the designer in this context is to build an implementation of a primitive in such a way that an attacker with full access to said implementation cannot

- find an implementation of the decryption function (strong white-box), or
- find a smaller implementation (weak white-box).

As a consequence, the aim of the attacker is to recover the hidden components used to build said implementation. This task is fairly similar to the decomposition of an S-Box from its LUT. The topic of white-box cryptography is discussed much more thoroughly in Part III.

I conclude this chapter with a list of block ciphers for which the structure/design method used to build the S-Boxes have not been specified by their designers. First, the “mysterious” 8-bit S-Boxes I am aware off are listed in Table 8.4 along with the results we managed to derive about them.

| Algorithms | Source | Our results |
|---|---|---|
| Skipjack [U.S98] | NSA | Optimized linear properties, possibly *via* hill climbing; see Section 9.2 (p. 168). |
| Chiasmus [STW13] | BSI † | Based on the multiplicative inverse function; see Section 9.3 (p. 177). |
| Stribog [Fed12]<br>Kuznyechik [Fed15] | FSB | Feistel- and exponential-like structures found, see Chapter 13 (p. 245). |
| CMEA [WSK97] | TIA ‡ | A TU-decomposition is possible, as shown in Section 12.3.2 (p. 241). |

† Federal office for information security (Germany).

‡ Telecommunications Industry Association (USA).

Table 8.4: 8-bit S-Boxes with unknown structures and our results about them.

There are also some unknown 4-bit S-Boxes. The Data Encryption Standard (DES) [U.S99] uses eight S-Boxes mapping $\mathbb{F}_2^6$ to $\mathbb{F}_2^4$. Each of them consists of 4 permutations of $\mathbb{F}_2^4$. While some design criteria have been published [Cop94], the exact generation procedure remains mysterious. Furthermore, some patterns were noticed in the 70's which, to the best of my knowledge, remain unexplained [HMS+76].

The Russian GOST block cipher [Dol10a] and its variants use different sets of S-Boxes. Unless those come from academia, as in GOST revisited and its use of the S-Box of PRESENT, the source of the S-Boxes remains mysterious. This cipher could still be used today as Magma, a variant of the old GOST, is specified in the same document as Kuznyechik [Fed15]. The eight 4-bit S-Boxes of Magma, which are denoted $\pi'_i$ for $i = 0, ..., 7$, are all affine-equivalent to one another: there are constants $a_i$ and linear permutations $A_i$ and $B_i$ such that $\pi'_i(x) = B_i\left(\pi'_0\left(A_i(x \oplus a_i)\right)\right)$ for all $i$.

While our results summarized in Table 8.4 show that attacking 8-bit S-Boxes is possible, our techniques are unlikely to work as well against 4-bit S-Boxes. Thus, the following problem remains largely open.

**Open Problem 8.5.1** (Reverse-Engineering 4-bit S-Boxes)**.** *What is the exact structure used by the designers of the DES and the GOST variants to build their small S-Boxes?*

Chapter 9

# Statistical Analysis of the DDT/LAT

Block ciphers such as the Des [U.S99], Skipjack [U.S98], as well the more recent Magma [Fed15] and Kuznyechik [Fed15] use different S-Boxes for which the generation method is partially or completely unknown. It is worth pointing out that the common thread linking these algorithms is their having been designed by governmental agencies. Indeed, the first two have been designed at least in part by the American National Security Agency (NSA) and the latter two by its Russian counterpart, the FSB.

As early as 1976, cryptanalysts have attempted to recover the design criteria or the hidden structure in the S-Boxes of the Des [HMS+76]. In fact, they managed to recover several of the design criteria which were published 18 years later by Coppersmith [Cop94]. Still, the exact process used remains unknown to this day.

In this chapter, I describe efficient methods for figuring out whether an S-Box could have been picked uniformly at random from the set of all possible S-Boxes or not. The idea is to look very carefully at the distribution of the coefficients in both its DDT and its LAT. First, I describe several patterns indicating that an S-Box cannot have been picked uniformly at random in Section 9.1 (p. 159). I then use these methods to study the S-Box of Skipjack in Section 9.2 (p. 168) and that of Chiasmus in Section 9.3 (p. 177). Finally, I describe the results of an analysis of all the 8-bit S-Boxes I know of in Section 9.4 (p. 178).

When attempting to reverse-engineer an S-Box, it is crucial to ground the investigation into a solid mathematical foundation, such as proper probability computations. Indeed, as already noted 40 years ago by the cryptographers who tried to reverse-engineer the S-Boxes of the Des [HMS+76]:

> The problem [of S-Box reverse-engineering] is complicated by the ability of the human mind to find apparent structure in random data, which is really not a structure at all.

## 9.1 Non-Randomness in the DDT/LAT

As discussed in Chapter 8 (p. 137), the DDT and LAT play a crucial role in assessing the security provided by an S-Box from a cryptographic stand-point. In this section, I illustrate how these tables can be put to use by a cryptanalyst attempting to recover information about the design process of an S-Box.

### 9.1.1 Coefficient Divisibility and Algebraic Degree

Lemma 8.2.5 links the algebraic degree of an S-Box with the congruence of its LAT coefficients modulo $2^{\ell}$. It provides a simple tool for identifying some non-random S-Boxes using only the absolute value of their LAT. Indeed, a function mapping $\mathbb{F}_2^m$ to $\mathbb{F}_2^n$ has an algebraic degree strictly smaller than $m-1$ with probability at most $1/2^{n \times m}$. This is due to the fact that each of the $m$ possible monomials of degree $m-1$ occurs in the ANF of each of the $n$ coordinates with an independent probability equal to $1/2$. Thus, if the LAT coefficients are all divisible by e.g. 4, then we can rule out that the S-Box has been picked uniformly at random.

Obviously, this distinguisher can be used independently of the LAT by computing the algebraic degree of the function directly.

### 9.1.2 Statistical Artifacts in Coefficient Distributions

Since it is better for an S-Box to have a low differential uniformity and a low linearity, it is natural to expect S-Box designers to minimize these quantities. In light of this, when given an S-Box of unknown origin, we must ask ourselves if the distribution of the coefficients in the DDT/LAT is compatible with its having been picked uniformly at random. If it is not the case, this information allows a cryptanalyst to recover some of the design criteria used by the designers.

We discard the first row and the first column of the DDT and LAT of permutations in this section. Indeed, the distribution of the coefficients in those is useless from a cryptanalyst's perspective as they are always the same.

#### 9.1.2.1 Coefficient Distributions

Such an analysis requires a careful study of the distribution of the coefficients in the DDT/LAT of an S-Box. Fortunately, these distributions are known for both random functions and random permutation as they were derived by Daemen and Rijmen in [DR07] as well as, in the case of the LAT, in [O'C95]. These results are based on the following assumption.

**Assumption 9.1.1** (Coefficient Independence). *Each coefficient in the DDT/LAT of a random function (or permutation) is an independent sample from a given distribution.*

The validity of this assumption is supported in [DR07] by experimental data.[1] Below, I list their results regarding the distribution of the coefficients in the DDT and LAT of a random function and permutation. Theorem 9.1.1 is a rephrasing of Corollaries 2 and 4 of [DR07] while Theorem 9.1.2 repeats the content of both Theorem 1 of [O'C95] and Theorem 5 of [DR07].

**Theorem 9.1.1** (DDT coefficient distribution). *Under Assumption 9.1.1, the coefficients* $\mathrm{DDT}[i,j]$ *in the DDT of a random S-Box mapping $m$ bits to $n$ (with $m \geq 5$ and $m-n$*

[1] This assumption is technically not true because the coefficient of the Walsh spectrum (from which the LAT is deduced) must satisfy Parseval's equality, namely $\sum_a W_{a,b}^2 = 2^{2n}$ for all $b$. Similarly, the coefficients of the DDT must be such that $\sum_\delta \mathrm{DDT}[\delta,\Delta] = 2^n$ for all $\Delta$. Nevertheless, Assumption 9.1.1 is a safe assumption in the sense that the results derived from it predict very precisely the distributions observed in practice. Besides, a thorough mathematical analysis of the case of the LAT is provided in [O'C94] which shows that the approximation we use for this table is a valid one.

*small) are independent and identically distributed random variables with a probability distribution which can be approximated by:*

$$\Pr[\mathrm{DDT}[i,j] = 2z] \;=\; \mathrm{Poisson}(z, 2^{m-n-1}) \;=\; e^{-2^{m-n-1}} \frac{2^{(m-n-1)z}}{z!}. \qquad (9.1)$$

*In particular, for an n-bit random permutation ($m = n$), this probability distribution becomes*

$$\Pr[\mathrm{DDT}[i,j] = 2z] \;=\; \frac{e^{-1/2}}{2^z z!}. \qquad (9.2)$$

**Theorem 9.1.2** (LAT coefficient distribution [O'C95, DR07]). *Under Assumption 9.1.1, the coefficients* LAT$[i,j]$ *in the LAT of a random function mapping m bits to n are independent and identically distributed random variables with the following probability distribution:*

$$\Pr[\mathrm{LAT}[i,j] = z] = 2^{-2^n} \binom{2^n}{2^{n-1} + z}. \qquad (9.3)$$

*If we consider an n-bit random permutation rather than a function, this probability distribution becomes*

$$\Pr[\mathrm{LAT}[i,j] = 2z] = \frac{\binom{2^{n-1}}{2^{n-2}+z}^2}{\binom{2^n}{2^{n-1}}}. \qquad (9.4)$$

#### 9.1.2.2 Maximum Value

Using the distributions from Theorem 9.1.1 and 9.1.2, we can compute the expected value of the maximum coefficient in both the DDT and the LAT. These are given for some values of $n$ in Table 9.1. The case $n \leq 5$ is not considered because the approximations in those theorems are no longer valid. I have indeed found significant discrepancies between the distribution of the maximum values predicted by theory and its experimental counterpart for those values.

| $n$ | E $(\Delta_s)$ | E $(\mathcal{L}_s/2)$ (permutation) | E $(\mathcal{L}_s/2)$ (function) |
|---|---|---|---|
| 6 | 9.11 | 14.90 | 14.93 |
| 7 | 10.32 | 23.12 | 23.13 |
| 8 | 11.34 | 35.30 | 35.32 |
| 9 | 12.45 | 53.35 | 53.34 |

Table 9.1: The expected value of the maximum coefficients in the DDT/LAT of a random permutation/function.

More generally, we can compute the probability that a bijective S-Box has a maximum coefficient equal to at most $M$ in its DDT or LAT using the following formula:

$$\Pr[\max(c) \leq M] \;=\; \left( \sum_{k=0}^{M} \Pr[c = k] \right)^{(2^n-1)(2^m-1)},$$

where $c$ is the random variable corresponding to a coefficient of one of the tables and $\Pr[c = k]$ is given by Theorem 9.1.1 if the table is a DDT and Theorem 9.1.2 if

| $\delta$ | $\log_2(\Pr[\Delta_s \leq \delta])$ |
|---|---|
| 4 | -1359.530 |
| 6 | -164.466 |
| 8 | -16.148 |
| 10 | -1.329 |
| 12 | -0.094 |
| 14 | -0.006 |

(a) max(DDT) ($\Delta_s$).

| $\ell$ | $\log_2(\Pr[\mathcal{L}_s/2 \leq \ell])$ |
|---|---|
| 22 | -371.609 |
| 24 | -161.900 |
| 26 | -66.415 |
| 28 | -25.623 |
| 30 | -9.288 |
| 32 | -3.160 |
| 34 | -1.008 |
| 36 | -0.302 |
| 38 | -0.084 |

(b) max(LAT) ($\mathcal{L}_s/2$).

Table 9.2: Probability that the maximum coefficient in the DDT/LAT of an 8-bit permutation is at most equal to a certain threshold.

it is an LAT. The value of these probabilities for 8-bit permutations are provided in Table 9.2a for the DDT and Table 9.2b for the LAT.

For example, the probability that the differential uniformity of an 8-bit permutation is at most equal to 6 is equal to about $2^{-164}$. Thus, if an 8-bit S-Box is differentially 6-uniform, it is safe to assume that it has not been picked uniformly at random. We can also rule out the idea that it has been chosen as the best among a feasibly large set of pseudo-random permutations as this set would need to contain about $2^{164}$ such elements. Therefore, using this result, we can say that a differentially 6-uniform 8-bit permutation *must* be the output of a specific generation procedure.

#### 9.1.2.3 The Pair Maximum/Number of Occurrences of the Maximum

As explained in Section 8.2.2 (p. 141), it is often not sufficient to look at the maximum coefficient of the LAT of an S-Box to assess its strength. Therefore, the designer of an S-Box may have tried to optimize not only the maximum value of the coefficients but also the number of occurrences of this maximum.

Therefore, in order to investigate the possibility of an S-Box having been picked uniformly at random, we can also consider the pair $(\max(|C|), \# \max(|C|))$ where $|C|$ is the set of the absolute values of all the coefficients in either table. In other words, we consider the maximum coefficient of the table along with its number of occurrences. Such pairs can be ordered using a simple lexicographic ordering. Two pairs $(M,u)$ and $(M',u')$ where $M, M', u$ and $u'$ are integers can be ordered using the following rule:

$$(M,u) < (M',u') \text{ if and only if } \begin{cases} M < M', & \text{or} \\ M = M', & \text{and } u < u'. \end{cases}$$

This ordering allows us to compute the probability that a random S-Box has a couple $(\max(|C|), \# \max(|C|))$ *at least as good as* a given one. The formula we need to apply to compute this quantity is obtained in a straight-forward way:

$$\Pr[(\max(|C|), \# \max(|C|)) < (M,u)] = \sum_{k=0}^{M} \binom{T}{k} (\Pr[c = u])^k (\Pr[c < u])^{T-k}, \quad (9.5)$$

where $T$ is the size of the table without its first column and row, so that $T = (2^m - 1)(2^n - 1)$, and $\Pr[c < u] = \sum_{i=0}^{u-1} \Pr[c = i]$. In the case of the LAT, we are considering the absolute values of the coefficient. This implies that if $i \neq 0$, then $\Pr[c = i] = \Pr[\text{LAT}[a,b] = i] + \Pr[\text{LAT}[a,b] = -i]$.

Let us consider 8-bit permutations. For the DDT coefficients, we see in Table 9.2a that the probability that their maximum is at most equal to 6 is equal to $2^{-164.5}$. This probability increases very quickly: the probability that it is differentially 8-uniform is equal to $2^{-16.2}$. Looking at the number of coefficients equal to 8 in the DDT of a differentially 8-uniform permutation allows a finer grained investigation of its properties. Figure 9.1 shows the evolution of $\Pr[\Delta_s \leq 8 \text{ and } \#\{\text{DDT}[i,j] = 8\} \leq N_8]$ for increasing values of $N_8$. As expected, we obtain $\Pr[\Delta_s = 6]$ when $N_8 = 0$. The probability then increases and converges towards $\Pr[\Delta_s = 8]$.

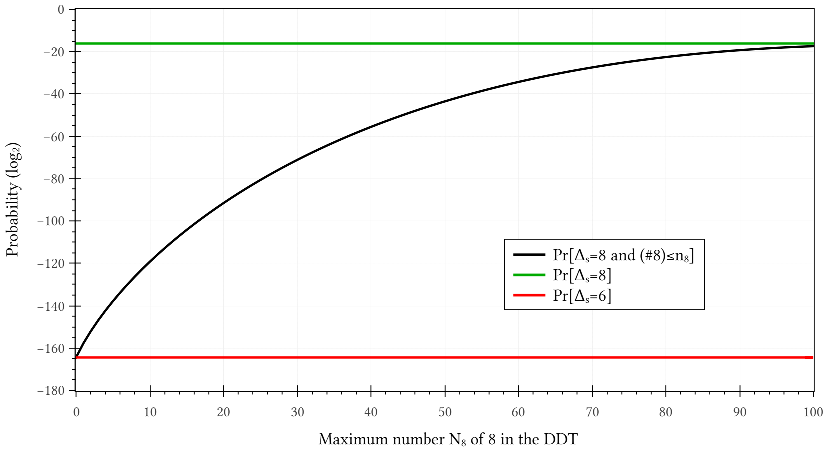


Figure 9.1: Probability that a differentially 8-uniform 8-bit permutation has at most $N_8$ coefficients equal to 8 in its DDT (log scale).

For the LAT, more values of its maximum coefficient can be of interest. The cases corresponding to the maximum coefficients being equal to 26, 28 and 30 are represented in Figure 9.2.

Our main result on the S-Box of Skipjack, which is presented in Section 9.2 (p. 168), is obtained via a direct application of this method.

#### 9.1.2.4 Row/Column Level Patterns

As already mentioned in the end of Section 8.2.1 (p. 139), monomials have a very specific pattern in their DDT: every line contains the exact same distribution of coefficients. Such a pattern is preserved by affine-whitening. This makes monomials very easy to identify. But the usefulness of considering patterns at the row/column level is not limited to monomials.

A direct consequence of Assumption 9.1.1 is the following observation.

**Observation 9.1.1.** *The rows and the columns of the DDT of a random permutation should be independent of one another. The same is true for its LAT.*

This observation is very simple but there are several large classes of functions for which it does not hold. Thus, a violation of this observation not only rules out that

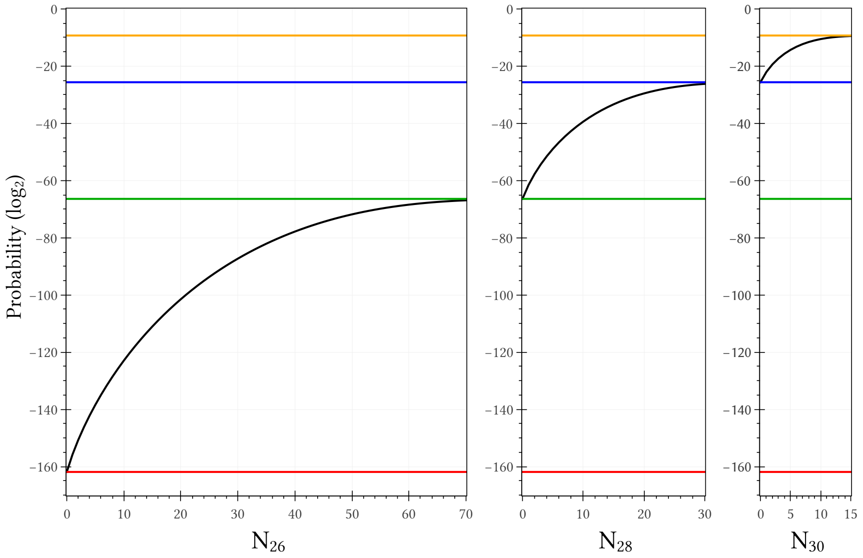


Figure 9.2: Probability that an 8-bit permutation has linearity $\mathcal{L}_s = 2k$ and at most $N_k$ coefficients equal to $k \in \{26, 28, 30\}$ in its LAT (log scale). The probabilities that an 8-bit permutation has a maximum LAT coefficient equal to 24, 26, 28 or 30 are represented in red, green, blue and orange respectively.

the S-Box has been picked uniformly at random, it also gives clear pointers towards its actual structure. An extreme case is that of monomials but other constructions exhibit patterns at the row/column level.

For example, Figure 9.3 shows the variance of the absolute value of the coefficients in each row of the LAT of the S-Box $H$ of the block cipher BelT, which has a pseudo-exponential structure described in Section 8.3.1.3 (p. 148). As we can see, some of the lines have identical and abnormally low variance.

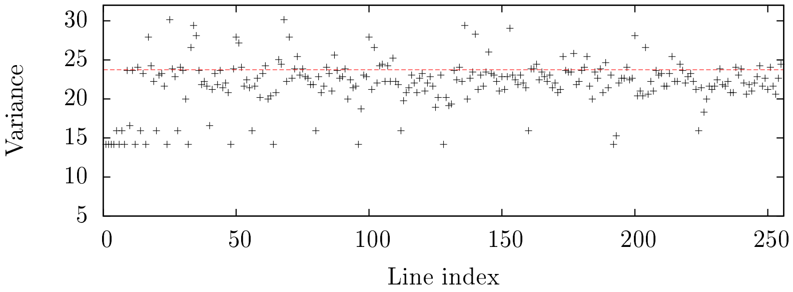


Figure 9.3: The variance of the lines of the LAT (absolute values) of $H$ (BelT). The expected variance is represented with a red dashed line.

For comparison, Figure 9.4 shows the variance of the absolute value of the coefficients in each row of the LAT of an 8-bit S-Box obtained using a Knuth shuffle. It uses the same scale as Figure 9.3.

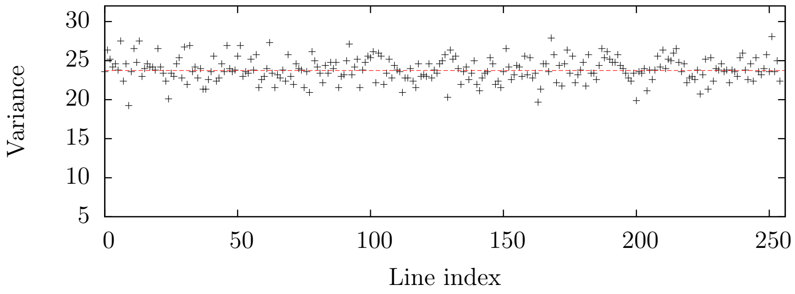


Figure 9.4: The variance of the lines of the LAT (absolute values) of a random S-Box. The expected variance is represented with a red dashed line.

Incidentally, the coordinates of $H^{-1}$ with low-variance Walsh spectra are related to the Boolean functions of Feng et al. [FLY09], though not directly because of the position of the 0.

It is trivial to recover the structure of exponential substitutions.

**Observation 9.1.2.** *If a permutation is an exponential substitution, then applying the Berlekamp-Massey algorithm [Mas69] on each coordinate will give the linear recurrence used to generate the permutation.*

This reverse-engineering method will fail if the S-Box is composed with affine layers (affine whitening). Nevertheless, exponential substitutions have a very strong algebraic structure. It thus comes as no surprise that such permutations have specific patterns at the row level in their LAT.

**Proposition 9.1.1.** *Let $\rho^d$ denote the rotation by $d$ bits to the left. The distribution of the coefficients in lines $a$ and $\rho^d(a)$ of the LAT of an exponential substitutions is identical for any $d$.*

*Proof.* Because of the relation between LAT and Walsh coefficients, we prove the proposition for the Walsh coefficients $W_{a,b}$ of an exponential substitution $s$. Remember that the Walsh coefficients $W_{a,b}$ of an S-Box $s$ are given by:

$$W_{a,b} = \sum_{x \in \mathbb{F}_2^n} (-1)^{a \cdot x + b \cdot s(x)}.$$

The scalar product $a \cdot x$ is equal to $\rho^d(a) \cdot \rho^d(x)$. Using this, we rewrite $W_{a,b}$ as follows:

$$\sum_{x \in \mathbb{F}_2^n} (-1)^{a \cdot x + b \cdot s(x)} = \sum_{x \in \mathbb{F}_2^n} (-1)^{\rho^d(a) \cdot \rho^d(x) + b \cdot s(x)} = \sum_{y \in \mathbb{F}_2^n} (-1)^{\rho^d(a) \cdot y + b \cdot s\left(\rho^{-d}(y)\right)},$$

where $y = \rho^d(x)$. Furthermore, for all $b$ there exists a unique $b'$ such that $b \cdot s\left(\rho^{-d}(y)\right) = b' \cdot s(y)$ for all $y$. Let us prove this fact. This equality obviously holds for $y = 0$. If $y \neq 0$, then the right-hand side is equal to

$$b \cdot s\left(\rho^{-d}(y)\right) = b \cdot \left( \prod_{i=0}^{n-1} \alpha^{y_i 2^{i-d}} \right) = b \cdot \left( \prod_{i=0}^{n-1} \alpha^{y_i 2^{i}} \right)^{2^{-d}},$$

where $x \mapsto x^{2^{-d}}$ is a linear permutation which can be written $x \mapsto M_d \times x$ for some $n \times n$ binary matrix $M_d$. We deduce that $b \cdot s\left(\rho^{-d}(y)\right) = b' \cdot s(y)$ for $b' = (M_d^t \times b)$, where $M_d^t$ is the transpose of $M_d$. As a consequence, the multisets $\{W_{a,b}, \forall b \in \mathbb{F}_2^n\}$ and $\{W_{\rho^d(a),b}, \forall b \in \mathbb{F}_2^n\}$ are identical, which in turn implies the proposition. □

This proposition can be used to distinguish exponential substitutions from random permutations.

If such an S-Box has been composed with an affine layer, i.e. if a permutation $\sigma$ is equal to $A \circ f$ where $A$ is an affine permutation and $f$ is an exponential permutation, then the pattern described in Proposition 9.1.1 is still present: because of Lemma 8.2.4, this composition merely shuffles the columns which leaves the distribution of the coefficients in each line unchanged. Adding another linear layer before $f$ shuffles the rows. While this breaks the rotational pattern, the fact that the rows fall into few distinct classes with regards to the distribution of the coefficients remains unchanged.

**Observation 9.1.3.** *Consider the LAT of a permutation $\sigma = A \circ f \circ B$ where $A$ and $B$ are affine permutations and $f$ is an exponential permutation. The distribution of the coefficients in the different lines are not independent. In fact, it is possible to recover some information about $B$ using the fact that the LAT of $A \circ f$ is such that rows $a$ and $\rho^d(a)$ have the exact same coefficient distribution.*

The LAT of pseudo-exponentials is not invariant through a rotation of the row indices, unlike for exponential substitution. In other words, Proposition 9.1.1 does not hold for pseudo-exponentials. Still, some patterns remain as explained in the following proposition and its corollary.

**Proposition 9.1.2.** *Let $\ell_z$ be the smallest integer such that $2^{\ell_z} > z$. The lines of the LAT of the pseudo-exponential $exp_{\lambda,z}$ with indices $a = k \times 2^{\ell_z}$ for any integer $k$ do not depend on $z$. In particular, they are identical to those of the LAT of an exponential substitution with the base $\lambda$, which corresponds to the case $z = 0$.*

*Proof.* Let $s$ be a pseudo-exponential substitution with exponent $\lambda$ and preimage for zero $z$, i.e. $s = \exp_{\lambda,z}$, and let $f$ be the exponential substitution with the same $\lambda$. We must prove that the following quantities are identical as long as $a = k \times 2^{\ell_z}$:

$$\sum_{x \in \mathbb{F}_2^n} (-1)^{a \cdot x \oplus b \cdot s(x)} = \sum_{x \in \mathbb{F}_2^n} (-1)^{a \cdot x \oplus b \cdot f(x)}$$

First of all, let us rewrite the left-hand side:

$$\begin{aligned}
\sum_{x \in \mathbb{F}_2^n} (-1)^{a \cdot x \oplus b \cdot s(x)} &= \sum_{\overline{x} < z} (-1)^{a \cdot x \oplus b \cdot \lambda^{x \boxplus 1}} + (-1)^{a \cdot z} + \sum_{\overline{x} > z} (-1)^{a \cdot x \oplus b \cdot \lambda^x} \\
&= (-1)^{a \cdot z} + \sum_{0 < \overline{x} \leq z} (-1)^{a \cdot (x \boxminus 1) \oplus b \cdot \lambda^x} + \sum_{\overline{x} > z} (-1)^{a \cdot x \oplus b \cdot \lambda^x} \\
&= (-1)^{a \cdot z} + \sum_{x \neq 0} (-1)^{a \cdot \phi_z(x) \oplus b \cdot \lambda^x},
\end{aligned}$$

where $\phi_z(x) = x$ if $\overline{x} > z$ and $\phi_z(x) = x \boxminus 1$ otherwise, where $\overline{x} = \sum_{i=0}^{n-1} x_i 2^i$. It is sufficient to prove the proposition to show that if $\overline{a} = k \times 2^{\ell_z}$ then $a \cdot \phi_z(x) = a \cdot x$ for all $x \neq 0$.

If there exists $i$ in $[0, \ell_z - 1]$ such that $x_i = 1$ , then the carry from the subtraction cannot propagate to $x_j$ and the other bits with higher weight. Therefore, it is necessary for $a \cdot \phi_z(x)$ to be different from $a \cdot x$ that $x_0 = \ldots = x_{\ell_z - 1} = 0$. If it is the case, then $\overline{x} = m \times 2^{\ell_z}$. However, it also holds that $\phi_z(x) = x$ because $x \geq 2^{\ell_z} > z$. Thus, for all $x \neq 0$ and all $a = k \times 2^{\ell_z}$, $a \cdot \phi_z(x) = a \cdot x$. The proposition follows. □

Proposition 9.1.2 does not work when $z = 2^n - 1$, i.e. for exponentials studied in [HN10]. If $z$ is this large, no LAT row index can be higher.

As the LAT of a pseudo-exponential shares some of its lines with an exponential substitution, these lines also share the patterns presented in Proposition 9.1.1.

**Corollary 9.1.1.** *The distributions of the coefficients in lines with indices $a2^d$ and $a2^{d'}$ of the LAT of a pseudo-exponential substitutions are identical for any $a$ and any $d, d'$ such that $z < \min(2^d, 2^{d'})$ and $\max(a2^d, a2^{d'}) < 2^n$.*

This partially explains the patterns visible in Figure 9.3. However, the LAT of the S-Box $H$ of BelT contains stronger patterns than those which can be deduced from Corollary 9.1.1. Indeed, rows with indices $2^i$ and $3 \times 2^i$ all shared the same distribution of absolute value of coefficients.

These patterns were experimentally confirmed for other such pseudo-exponential, so that we make the following conjecture.

**Conjecture 9.1.1.** *Let $\mathcal{L}$ and $\mathcal{L}'$ be the LAT of the exponential substitution with base $\lambda$ and $exp_{\lambda,z}$ respectively. Then, for all i, the distribution of the absolute value of the coefficients at line $2^i$ and $3 \times 2^i$ is the same in $\mathcal{L}$ and $\mathcal{L}'$, regardless of $z$.*

The proof of Proposition 9.1.1 cannot be simply adapted to prove this conjecture. Indeed, although the distribution of the absolute value of the coefficients is preserved, we observed that neither the sign nor the position of the coefficients are identical if $a \neq k \times 2^{\ell_z}$.

**Open Problem 9.1.1.** *Is Conjecture 9.1.1 true?*

#### 9.1.2.5 Alternative DDT Definitions

In some contexts, in particular when the round function uses modular addition (see e.g. the old "GOST" cipher, Magma [Dol10a] or BelT [Bel11]), it is natural to consider patterns using different operations in the input and output. For example, we can count the number of solutions of the following equation for some S-Box $s$ and for all $a, b$ in $\mathbb{F}_2^8$:

$$s(x \boxplus a) \oplus s(x) = b,$$

where $\oplus$ denotes a bit-wise exclusive-or and $\boxplus$ denotes addition modulo $2^8$. The results obtained for different S-Boxes are given in Table 9.3, where the cases $a = 0$ and $b = 0$ are ignored.

There exists on average one $x$ such that $s(x \boxplus a) \oplus s(x) = b$ so we model the number of solutions of this equation for a given pair $(a, b)$ as a sample from a Poisson distribution with parameter 1. The corresponding expected number of solutions are listed in the "Theoretical" column of Table 9.3. "Random" corresponds to an 8-bit permutation generated using a Knuth shuffle.

As we can see, the maximum number of solutions for BelT is far too small. As explained before in Section 8.3.1.3 (p. 148), its S-Box is based on a finite field exponentiation and having a low probability for transitions of the form $H(x \oplus a) = H(x) \boxplus b$ was one of its design criteria.

| # solutions $N$ | $\#\{(a,b), \#\text{solutions} = N\}$ | | | | |
|---|---|---|---|---|---|
| | BelT | Kuznyechik | AES | Random | Theoretical |
| 0 | 11175 | 23252 | 22270 | 23582 | 23921.36 |
| 1 | 42498 | 24466 | 25486 | 24148 | 23921.36 |
| 2 | 11274 | 12414 | 12968 | 12271 | 11960.68 |
| 3 | 78 | 3784 | 3490 | 3810 | 3986.89 |
| 4 | 0 | 945 | 685 | 965 | 996.72 |
| 5 | 0 | 132 | 110 | 204 | 199.34 |
| 6 | 0 | 30 | 14 | 37 | 33.22 |
| 7 | 0 | 2 | 2 | 6 | 4.75 |
| 8 | 0 | 0 | 0 | 2 | 0.59 |

Table 9.3: The distribution of the number of solutions of $s(x \boxplus a) \oplus s(x) = b$ depending on $a$ and $b$ for different S-Boxes and for a Poisson distribution.

## 9.2 Detailed Results on Skipjack

Skipjack is a block cipher released in [U.S98] which was intended for use inside the *Clipper chip*. This device was supposed to encrypt communications in such a way as to provide a key escrow allowing American government employees to eavesdrop on any communication encrypted with it. The announcement of this device caused the first *Crypto War*: the American government argued for the need of such an escrow from fear of "going dark", that is, of not being able to (lawfully) eavesdrop on the conversations of criminals; while privacy advocates pointed out that an algorithm with a key escrow cannot be secure and that criminals would simply use other forms of encryption without such a "feature".

Quickly, several attacks against the mode of operation used in the chip were published [Bla94, FY95]. They allowed an easy bypass of the key escrow, thus making the chip useless. These attacks, combined with the widespread use of open-source cryptography, turned the Clipper chip into a complete fiasco: only one device[2] ever used it.

When Skipjack was released, only its specification was published. The rationale behind its design was — and in fact still is — kept secret by the NSA. In this section, I present how Alex Biryukov and myself managed to recover some of the design criteria of its S-Box.

### 9.2.1 Overview of Skipjack

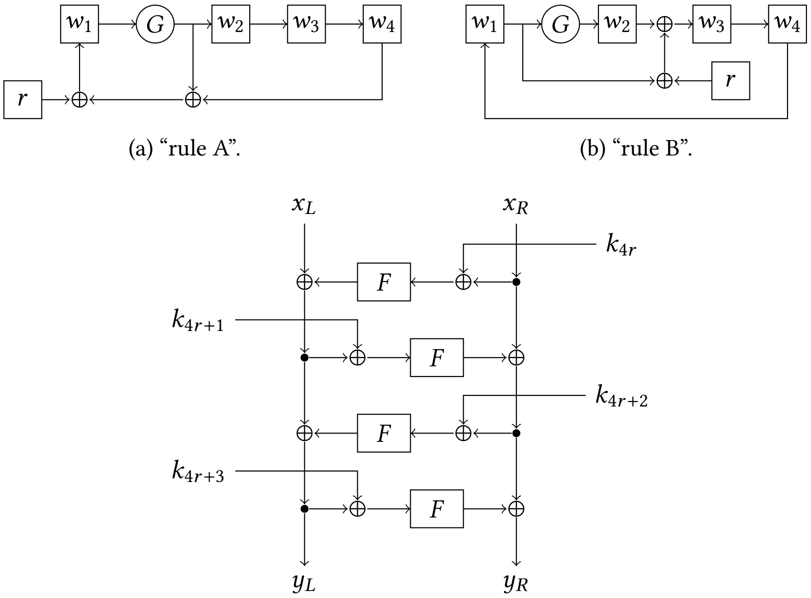
Skipjack has a block size of 64 bits and key size of 80 bits. A high level view of the two types of rounds used during encryption, called "rule A" and "rule B", are provided in Figure 9.5a and 9.5b. The $G$ function is given in Figure 9.5c and the so-called "F-Table" used inside the $G$ function is in fact an 8-bit bijective S-Box specified using only its LUT (see Table 9.4).

Encryption consists of 8 rounds of rule A, 8 rounds of rule B, 8 rounds of rule A and finally 8 rounds of rule B. A round counter, which we denote $r$, is used in both

[2] Namely the AT&T TSD-3600 telephone encryptor, see http://www.cryptomuseum.com/crypto/att/tsd3600/index.htm.

rules. The key schedule is extremely simple: 32 bits of the master key are extracted in a rolling manner in each round, i.e. 8-bit words $k_0, ..., k_3$ are used in round 1, $k_4, ..., k_7$ in round 2, $k_8, k_9, k_0, k_1$ in round 3, etc. The interested reader may refer to the official specification [U.S98] or to the best attack on the cipher [BBS05], an impossible differential attack leveraging its particular round structure.

In previous works, other authors tried to discover the design criteria of Skipjack e.g. in [KRW99, KW01]. For example, Knudsen and Wagner showed that rule A and rule B are almost the inverse of one another, which means that encryption and decryption offer the same level of security. They also show that starting with rule B instead of rule A would weaken the cipher. However, these studies focus on the overall structure of the algorithm, not the specifics of its S-Box. Another work on Skipjack [BBD+99] lists the differential and linear properties of $F$ without attempting to decompose it.



(a) "rule A".

(b) "rule B".

(c) The $G$ function.

Figure 9.5: The structure of the Skipjack block cipher.

The distribution of the coefficients in the DDT of Skipjack is summarized in Table 9.5 along with the theoretical distribution from Theorem 9.1.1. As we can see it is differentially 12-uniform, the same as you would expect from a random permutation, which is surprising since minimizing the differential uniformity is usually one of the corner stones of provable resilience against differential attacks.

### 9.2.2 The Linear Properties are Too Good to be True

Figure 9.6 contains the distribution of the value of the coefficients of the LAT (minus the first line and column) along with the theoretical proportions obtained by applying Theorem 9.1.2.

The probability that a random 8-bit permutation has a maximum LAT coefficient

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

Table 9.4: Skipjack's S-Box, $F$, in hexadecimal notation. For example, $F($`0x7a`$)$ = `0xd6`.

| Coefficient | Number | Proportion (%) in $F$ | Poisson(1/2) (%) |
|---:|---:|---:|---:|
| 0 | 39104 | 60.14 | 60.65 |
| 2 | 20559 | 31.62 | 30.33 |
| 4 | 4855 | 7.467 | 7.582 |
| 6 | 686 | 1.055 | 1.264 |
| 8 | 69 | 0.106 | 0.158 |
| 10 | 5 | 0.008 | 0.016 |
| 12 | 2 | 0.003 | 0.002 |

Table 9.5: Distribution of the coefficients in the DDT of $F$.

equal to 28 is given in Table 9.2b and is equal to

$$P[\max(\text{LAT}) \le 28] = 2^{-25.62}.$$

This probability is low but it would be feasible to generate a set of about $2^{26}$ random 8-bit permutations and compute the LAT for each of them as this computation costs about $8 \times 2^{2\times 8} = 2^{19}$ simple operations. In such a set, the best S-Box $s$ should verify max(LAT) = 28. However, we can better estimate the number of S-Boxes that would need to be generated to imitate the linear properties of $F$ by taking into account the number of occurrences of 28 in its LAT. Coefficients with absolute value 28 occur 3 times in the LAT of $F$. As we can see in Figure 9.2, this quantity is close to $2^{-55}$.

More rigorously, we compute the probability to have at most $u$ coefficients equal to 28 in the LAT of a permutation picked uniformly at random from the set of all 8-bit permutations using Equation (9.5). If we let $p(2i) = \Pr[\text{LAT}[a,b] = 2i]$, then this probability is equal to $P_{28,u}$ where

$$P_{28,u} = \sum_{j=0}^{u} \left[ \binom{(2^8-1)^2}{j} \big(p(28) + p(-28)\big)^j \Big( \sum_{k=-13}^{13} p(2k) \Big)^{(2^8-1)^2 - j} \right].$$

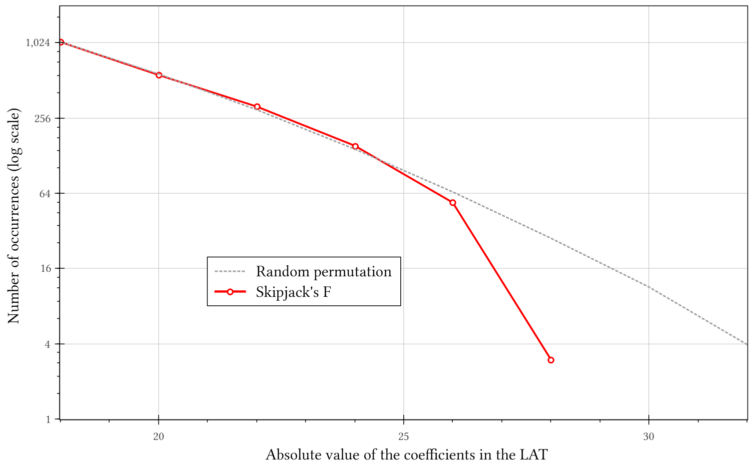


Figure 9.6: Coefficients of the LAT of $F$ and the expected distribution for a random 8-bit permutation.

For $u = 3$, the case of the F-table of Skipjack, we find:

$$P_{28,3} = 2^{-54.4}.$$

The probability that a random permutation has linear properties comparable to those of Skipjack's $F$ is thus equal $2^{-54.4}$. Hence, we make the following claims:

- $F$ was not chosen uniformly at random among the set of all 8-bit permutations,
- the designers of Skipjack did not generate many 8-bit random permutation to then pick the best according to some criteria as they would need to have generated at least about $2^{55}$ S-Boxes,
- the method used to build $F$ improved its linear properties.

When I presented this work at CRYPTO'15, a member of the audience pointed out that $2^{55}$ is, in some sense, not that large. It is true that such an amount of computing power is within reach nowadays. However, Skipjack was designed in the end of the 1980's/beginning of the 1990's. Furthermore, and I find this argument more compelling, a hypothetical designer would need to generate $2^{55}$ S-Boxes and then, for each of them, compute the LAT. This operation requires about $8 \times 2^{2 \times 8} = 2^{19}$ steps in this case, which brings the total cost of such a brute-force search to roughly $2^{74}$ operations. While we can only speculate about the feasibility of such a computation for an institution as powerful as the NSA, we must keep in mind that, as shown in the following section, it is extremely easy to design an algorithm which obtains similar results in negligible time.

While it is impossible at this stage to rule out the idea that $F$ was picked from a very large set of random 8-bit S-Boxes, the tremendous cost of such a search would not be justified by its end result: while the linear properties of $F$ are indeed observably better than those of a random S-Box, they remain less than impressive compared to those of a dedicated structure like the inverse function.

I therefore still think that this S-Box was somehow engineered and not merely picked from a random set.

### 9.2.3 A Possible Design Criteria

We tried to create an algorithm capable of generating S-Boxes with linear and differential properties similar to those of $F$. It turns out that such an algorithm is both efficient and easy to write. First, we introduce a quantity we denote $R(f)$ and define as follows:

$$R(f) = \sum_{\ell \geq 0} N_\ell \cdot 2^\ell,$$

where $N_\ell$ counts the number coefficients with absolute value $\ell$ in the LAT of $f$: $N_\ell = \#\{\text{LAT}[i,j] \in (\text{LAT of } f), |\text{LAT}[i,j]| = \ell\}$.

Algorithm 9.1 starts from a random 8-bit permutation $s$ and returns a new permutation $s'$ such that $R(s') < R(s)$ and such that $s'$ is identical to $s$ except for two entries $x$ and $y$ which are swapped: $s'(x) = s(y)$ and $s'(y) = s(x)$. It works by identifying one of the highest coefficients in the LAT, removing it through swapping two entries, and checking whether $R(s)$ was actually improved. This algorithm can be used in two different ways: either we keep iterating it until it reaches a point at which no swap can improve $R(s)$ or we stop as soon as $R(s)$ is below an arbitrary threshold.

**Algorithm 9.1** `Improve-`$R$(): optimizing the linear properties of an S-Box.
**Input:** S-Box $s$;
**Output:** S-Box $s'$ with improved $R()$

```
$\mathcal{L}$ := LAT of $s$
Find $a, b$ such that $|\mathcal{L}[a,b]| = \mathcal{L}_s/2$
$\ell$ := empty list
for all $x \in \mathbb{F}_2^8$ do
    if $a \cdot x = b \cdot f(x)$ then
        Append $x$ to $\ell$
    end if
end for
for all $(x,y) \in \ell \times \ell, x \neq y$ do
    $s' = s$ ; $s'(x) = s(y)$ ; $s'(y) = s(x)$
    if $R(s') < R(s)$ then
        return $s'$
    end if
end for
return Fail
```

We implemented both variants. For the second one, we stop when $R(s) < 10^{10}$ because $R(F) \approx 10^{9.92} \approx 2^{33.1}$. We now denote $N_\ell^T$ the average number of coefficients with absolute value $\ell$ in the table $T$ taken over several S-Boxes obtained in different ways, where $T \in \{\text{LAT}, \text{DDT}\}$. For the LAT, $\log_2(N_\ell^{\text{LAT}})$ is given in Table 9.6b and in Figure 9.7; for the DDT it is in Table 9.6a. In those tables:

- "Rand." corresponds to the average over 200 bijective 8-bit S-Boxes picked uniformly at random;
- "$F$" to the distribution for the S-Box of Skipjack;
- "$F$-like" to the average over 100 S-Boxes obtained using `Improve-`$R$() and stopping when $R(s) < 10^{10}$; and

- “best” to the average over 100 S-Boxes obtained using `Improve-`$R$() and stopping only when it fails.

Figure 9.7 also contains the average coefficient distribution in the LAT of 100 S-Boxes generated using a variant of Algorithm 9.1. Instead of optimizing $R(s)$, it merely improves the pair $(\mathcal{L}_S/2, N_{\mathcal{L}_S/2})$. That is, it tries to minimize the maximum value in the LAT and its number of occurrences without considering other coefficients. The program was made to stop when reaching the value corresponding to $F$, namely $(28, 3)$.

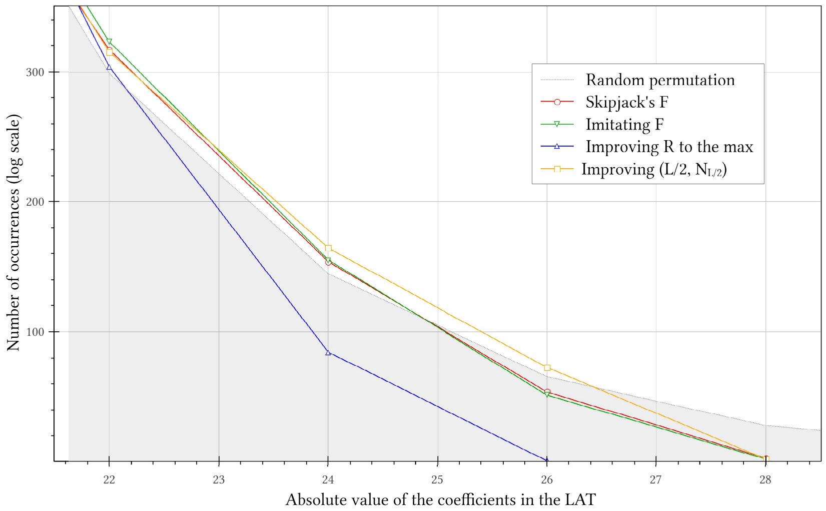


Figure 9.7: Distribution of the coefficients in the LAT of $F$ and of some outputs of `Improve-`$R(s)$.

| $\ell$ | Rand. | $F$ | $F$-like | best $R()$ |
|---|---|---|---|---|
| 0 | 15.265 | 15.246 | 15.250 | 15.227 |
| 2 | 14.270 | 14.327 | 14.314 | 14.380 |
| 4 | 12.277 | 12.245 | 12.257 | 12.210 |
| 6 | 9.693 | 9.422 | 9.492 | 9.126 |
| 8 | 6.701 | 6.109 | 6.198 | 5.265 |
| 10 | 3.374 | 2.322 | 2.287 | 0.714 |
| 12 | -0.059 | 1.000 | -1.786 | -5.059 |
| 14 | -4.059 | - | -5.059 | - |

(a) DDT

| $\ell$ | Rand. | $F$ | $F$-like | best $R()$ |
|---|---|---|---|---|
| 20 | 9.164 | 9.147 | 9.230 | 9.311 |
| 22 | 8.220 | 8.308 | 8.336 | 8.247 |
| 24 | 7.173 | 7.267 | 7.280 | 6.400 |
| 26 | 6.041 | 5.755 | 5.688 | 0.000 |
| 28 | 4.826 | 1.585 | 1.157 | - |
| 30 | 3.506 | - | - | - |
| 32 | 2.146 | - | - | - |
| 34 | 0.664 | - | - | - |

(b) LAT

Table 9.6: Distribution of $\log_2(N_\ell^T)$ for $T \in \{\text{DDT}, \text{LAT}\}$ and for different S-Boxes.

Using `Improve-`$R()$ with an appropriate threshold allows us to create S-Boxes with both linear and differential properties very close to $F$. However, in order to achieve this, we need to choose a threshold value computed from $F$ and which does not correspond to anything specific. In fact, to the best of our knowledge, the quantity $R(s)$ does not have any particular importance unlike for instance the linearity $\mathcal{L}_s$. Still, replacing $R(s)$ by $\mathcal{L}_s$ or the pair $(\mathcal{L}_s, \#\{(i,j), |\text{LAT}[i,j]| = \mathcal{L}_s/2\})$ yields S-Boxes which are very different from $F$. Such S-Boxes indeed have a value of $N_{\mathcal{L}_s/2-2}$ observably higher than in the random case and much higher than for $F$, as can be seen in Figure 9.7.

While our definition of $R(s)$ may seem arbitrary, it is the only one we could find that leads to linear properties similar to those of $F$. For instance it may have been tempting to base $R(s)$ on the square of $\ell$ which is used when computing the correlation potential of a linear trail, a quantity useful when looking for linear attacks. We would thus define $R(s) = \sum_{\ell \geq 0} N_\ell \ell^2$. However this quantity is worthless as an optimization criteria since it is constant: Parseval's equality on the Walsh spectrum of a Boolean function imposes that the sum of the $(\text{LAT}[i,j])^2$ over each column is equal to $2^{2n-2}$.

In summary, we have found new non-random properties of the S-box of Skipjack which are improving its strength against linear cryptanalysis and we developed and algorithm which could be used to generate such S-boxes.

### 9.2.4 Public Information About the Design of Skipjack

The only information indirectly published by the NSA on Skipjack corresponds to an "Interim Report" [BDK+93] written by external cryptographers and it contains no information on the specifics of the design. The most relevant parts of this report as far as the S-Box is concerned are the following ones.

> SKIPJACK was designed to be evaluatable [...]. In summary, SKIPJACK is based on some of NSA's best technology. Considerable care went into its design and evaluation in accordance with the care given to algorithms that protect classified data.

Furthermore, after the "leakage" of an alleged version of Skipjack to Usenet further discussed in Section 9.2.5 (p. 175), Schneier replied with a detailed analysis of this cipher [Sch95] which contained in particular the following quote indicating that the S-box was changed in August 1992.

> The only other thing I found [through documents released under FOIA] was a SECRET memo. [...] The date is 25 August 1992. [...] [P]aragraph 1 reads:
>
> 1. (U) The enclosed Informal Technical Report revises the F-table in SKIPJACK 3. No other aspect of the algorithm is changed.

Note also that the first linear cryptanalysis of DES [Mat94] had not been published yet in August 1992 when the F-Table was changed. At CRYPTO'90 [GC91], Gilbert *et al.* suggested the use of linear equations to help with key guessing in a differential cryptanalysis against FEAL. This block cipher was later attacked at CRYPTO'91 [TCG92] and EUROCRYPT'92 [MY93] using directly some linear equations involving plaintext, ciphertext and key bits. We can but speculate about a connection between these papers and the change of Skipjack's S-Box.[3]

[3] I tried asking NSA cryptographers I met during conferences but they claimed, quite plausibly, that this algorithm was too old for them to know about the specifics of its design.

### 9.2.5 What About S-1?

The S-1 algorithm, alleged to be Skipjack in [Ano95], gives us no information about the F-table of Skipjack. However, I think it is worth investigating its possible relationship with Skipjack now that this algorithm is known.

The Usenet post which prompted Schneier's answer was the publication by an anonymous member of `sci.crypt` of the C implementation of an algorithm which they claimed to be Skipjack at a time when this algorithm was still classified [Ano95]. The algorithm described, "S-1", turned out to be different from Skipjack as we know it.

It is an unbalanced Feistel network encrypting 64-bit blocks divided into four 16-bit words using an 80-bit key. Its round function is represented in Figure 9.8 where different line widths indicate different data-path widths. This algorithm uses 16-, 8-, 4- and 2-bit data paths. The Feistel function uses four distinct functions $F_i : \mathbb{F}_2^8 \rightarrow \mathbb{F}_2^4$ which are stored in a so-called "F-table", the same term as in Skipjack. The content of one branch is sent through a function $G$ which outputs a 2-bit counter $j$. This counter is used to "rotate" the ordering of the functions $F_i$: $F_j$ is applied on the first byte, $F_{(j+1) \mod 4}$ on the second, etc. In a way, the F-table can be seen as a rotor of S-Boxes which rotates depending on the content of one branch. The content of the other two branches is then sent through the F-table and xored with the last branch. Since the $F_i$ functions have an output half as large as their input, the function is well defined. An encryption uses 32 rounds.

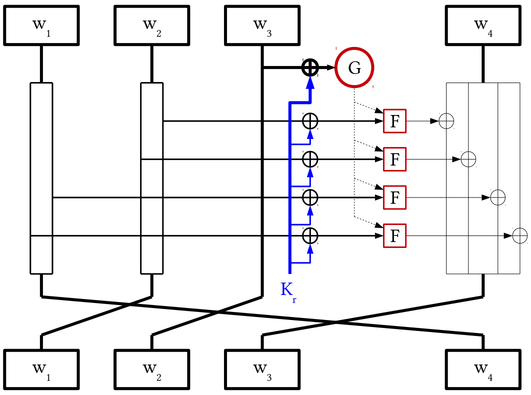


Figure 9.8: The structure of the "S-1" cipher.

The round-key is xored just before the computation of the round function. The key schedule is very simple: the round-key of round $i$ consists of 6 bytes $\mathrm{rk}_0^i, \ldots \mathrm{rk}_5^i$ where the least significant and most significant 4-bit nibbles are equal to:

$$\begin{cases} \mathtt{lsn}(\mathrm{rk}_j^i) = F_0(K[6i+j+s(j+0)]) \ \oplus \ F_1(K[6i+j+s(j+1)]) \\ \mathtt{msn}(\mathrm{rk}_j^i) = F_2(K[6i+j+s(j+2)]) \ \oplus \ F_3(K[6i+j+s(j+3)]) \ , \end{cases}$$

where $K$ is the 80-bit master key, the index of the master key byte is taken modulo 10 and where $s = \{5,8,3,1,4,0\}$. Note that $s$ is the concatenation of $s' = \{5,8,3,1,4\}$ and $\{0\}$ where $s'$ is such that $s'[i+2] = s[i] + s[i+1] \mod 10$. However, the most important property of this key schedule is its periodicity: just like in the actual Skipjack, the round keys are identical every 5 rounds!

What can we say about S-1? Is it a hoax? Since Schneier was able to get the report quoted above around the time of the publication of S-1, it could be that the anonymous poster found out about the term “F-table”, the existence of two previous versions of Skipjack and the block/key size through the same mean. This observation echoes the following remark from Schneier’s analysis:

> The hoaxer knew about Blaze’s and my MacGuffin paper and that we thought SKIPJACK was a 48:16 UFN. We made no secret about this, and our paper is on Blaze’s web page. The hoaxer knew to use the term F-table. I haven’t shown many people what I found in EPIC’s documents, so the hoaxer either had to look through them himself or get them by some other means (maybe an independent FOIA request).
>
> [...]
>
> So, maybe it’s SKIPJACK. It has a 64-bit block size and an 80-bit key size. It’s a 48:16 UFN with 32 rounds (or shifts, or whatever). And it has an F-table. This is really interesting, because the structure really is an S-box. Everyone knows it’s an S-box, and it makes no sense for a hoaxer to call it something else. But in S-1 it’s called an F-table. (I think this is very significant, but others find it less convincing.)

The key aspect about S-1 which links it to Skipjack as we know it is the key schedule. Schneier was unimpressed by it:

> The hoaxer knew enough to make a design that included [...] a bizarre key schedule. [...] The key schedule is hopelessly flawed (David Wagner posted an attack to sci.crypt). [...] But the key schedule is just plain wrong.

The attack he refers to is one of the first two[4] slide attacks [Wag95]. It exploits the fact that the key repeats itself every 5 rounds without having any round constant in the round function.

And yet, although the periodicity of the key schedule is a flaw in the context of S-1, the counter added in the round function of Skipjack allows the use of a similar repeating key schedule with a period of 5 rounds.

The conclusion of Shneier’s analysis is below (emphasis mine):

> And maybe the code originally didn’t have an 80-bit key schedule. Maybe it had a longer key schedule. The poster then modified this key schedule to make it look more like SKIPJACK. (This might also explain the bug in the code, which might not be a bug if it still had the original key schedule.)
>
> Which leaves us precisely nowhere. The most likely explanation is that it is a hoax, but I am hard-pressed to imagine a hoaxer with the requisite combination of skills, resources, and attitude. I also don’t believe that it is SKIPJACK. It might be a preliminary design for SKIPJACK, but **if both the key schedule and F-table entries are wrong**, we really haven’t

[4] The other of the first two slide attacks was independently found by Biryukov to break the TREYFER lightweight block cipher [Yuv97]. After a suggestion of Bruce Schneier — who also proposed the name “slide attack”, Biryukov and Wagner later collaborated to write the first academic paper on slide attacks [BW99, Bir17].

> learned anything. If we suddenly discovered that unbalanced S-boxes are far superior to balanced ones, then all [bets] are off.

We now know that its key schedule is in fact *not* wrong, it is close to the one actually used in Skipjack. Thus, I tend to agree with Schneier: I strongly suspect that S-1 is indeed Skipjack-1 (recall that Skipjack as we know it is actually Skipjack-3).

Unfortunately, I did not manage to find any evidence that the functions in the F-table of S-1 have not been picked uniformly at random: their algebraic degree, linearity and differential uniformity are on par with what would be expected from a random S-Box mapping $\mathbb{F}_2^8$ to $\mathbb{F}_2^4$.

## 9.3 Detailed Results on Chiasmus

The S-Box of Chiasmus is based on the finite field inverse. The use of the inverse function has not been made public by the designers of Chiasmus to the best of my knowledge. This is to be expected as this cipher was not supposed to be public in the first place, it was designed for internal use within the German government by the *Federal Office for Information Security*. The structure of the cipher was successfully reverse-engineered from an encryption program by two independent groups: Schejbal *et al.* [STW13] and Schuster [Sch14]. These revealed the two S-Boxes used, one being the functional inverse of the other. The proximity between the S-Boxes of Chiasmus and the Aes was already noted by Schuster who however ruled out an identical structure, as explained in his slides:

> - I.e. there exists no combination of any inversion in $GF(2^8)$ that in combination with any affine mapping in $GF(2)^8$ constructs the Chiasmus s-boxes. [...]
> - Probably there is just another affine mapping before the inversion is applied.

This intuition was correct: the S-Box of Chiasmus is indeed built from the inverse function composed with two different affine mappings. These could be recovered using the algorithm of Biryukov et al. [BDBP03] for affine equivalence in time $O\left(n^3 2^{2n}\right)$ but, in the case where the basic building block is a monomial differentially $\Delta$-uniform and locally $(\Delta - 2)$-uniform, such as the multiplicative inverse in $\mathbb{F}_{2^n}$ for even $n$ and $\Delta = 4$, we can bypass the xor of the input and output constants and thus use the more efficient algorithm for linear equivalence in time $O\left(n^3 2^n\right)$ presented in the same paper. The trick is based on the following lemma.

**Lemma 9.3.1.** *Let $x \mapsto x^e$ be a differentially $\Delta$-uniform and locally differentially $(\Delta - 2)$-uniform permutation of $\mathbb{F}_{2^n}$ and let $s : x \mapsto B\left((A(x) \oplus c)^e\right) \oplus d$, where $A$ and $B$ are linear permutations and $c, d$ are constants from $\mathbb{F}_{2^n}$. Let $(a_i, b_i)$ for $1 \leq i < 2^n$ be the set of the indices such that $\mathrm{DDT}[a_i, b_i] = \Delta$ in the* ddt *of $s$. Then the indices are linked by the following relation:*

$$b_i = B\left((A(a_i))^e\right) .$$

*In other words, by looking at the highest coefficients in the* ddt *of $s$ we can bypass the constant additions.*

*Proof.* Because $x \mapsto x^e$ is a monomial, the coefficients in its DDT $D_e$ are such that $D_e[a,b] = D_e[1, ba^{-e}]$ because

$$(x+a)^e + x^e = b \text{ is equivalent to } \left(\frac{x}{a}+1\right) + \left(\frac{x}{a}\right)^e = \frac{b}{a^e}.$$

Furthermore, the DDT $D_s$ of $s$ can be obtained from $D_e$ using the fact that $D_s[a,b] = D_e[A(a), B^{-1}(b)]$, as explained in Lemma 8.2.2. We deduce that the DDT of $s$ can be expressed as $D_s[a,b] = D_e[A(a), B^{-1}(b)]$ which is equal to $D_e[1, B^{-1}(b)\,(A(a))^{-e}]$. Since the only index $a$ such that $D_e[1,a] = \Delta$ is 1 (by definition of local differential $(\Delta-2)$-uniformity), we deduce that if $D_s[a_i,b_i] = \Delta$ then $B^{-1}(b_i)\,(A(a_i))^{-e} = 1$, so that

$$b_i = B\Big(\,(A(a_i))^e\,\Big),$$

meaning that $a_i \mapsto b_i$ is the same as $s$ minus its constant additions. □

Incidentally, this gives another criteria to know if an 8-bit S-Box has been built using a finite field inverse. By applying first this method and then the linear equivalence algorithm of [BDBP03], we can find the affine mappings $x \mapsto A(x) \oplus a$ and $x \mapsto B(x) \oplus b$ in time $O(2^{2n})$, as it is the time taken to compute the DDT, instead of $O(n^3 2^{2n})$.

I applied this method[5] to Chiasmus and recovered many such functions. This is not surprising as the existence of one decomposition implies that of many others because constant multiplications and squarings can be applied to the first linear mapping — provided that their inverses are applied to the second one — without breaking the equivalence. I brute-forced all such linear mappings hoping to find a “good-looking” decomposition. Unfortunately, none of them looked particularly structured.

The S-Box $C$ of Chiasmus can be computed as

$$C(x) \;=\; B\Big(\,(A(x) \oplus a)^{-1}\,\Big) \oplus b,$$

where the inversion is in $\mathbb{F}_2[X]/X^8 + X^4 + X^3 + X + 1$, $a = $ `0x8f`, $b = $ `0x59` and

$$A = \begin{bmatrix} 1&0&0&0&0&0&0&0\\ 1&0&0&0&0&1&0&0\\ 0&0&0&0&1&1&0&0\\ 1&0&0&1&1&0&1&1\\ 0&1&1&0&0&0&0&0\\ 0&1&1&1&1&1&1&0\\ 1&1&1&0&0&0&1&1\\ 0&0&1&0&1&1&1&1 \end{bmatrix} \text{ and } B = \begin{bmatrix} 0&0&0&1&0&1&0&0\\ 0&0&1&1&1&1&1&0\\ 0&1&0&1&0&0&1&0\\ 1&0&1&0&1&0&1&0\\ 1&1&0&1&1&0&0&1\\ 1&1&1&1&0&0&0&0\\ 0&0&0&0&0&1&1&1\\ 1&0&0&1&0&1&0&1 \end{bmatrix}.$$

## 9.4 Application to Other S-Boxes

Using Theorems 9.1.1 and 9.1.2, we can compute the probability that an S-Box picked uniformly at random is at least as good as another one from the differential and linear perspective. If this probability is close to 1, it means that the S-Box is not better than a random one from a differential or linear stand-point. Conversely, if this probability is low, it means that the S-Box is much better.

[5] I thank Aleksei Udovenko for providing his implementation of the linear equivalence algorithm from [BDBP03].

This provides a mean to compare the different constructions described in Section 8.3 (p. 146). The results are summarized in Table 9.7. I have only considered 8-bit S-Boxes because these are very common and because smaller ones usually do not have a specific structure other than that of the finite field multiplicative inverse. The probability that a random S-Box $r$ has a pair $(\Delta_r, \#\{(i,j), \text{DDT}[i,j] = \Delta_r\})$ under that of a particular S-Box $s$ is denoted $2^{\ell_{\text{dif}}}$. The counterpart for the LAT is denoted $2^{\ell_{\text{lin}}}$. All the properties listed in Table 9.7 are invariant under affine-equivalence. The only exception is the fact that a function is an involution.

We can see that the S-Boxes of Iceberg and Khazad have better differential properties than those of Clefia, Twofish and Crypton 0.5. This is probably due to the former using 3 layers of 4-bit S-Boxes while the latter only use 2. Interestingly, the S-Box $S_0$ of Clefia has much better linear properties than those of Twofish and Crypton 0.5. The linear layer inside $S_0$ is based on an MDS matrix mapping $(\mathbb{F}_{2^4})^2$ to itself, thus offering better diffusion than the other two.

There is also a significant discrepancy between the properties of the S-Box of CS-cipher and other Feistel-based constructions, in particular that of Zuc ($s_0$) and the one of iScream. A possible reason is that the designers of the CS-cipher used a Feistel function with a simple structure while Zuc uses APN functions.

It is also interesting to see that the S-Box of Anubis was picked as the best in a set of roughly $600 \times 10^6 \approx 2^{29.2}$ random involutions, which is very close to the value of $2^{-\min(\ell_{\text{dif}}, \ell_{\text{lin}})} = 2^{28.7}$. This confirms that the approach described in Section 9.1.2.3 (p. 162) provides an accurate method for estimating the probability that a random permutation has differential and linear properties similar to those of a given S-Box.

| Structure. | Algorithm | Reference | $\Delta_s$ | $\mathcal{L}_s/2$ | $\ell_{\text{dif}}$ | $\ell_{\text{lin}}$ | Involution | Deg. |
|---|---|---|---|---|---|---|---|---|
| Mathematical | Inverse | [Nyb94] | 4 | 16 | -7382.1 | -3329.4 | Yes | 7 |
| | BelT | [Bel11] | 8 | 26 | -74.8 | -123.0 | No | 7 |
| | E2 | [KMA+00] | 10 | 28 | -12.7 | -41.0 | No | 7 |
| | Magenta | [JH98] | 8 | 26 | -91.5 | -128.6 | No | 7 |
| | Picaro† | [PRC12] | 4 | 34 | -1359.5 | -1.40 | No | 4 |
| | Safer | [Mas94] | 128 | 46 | 0 | 0 | No | 7 |
| | Snow 3G | [ETS06a] | 8 | 32 | -36.8 | -3.16 | No | 5 |
| SPN | Clefia $S_0$ | [SSA+07] | 10 | 28 | -2.57 | -25.6 | No | 6 |
| | Crypton 0.5 | [LH98] | 16 | 40 | 0 | -0.02 | No | 6 |
| | Enocoro | [WIK+08] | 10 | 32 | -1.92 | -3.26 | No | 6 |
| | Iceberg | [SPR+04] | 8 | 32 | -18.4 | -3.59 | Yes | 7 |
| | Khazad | [BR00b] | 8 | 32 | -18.4 | -3.16 | Yes | 7 |
| | Midori-128 | [BBI+15] | 64 | 64 | 0 | 0 | Yes | 3 |
| | Qarma-($\sigma_{0/1}$)-128 | [Ava17] | 64 | 64 | 0 | 0 | No | 3 |
| | Twofish $p_0$ | [SKW+98] | 10 | 32 | -1.36 | -3.16 | No | 6 |
| | Twofish $p_1$ | [SKW+98] | 10 | 32 | -1.34 | -3.16 | No | 6 |
| Feistel | Crypton 1.0 | [DPV01] | 10 | 32 | -3.67 | -3.37 | No | 6 |
| | CS-cipher | [SV00] | 16 | 32 | 0 | -3.16 | Yes | 5 |
| | iScream | [GLS+14] | 16 | 32 | 0 | -3.16 | Yes | 6 |
| | Scream | [GLS+14, CDL16] | 8 | 32 | -16.1 | -3.16 | No | 6 |
| | Zorro | [GGNS13] | 10 | 32 | -2.19 | -3.37 | No | 7 |
| | Zuc $s_0$ | [ETS11] | 8 | 32 | -18.4 | -3.16 | No | 5 |
| GFN | Skinny-128 | [BJK+16] | 64 | 64 | 0 | 0 | No | 6 |
| Misty-like | Fantomas | [GLSV15] | 16 | 32 | 0 | -3.16 | No | 5 |
| Lai-Massey | Fox | [VJ04] | 16 | 32 | 0 | -3.16 | No | 6 |
| | Whirlpool | [BR00c] | 8 | 28 | -23.0 | -25.7 | No | 7 |
| | Fly | [KG16] | 16 | 32 | 0 | -3.16 | No | 5 |
| Hill-climbing | Anubis | [BR00a] | 8 | 34 | -28.7 | -1.04 | Yes | 7 |
| | Kalyna $\pi_0$ | [OGK+15b] | 8 | 24 | -104.2 | -235.8 | No | 7 |
| | Kalyna $\pi_1$ | [OGK+15b] | 8 | 24 | -122.6 | -268.1 | No | 7 |
| | Kalyna $\pi_2$ | [OGK+15b] | 8 | 24 | -129.9 | -239.3 | No | 7 |
| | Kalyna $\pi_3$ | [OGK+15b] | 8 | 24 | -122.6 | -242.9 | No | 7 |
| Pseudo-random | MD2 | [Kal92] | 10 | 38 | -1.355 | -0.10 | No | 7 |
| | newDES | [Sco85] | 12 | 36 | -0.44 | -0.32 | No | 7 |
| | Turing | [RH03] | 12 | 34 | -0.18 | -1.84 | No | 7 |
| Unknown | Cmea† | [WSK97] | 12 | 32 | -0.44 | -4.53 | No | 7 |
| | Kuznyechik | [Fed15] | 8 | 28 | -80.6 | -34.35 | No | 7 |
| | Skipjack | [U.S98] | 12 | 28 | -0.18 | -54.38 | No | 7 |

The dagger symbol "†" indicates that the function is not a permutation.

Table 9.7: An overview of different 8-bit S-Boxes from the literature.

Chapter 10

# Structural Attacks Against Feistel Networks

A possible structure for an S-Box is a Feistel network with few rounds. Given the look-up table of such a permutation, is it possible to recover the Feistel functions, even if they have been picked uniformly at random? In this chapter, we investigate algorithms allowing us to distinguish Feistel networks from random permutation and to actually recover their Feistel functions.

Our results are different depending on whether the Feistel network attacked uses an exclusive-or ($\oplus$) or a modular addition ($\boxplus$). Thus, we refer to a Feistel network using XOR as a $\oplus$-Feistel and to one based on modular addition as a $\boxplus$-Feistel. If we do not specify which operation is used, XOR is implied.

After clarifying the notation used throughout this chapter in Section 10.1 (p. 181), I describe attacks from the literature in Section 10.2 (p. 182).

Then, in Sections 10.3 (p. 186) and 10.4 (p. 190), I present new attacks against generic 5-round Feistel networks which recover all Feistel functions efficiently instead of only distinguishing them from random. Furthermore, unlike distinguishers from the literature, these attacks do not make any assumptions about whether the Feistel functions are bijective or not. The attack only works against $\oplus$-Feistel. It uses the *yoyo game*, a tool introduced in [BBD+99] which is improved by providing a more general theoretical framework for it and leveraging particular cycle structures to diminish its cost. The principle of the yoyo game is introduced in Section 10.3 (p. 186) and how to use cycles to improve it is described in Section 10.4 (p. 190).

In Section 10.5 (p. 197), the HDIM of Feistel networks is studied and, in particular, the presence of specific artifacts is proved. These can be used to distinguish Feistel networks efficiently from random permutation even after many rounds, provided that the degree of the Feistel functions is small enough.

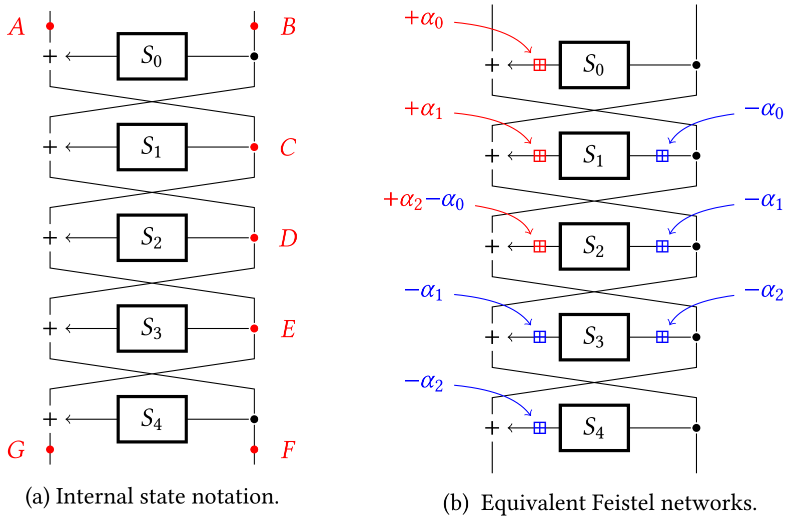
## 10.1 Notation

We introduce some notation for the different states during encryption which are summarized in Figure 10.1a. Each of the values is assigned a letter, e.g. the left side of the input is in position "$A$". When we look at 5-round Feistel networks, the input is fed into positions $A$ and $B$ and the output is read in $G, F$. For 6 rounds, the input is the same but the output is read in $H, G$ with $H = S_5(G) + F$. If we study a $\boxplus$-Feistel then "+" denotes modular addition ($\boxplus$); it denotes exclusive-or ($\oplus$) if we attack a $\oplus$-

Feistel. Concatenation is denoted "||" and encryption is denoted $\mathcal{E}$ (the number of rounds being clear from the context). For example, $\mathcal{E}(a||b) = g||f$ for a 5-round Feistel network. The bit-length of a branch of the Feistel network is equal to $n$.

In addition, we make the following observation.

**Observation 10.1.1.** *For an $R$-round Feistel, we can fix one entry of the last $R-2$ Feistel functions (or the first $R-2$ ones) arbitrarily. For example, the output of the 5-round Feistel network described in Figure 10.1b does not depend on $\alpha_0$, $\alpha_1$ or $\alpha_2$.*



(a) Internal state notation.

(b) Equivalent Feistel networks.

Figure 10.1: Notation and an observation about Feistel networks.

## 10.2 Overview of Structural Attacks Against Feistel Networks

After a quick summary of all the structural attacks and distinguishers I am aware of in Section 10.2.1 (p. 182) and in Table 10.1, I present in a bit more detail several of them. The differential distinguishers are in Section 10.2.2 (p. 184) and the impossible differential ones in Section 10.2.3 (p. 184). The recovery attacks based on SAT-solver usage and integral properties are described in Sections 10.2.4 (p. 185) and 10.2.5 (p. 185) respectively.

While our focus is on 2-branched balanced Feistel networks, we briefly mention attacks against variants of this structure in Section 10.2.6 (p. 186).

### 10.2.1 Summary

A first theoretical analysis of the Feistel structure and the first generic attacks were proposed in the seminal paper by Luby and Rackoff [LR88]. Since then, several cryptanalyses have been identified with the aim either to distinguish a Feistel network from a random permutation or to recover the Feistel functions.

Differential distinguishers against up to 5 rounds in the usual setting and 6 rounds in a multi-key setting are presented in [Pat08], although they assume that the Feistel functions are random functions and thus have inner-collisions. Conversely, an

| R | Type | Power | Restrictions | Time | Data | Ref. |
|---|---|---|---|---|---|---|
| 4 | Differential | Distinguisher | Non bij. round func. | $2^n$ | $2^n$ | [Pat08] |
| | 0-correlation | Distinguisher | Bij. round func. | $n2^{2n}$ | $2^{2n}$ | Sec 12.2.3.3 |
| | Guess & Det. | Recovery | – | $2^{3n}$ | $2^{3n}$ | [BLP16] |
| 5 | Differential | Distinguisher | Non bij. round func. | $2^{2n}$ | $2^{2n}$ | [Pat08] |
| | Imp. diff. | Distinguisher | Bij. round func. | $2^{2n}$ | $2^n$ | [Knu98] |
| | HDIM-based | Distinguisher | Bij. round func. | $2^{2n-1}$ | $2^{2n-1}$ | Sec. 10.5.3 |
| | SAT-based | Recovery | $n \leq 7$ | Practical | $2^{2n}$ | [BP15] |
| | Yoyo | Recovery | Only for $\oplus$-Feistel | $2^{2n}$ | $2^{2n}$ | Sec. 10.4.4 |
| | Integral | Recovery | $S_1$ or $S_3$ bij. | $2^{2.81n}$ | $2^{2n}$ | [BLP16] |
| | Guess & Det. | Recovery | – | $2^{n2^{3n/4}}$ | $2^{2n}$ | [BLP16] |
| | Imp. monom. | Recovery | Bij. round func. | $2^{3n}$ | $2^{2n}$ | [PU16] |
| 6 | Differential | Distinguisher | Multi-key setting | $2^{4n}$ | $2^{4n}$ | [Pat08] |
| | Yoyo | Recovery | Only for $\oplus$-Feistel | $2^{n2^{2n}+2n}$ | $2^{4n}$ | Sec. 10.4.5 |
| 7 | Yoyo | Recovery | Only for $\oplus$-Feistel | $2^{n2^{2n+1}+2n}$ | $2^{4n}$ | Sec. 10.4.5 |
| r | HDIM-based | Distinguisher | Bij. $F_i$, $\theta(d, r-1) < 2n$ | $2^{2n-1}$ | $2^{2n-1}$ | Sec. 10.5.3 |
| | HDIM-based | Distinguisher | Non bij. $F_i$, $\theta(d, r) < 2n$ | $2^{2n-1}$ | $2^{2n-1}$ | Sec. 10.5.3 |
| | Imp. monom. | Recovery | $d^{r-3} < n$ | $2^{3n}$ | $2^{2n}$ | [PU16] |

Table 10.1: Structural attacks against Feistel networks. $n$ is the branch size, $d$ is the degree of the Feistel functions, $F_i$ is the Feistel function at round $i$.

impossible differential covering 5 rounds in the case where the Feistel functions are permutations is described in [Knu98] and used to attack DEAL, a block cipher based on a 6-round Feistel network.

More recently, Aleksei Udovenko and I identified some patterns in the HDIM of Feistel networks. These can cover much more rounds than the differential ones provided that the algebraic degree of the Feistel function is not too high. These distinguishers are described in Section 10.5. There are also visible patterns in the LAT of 4-round Feistel networks which allow an easy identification of those, as explained in Section 12.2.3.3 (p. 232).

Cryptographers have also investigated recovery attacks aiming at recovering the secret components of a Feistel network instead of merely identifying its structure. Lampe *et al.* [LS15], followed by Dinur *et al.* [DDKS15], studied Feistel networks where the Feistel function at round $i$ consists of $x \mapsto F_i(x \oplus k_i)$, with $F_i$ being public but $k_i$ being kept secret. If the subkeys are independent then it is possible to recover all of them for a 5-round (respectively 7-round) Feistel network in time $O(2^{2n})$ (resp. $O(2^{3n})$) using only 4 known plaintexts with the optimized Meet-in-the-Middle attack described in [DDKS15].

However, we consider the much more complex case where the Feistel functions are completely unknown. First, I designed a method relying on a SAT-solver which was published in [BP15] and is summarized in Section 10.2.4 (p. 185). It is capable of decomposing Feistel networks with up to $n = 7$ in at most a couple of hours. How this time scales for larger $n$ is unclear but will anyway remain slower than the attacks presented in this chapter.

The yoyo game described in Section 10.3 (p. 186) and its cycle-based improvement described in Section 10.4 (p. 190) are the most efficient recovery attacks. However, they only work against ⊕-Feistel. In a joint work, Leurent found guess and determine attacks which, while less efficient, do work against ⊞-Feistel [BLP16]. If some Feistel functions are bijective, then some integral attacks become possible. These are summarized in Section 10.2.5 (p. 185).

All these attacks, their limitations and their efficiency are summarized in Table 10.1, where "Recovery" means that the attack recovers the LUT of all Feistel functions while "Distinguisher" means the attack merely identifies the Feistel network as such. The function $\theta : \mathbb{Z}^2 \rightarrow \mathbb{Z}$ is defined by

$$\theta(d, r) = d^{\lfloor r/2 \rfloor - 1} + d^{\lceil r/2 \rceil - 1},$$

where $\lfloor 2k \rfloor = \lfloor 2k + 1 \rfloor = 2k$ and $\lceil 2k \rceil = \lceil 2k - 1 \rceil = 2k$.

A short description of some these attacks is given in the remainder of this section for the sake of completeness.

### 10.2.2 Differential Distinguishers

In [Pat08], Patarin shows a differential distinguisher against 5-round Feistel networks. However, it only works if the Feistel functions have inner-collisions. It is based on the following observation. Let $(g_i || f_i)$ be the image of $(a_i || b_i)$ by a permutation and let $b_i$ be constant. Then for $i \neq j$, such that $f_i = f_j$, count how many times $a_i \oplus a_j = g_i \oplus g_j$. This number is roughly twice as high for a 5-round Feistel network than for a random permutation.

In the same paper, Patarin suggests two distinguishers against 6-round ⊕-Feistel networks. However, these do not target a permutation but a generator of permutations. This can be interpreted as a multi-key attack: the attacker has a black-box access to several permutations and either none or all are 6-round ⊕-Feistel networks. The first attack uses the fact that the signature of a ⊕-Feistel network is always even. The second attack exploits a statistical bias too weak to be reliably observable using one codebook but usable when several permutations are available. It works by counting all quadruples of encryptions $(a_i || b_i) \rightarrow (g_i || h_i)$, $i = 1..4$ satisfying this system:

$$\begin{cases} b_1 = b_3,\ b_2 = b_4 \\ g_1 = g_2,\ g_3 = g_4 \\ a_1 \oplus a_3 = a_2 \oplus a_4 = g_1 \oplus g_3 \\ h_1 \oplus h_2 = h_3 \oplus h_4 = b_1 \oplus b_2. \end{cases}$$

If there are $\lambda$ black-boxes to distinguish and if $m$ queries are performed for each then we expect to find about $\lambda m^4 2^{-8n}$ solutions for a random permutation and $2\lambda m^4 2^{-8n}$ for 6-round Feistel networks, i.e. twice as much.

### 10.2.3 Impossible Differential

Knudsen described in [Knu98] an impossible differential attack against his AES proposal, DEAL, a 6-round Feistel network using the DES [U.S99] as a round function. This attack is made possible by the existence of a 5-round impossible differential caused by the Feistel functions being permutations. In this case, an input difference $(\alpha || 0)$ cannot be mapped to a difference of $(\alpha || 0)$ after 5 rounds. This would imply that the non-zero difference which has to appear in $D$ as the image of $\alpha$ by $S_2$ is mapped to 0, which is impossible.

To distinguish such a 5-round Feistel network from a random permutation we need to generate $\lambda \cdot 2^{2n}$ pairs with input difference $(\Delta||0)$. Among those, about $\lambda$ should have an output difference equal to $(\Delta||0)$ if the permutation is a random permutation while it is impossible to observe if for a 5-round Feistel network with bijective Feistel functions. Note that while the time complexity is $O(2^{2n})$, the data complexity can be brought down to $O(2^n)$ using structures.

An attack on 6 rounds uses this property by identifying pairs of encryptions with difference $(\alpha||0)$ in the input and $(\alpha||\Delta)$ for the output for any $\Delta \neq 0$. A pair has a correct output difference with probability $2^{-n}(1-2^{-n})$ since $\alpha$ is fixed and $\Delta$ can take any value except 0. We repeat this process for the whole codebook and all $\alpha \neq 0$ to obtain $2^{n+(2n-1)} \cdot 2^{-n}(1-2^{-n}) = 2^{2n-1} - 2^{n-1}$ pairs. Each of them gives an impossible equation for $S_5$: if $\{(a||b) \rightarrow (g||h), (a \oplus \alpha||b) \rightarrow (g \oplus \alpha||h \oplus \Delta)\}$ is a pair of encryptions then it is impossible that $S_5(g) \oplus S_5(g \oplus \alpha) = \Delta$ as it would imply the impossible differential. In the end, we have a system of about $2^{2n-1} - 2^{n-1}$ impossible equations, a random Feistel function satisfying an impossible equation with probability $(1-2^{-n})$. Thus, this attack filters out all but the following fraction of candidates for $S_5$:

$$\text{Impossible differential filter} = \left(1-2^{-n}\right)^{2^{2n-1}-2^{n-1}} \approx 2^{0.72-1.443 \cdot 2^{n-1}}.$$

### 10.2.4 Sat-based Recovery

In [BP15], I proposed an algorithm based on a sat-solver for attacking both ⊕-Feistel and ⊞-Feistel networks. The idea is the same in both cases and consists of building a cnf formula which uses the entries of the lut of the $r$ Feistel functions as unknowns. Using these unknowns, we build for each plaintext/ciphertext a cnf encoding that a block cipher using the unknown Feistel functions maps said plaintext to said ciphertext. An off-the-shelf sat-solver such as minisat [ES04] is then used to solve the cnf formula. If it has no solution, it means that the function considered is not a Feistel network with $r$ rounds. If a solution is found, we deduce the Feistel functions from the variable assignment returned.

### 10.2.5 Integral Attack

This attack was first presented in [BLP16]. In fact, its principle had been suggested by an anonymous reviewer of the sac'15 conference.

Let us use the notation described in Section 10.1 (p. 181). Suppose that $S_1$ is a bijection. If $B$ is fixed and if $A$ takes all $2^n$ possible values, then $C$ and, in turn, $D$ also take all possible values. It means that the sum of the values $E(A,B)$ taken over all values of $A$ is fixed. Indeed, this sum is equal to

$$\sum_{A \in \mathbb{F}_2^n} E(A,B) = \sum_{A \in \mathbb{F}_2^n} S_2\left((D(A,B)\right) + \sum_{A \in \mathbb{F}_2^n} C(A,B)\,,$$

where $\sum_{A \in \mathbb{F}_2^n} C(A,B) = 0$ because[1] $C$ takes all possible values and the other half of this sum does not depend on $B$: $\sum_{A \in \mathbb{F}_2^n} S_2\left((D(A,B)\right) = \sum_{D \in \mathbb{F}_2^n} S_2(D)$.

Using this property, we can attack both 5-round ⊕-Feistel and 5-round ⊞-Feistel. The recovery is performed by writing a system of linear equations encoding that $\sum_{A \in \mathbb{F}_2^n} E(A,B) = \sum_{A \in \mathbb{F}_2^n} E(A,B')$ for all pairs $(B,B')$ using the outputs of $S_4(x)$ as

[1] If + denotes xor then the equality holds. If it denotes addition modulo $2^n$, it holds modulo $2^n$.

variables. Solving this system yields the look-up table of $S_4$. After that, only 4 rounds remain to attack.

A similar attack was described by Aleksei Udovenko in a joint work with me [PU16]. Instead of recovering the LUT of the last Feistel function using a constant sum, it works by identifying some monomials that cannot appear in the ANF of the Feistel network at round 4. This information is then used to recover the ANF of the last Feistel function.

### 10.2.6 Attacks Targeting Variants of Feistel Networks

Variants of the Feistel structure have received less attention in terms of structural attacks. Unbalanced Feistel networks with public Feistel functions but XORed with secret round keys are vulnerable to the Meet-in-the-Middle attacks described in [GJNS17].

Similarly, generalized Feistel networks based on public Feistel functions preceded by secret key XOR have been studied by Blondeau and Minier in [BM15]. They found some structural integral, impossible differential and zero-correlation distinguishers against such structures. When the Feistel functions used in a generalized Feistel network are secret and independent, some differential distinguishers exist. They are described in [NVP13].

## 10.3 Yoyo Game and Cryptanalysis

Several cryptanalyses have been proposed in the literature that rely on encrypting a plaintext, performing an operation on the ciphertext and then decrypting the result. For example, the "double-swiping" used against newDES [KSW97] in the related-key setting relies on encrypting a pair of plaintexts using two related-keys and decrypting the result using two different related-keys. Another example is the boomerang attack introduced by Wagner [Wag99] in the single-key setting. A pair with input difference $\delta$ is encrypted. Then, a difference $\Delta$ is added to the ciphertexts and the results are decrypted, hopefully yielding two plaintexts with a difference of $\delta$.

In this section, we discuss how the yoyo game introduced in [BBD+99] and described in Section 10.3.1 (p. 186) can be used against Feistel networks. A theoretical framework is provided in Section 10.3.2 (p. 187) and it is applied to 5-round $\oplus$-Feistel in Section 10.3.3 (p. 188).

### 10.3.1 The Original Yoyo Game

The yoyo game was introduced by Biham *et al.* in [BBD+99] where it was used to attack the 16 center rounds of Skipjack [U.S98], a block cipher described in Section 9.2.1 (p. 168). We describe this attack using slightly different notation and terminology to be coherent with the rest of our paper. In this paragraph, $\mathcal{E}_k$ denotes an encryption using round-reduced Skipjack under key $k$.

If the difference between two encryptions at round 5 is $(0, \Delta, 0, 0)$ with $\Delta \neq 0$ then the other three words have difference 0 between rounds 5 and 12. Two encryptions satisfying this truncated differential are said to be *connected*.

The key observation is the following. Let $x, x'$ defined as $x = (x_0, x_1, x_2, x_3)$ and $x' = (x'_0, x'_1, x_2, x'_3)$ be two plaintexts with the same value $x_2$. If they are connected, then the pair $\phi(x, x') = \big((x_0, x'_1, x_2, x_3), (x'_0, x_1, x_2, x'_3)\big)$ is connected as well. A detailed explanation on why it is the case is given in [BBD+99]. Furthermore, let

$y = (y_0, y_1, y_2, y_3)$ and $y' = (y'_0, y'_1, y'_2, y'_3)$ be the encryption of $x$ and $x'$ respectively, i.e. $y = \mathcal{E}_k(x)$ and $y' = \mathcal{E}_k(x')$. We can form two new ciphertexts by swapping their first words to obtain $z = (y'_0, y_1, y_2, y_3)$ and $z' = (y_0, y'_1, y'_2, y'_3)$. If we decrypt them to obtain $(u, u') = (\mathcal{E}_k^{-1}(z), \mathcal{E}_k^{-1}(z'))$, then $u$ and $u'$ are connected. If we denote $\psi(x, x')$ the function which encrypts $x$ and $x'$, swaps the first words of the ciphertexts obtained and decrypts the result then $\psi$ preserves connection, just like $\phi$. It is thus possible to iterate $\phi$ and $\psi$ to obtain many connected pairs, this process being called the *yoyo game*.

In this section, we present other definitions of the connection and of the functions $\phi$ and $\psi$ which allow us to play a similar yoyo game on 5-round Feistel networks.

### 10.3.2 Theoretical Framework for the Yoyo Game

Consider two plaintexts $a||b$ and $a'||b'$ such that the difference between their encryptions in positions $(C, D)$ is equal to $(\gamma, 0)$ with $\gamma \neq 0$. Then the difference in position $E$ is equal to $\gamma$. Conversely, the difference in $(E, D)$ being $(\gamma, 0)$ implies that the difference in $C$ is $\gamma$. When this is the case, the two encryptions satisfy the systems of equations and the trail described in Figure 10.2.

**Top equations**

$$\begin{cases} S_0(b) \oplus S_0(b') = a \oplus a' \oplus \gamma \\ S_1(a \oplus S_0(b)) \oplus S_1(a' \oplus S_0(b')) = b \oplus b' \end{cases}$$

**Bottom equations**

$$\begin{cases} S_4(f) \oplus S_4(f') = g \oplus g' \oplus \gamma \\ S_3(g \oplus S_4(f)) \oplus S_3(g' \oplus S_4(f')) = g \oplus g' \end{cases}$$

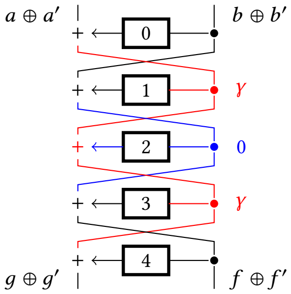


Figure 10.2: The equations defining connection in $\gamma$ and the corresponding differential trail.

**Definition 10.3.1.** *If the encryptions of $a||b$ and $a'||b'$ follow the trail in Figure 10.2 then they are said to be* connected in $\gamma$.

This *connection* is an “exclusive” relation: if $(a||b)$ and $(a'||b')$ are connected, then neither $(a||b)$ nor $(a'||b')$ can be connected to anything else. Furthermore, we can replace $(a, a')$ by $(a \oplus \gamma, a' \oplus \gamma)$ in the top equations and still have them be true. Indeed, the two $\gamma$ cancel each other in the first one. In the second, the values input to each call to $S_1$ are simply swapped as a consequence of the first equation. Similarly, we can replace $(g, g')$ by $(g \oplus \gamma, g' \oplus \gamma)$ in the bottom equations.[2] As a consequence of these observations, we state the following lemma.

**Lemma 10.3.1.** *We define the following two involutions*

$$\phi_\gamma(a||b) = (a \oplus \gamma)||b, \ \psi_\gamma = \mathcal{E}^{-1} \circ \phi_\gamma \circ \mathcal{E}.$$

*If $a||b$ and $a'||b'$ are connected then, with probability 1:*

[2] However, such a yoyo game cannot be played against a ⊞-Feistel, as explained in Section 10.3.4. It only works in characteristic 2.

- *$\phi_\gamma(a||b)$ and $\phi_\gamma(a'||b')$ are connected,*
- *$\psi_\gamma(a||b)$ and $\psi_\gamma(a'||b')$ are connected.*

By repeatedly applying $\phi_\gamma$ and $\psi_\gamma$ component-wise on a pair of plaintexts $(x, x')$, we can play a yoyo game which preserves connection in $\gamma$. This process is defined formally below.

**Definition 10.3.2.** *Let $\left(x_0 = (a_0||b_0), x'_0 = (a'_0||b'_0)\right)$ be a pair of inputs. The* yoyo game *in $\gamma$ starting in $(x_0, x'_0)$ is defined recursively as follows:*

$$(x_{i+1}, x'_{i+1}) = \begin{cases} \left(\phi_\gamma(x_i), \phi_\gamma(x'_i)\right) & \text{if } i \text{ is even,} \\ \left(\psi_\gamma(x_i), \psi_\gamma(x'_i)\right) & \text{if } i \text{ is odd} \end{cases}$$

**Lemma 10.3.2.** *If $(x_0, x'_0)$ is connected in $\gamma$ then all pairs in the game starting in $(x_0, x'_0)$ are connected in $\gamma$. In other words, either all pairs within the game played using $\phi_\gamma$ and $\psi_\gamma$ are connected in $\gamma$ or none of them are.*

### 10.3.3 The Yoyo Cryptanalysis Against 5-Round ⊕-Feistel Networks

Given a yoyo game connected in $\gamma$, it is easy to recover Feistel functions $S_0$ and $S_4$ provided that the yoyo game is long enough, i.e. that it contains enough connected pairs to be able to recover all $2^n$ entries of both S-Boxes. If the yoyo game is not connected in $\gamma$ then *yoyo cryptanalysis* (Algorithm 10.1) identifies it as such very efficiently.

It is a differential cryptanalysis which uses the fact that all pairs in the game are (supposed to be) right pairs for the differential trail defining connection in $\gamma$. If it is not the case, $S_0$ or $S_4$ will end up requiring contradictory entries, e.g. $S_0(0) = 0$ and $S_0(0) = 1$. In this case, the game is not connected in $\gamma$ and must be discarded. Yoyo cryptanalysis[3] is described in Algorithm 10.1. It only takes as inputs a (possible) yoyo game and the value of $\gamma$. Algorithm 10.2 describes `AddEntry`, a subroutine handling some linear equations. Because of Observation 10.1.1 (p. 182), one entry can be set arbitrarily. In this algorithm, we chose $S_0(0) = 0$.

Let $\mathcal{Y}$ be a (supposed) yoyo game containing $|\mathcal{Y}|$ pairs of plaintexts. For each pair in it, one of three operations is undertaken: adding an equation to the list, returning `FAIL` or calling `AddEntry`. While the recursive calls to `AddEntry` may lead to a worse time complexity quadratic in $|\mathcal{Y}|$ if naïvely implemented, this problem can be mitigated by using a hashtable indexed by the Feistel functions' inputs instead of a list. Furthermore, since already solved equations are removed, the total time complexity is $O(|\mathcal{Y}|)$.

### 10.3.4 On the Infeaseability of Our Yoyo Game Against an ⊞-Feistel

Assume that the following equations hold:

$$\begin{cases} (S_0(b) + a) - (S_0(b') + a') = \gamma \\ (S_1(S_0(b) + a) + b) - (S_1(S_0(b') + a') + b') = 0. \end{cases} \tag{10.1}$$

In order to be able to play a yoyo game against the corresponding ⊞-Feistel, we need to be able to replace $a$ by $a + \gamma$ and $a'$ by $a' + \gamma$ in System (10.1) and still have it hold.

[3] It can also recover $S_4$ in an identical fashion but this part is omitted for the sake of clarity

**Algorithm 10.1** Yoyo cryptanalysis against a 5-round $\oplus$-Feistel network
**Inputs:** supposed yoyo game $\left(a_i || b_i, a'_i || b'_i\right)$ ; difference $\gamma$
**Output:** $S_0$ or `FAIL`.

```
$L_e \leftarrow []$                                              ▷ List of equations
$S_0 \leftarrow$ empty S-Box
$\delta_0 \leftarrow a_0 \oplus a'_0 \oplus \gamma$
$S_0(b_0) \leftarrow 0, S_0(b'_0) \leftarrow \delta_0$
for all $i \geq 1$ do
    $\delta_i \leftarrow a_i \oplus a'_i \oplus \gamma$
    if $S_0(b_i)$ and $S_0(b'_i)$ are already known and $S_0(b_i) \oplus S_0(b'_i) \neq \delta_i$ then
        return FAIL
    else if $S_0(b_i)$ is known but not $S_0(b'_i)$ then
        AddEntry $\left(S_0, b'_i, S_0(b_i) \oplus \delta_i, L_e\right)$ ; if it fails then return FAIL
    else if $S_0(b'_i)$ is known but not $S_0(b_i)$ then
        AddEntry $\left(S_0, b_i, S_0(b'_i) \oplus \delta_i, L_e\right)$ ; if it fails then return FAIL
    else
        add "$S_0(b'_i) \oplus S_0(b_i) = \delta_i$" to $L_e$.
    end if
end for
return $S_0$
```

**Algorithm 10.2** `AddEntry`: adding a new entry to $S_0$
**Inputs:** S-Box $S_0$ ; input $x$ ; output $y$ ; List of equations $L_e$
**Output** `SUCCESS` or `FAIL`.

```
if $S_0(x)$ already set and $S_0(x) = y$ then
    return SUCCESS                                         ▷ No new information
else if $S_0(x)$ already set and $S_0(x) \neq y$ then
    return FAIL                                            ▷ Contradiction identified
else
    $S_0(x) \leftarrow y$
    for all Equation $S_0(x_i) \oplus S_0(x'_i) = \Delta_i$ in $L_e$ do
        if $S_0(x_i)$ and $S_0(x'_i)$ are set then
            if $S_0(x_i) \oplus S_0(x'_i) \neq \Delta_i$ then return FAIL ; else Remove eq. from $L_e$
                                                                      ▷ Eq. satisfied
        else if $S_0(x_i)$ is set but not $S_0(x'_i)$ then
            AddEntry $\left(S_0, x'_i, S_0(x_i) \oplus \Delta_i, L_e\right)$ ; if it fails then return FAIL
                                                                 ▷ Eq. gives new entry
        else if $S_0(x'_i)$ is set but not $S_0(x_i)$ then
            AddEntry $\left(S_0, x_i, S_0(x'_i) \oplus \Delta_i, L_e\right)$ ; if it fails then return FAIL
                                                                 ▷ Eq. gives new entry
        end if
    end for
end if
return SUCCESS
```

In other words, we need that if Equations (10.1) hold then the following equations hold as well:

$$\begin{cases} (S_0(b) + a + \gamma) - (S_0(b') + a' + \gamma) = \gamma \\ (S_1(S_0(b) + a + \gamma) + b) - (S_1(S_0(b') + a' + \gamma) + b') = 0. \end{cases} \tag{10.2}$$

The first one trivially does. Using it, we note that $S_0(b) + a + \gamma = S_0(b') + a' + 2\gamma$. Let $X = S_0(b') + a'$. Then the left-hand side of the second equation in System (10.2) can be re-written as $S_1(X + 2\gamma) - S_1(X + \gamma) + b - b'$. Furthermore, the second equation in System (10.1), which is assumed to hold, implies that $S_1(X + \gamma) - S_1(X) = b' - b$. Thus, the left-hand side of the second equation in System (10.2) is equal to

$$S_1(X + 2\gamma) - (b' - b + S_1(X)) + b - b' \;=\; S_1(X + 2\gamma) - S_1(X) - 2(b' - b).$$

The term $S_1(X+2\gamma)-S_1(X)$ has an unknown value unless $\gamma = 2^{n-1}$. Nevertheless, in this case, we would need $2(b' - b) = 0$ which does not have a probability equal to 1. However both $S_1(X+2\gamma)-S_1(X)$ and $2(b'-b)$ are always equal to 0 in characteristic 2 which is why our yoyo game can always be played against a $\oplus$-Feistel.

## 10.4 An Improvement Using Cycles

In order to perform a complete recover attack, it is necessary to choose a value $\gamma$ and pair of random plaintexts $(x, x')$, then iterate $\phi_\gamma$ and $\psi_\gamma$ to generate the corresponding yoyo game and then run Algorithm 10.1 in time at least $O(2^n)$ on it. If it failed, another pair of plaintexts is picked. The overall time complexity of this method is at least equal to $O\left(2^{3n}\right)$.

It is possible to significantly improve it into $O\left(2^{2n}\right)$ by essentially getting a yoyo game and exploiting it with Algorithm 10.1 in one go using a trick involving the cycle structure of the composition of $\phi_\gamma$ and $\psi_\gamma$.

The general method used is described in Section 10.4.1 (p. 190). It involves cycles with different structures described in Section 10.4.2 (p. 191) and for which some experimental results are given in Section 10.4.3 (p. 192). The attack leveraging these cycles targets 5-round $\oplus$-Feistel and is described in Section 10.4.4 (p. 194). It can be used as a subroutine to attack 6- and even 7-round $\oplus$-Feistel, as shown in Section 10.4.5 (p. 195).

### 10.4.1 Cycles and Yoyo Cryptanalysis

A yoyo game is a cycle of $\psi_\gamma$ and $\phi_\gamma$ applied iteratively component-wise on a pair of elements. Thus, it can be decomposed into two cycles, one for each "side" of the game: $(x_0, x_1, x_2, ...)$ and $(x'_0, x'_1, x'_2, ...)$. This means that both cycles must have the same length, otherwise the game would imply that $x_0$ is connected to $x'_j$ for $j \neq 0$, which is impossible. Since both $\phi_\gamma$ and $\psi_\gamma$ are involutions, the cycle can be iterated through in both directions. Therefore, finding one cycle gives us two directed cycles.

In order to exploit yoyo games, we could generate pairs $(x_0, x'_0)$ at random, generate the yoyo game starting at this pair and then try and recover $S_0$ and $S_4$ but this endeavor would only work with probability $2^{-2n}$ (the probability for two random points to be connected). Instead, we can use the link between cycles, yoyo games and connection in $\gamma$ as is described in this section. Note that the use of cycles in cryptography is not new; in fact it was used in the first cryptanalyses against

Enigma. More recently, particular distributions of cycle sizes were used to attack involutional ciphers [Bir03] and, in Section 4.3 (p. 83), to distinguish round-reduced Prince-core [BCG+12] from random.

### 10.4.2 Different Types of Cycles

Let $C = (x_i)_{i=0}^{\ell-1}$ be a cycle of length $\ell$ of $\psi_\gamma$ and $\phi_\gamma$, with $x_{2i} = \psi_\gamma(x_{2i-1})$ and $x_{2i+1} = \phi_\gamma(x_{2i})$. We denote the point connected to $x_i$ as $y_i$, where all indices are taken modulo $\ell$. Since $x_i$ and $y_i$ are connected, and the connection relation is one-to-one, we also have $y_{2i} = \psi_\gamma(y_{2i-1})$ and $y_{2i+1} = \phi_\gamma(y_{2i})$. Therefore, $C' = (y_i)_{i=0}^{\ell-1}$ is also a cycle of length $\ell$.

We now classify the cycles according to the relationship between $C$ and $C'$.

- If $C$ and $C'$ are **D**istincts, $C$ is a **Type-D** cycle. A representation is given in Figure 10.3a. Otherwise, there exists $k$ such that $y_0 = x_k$.

- If $k$ is even, we have $x_{k+1} = \phi_\gamma(x_k)$. Since $x_k = y_0$ is connected to $x_0$, $x_{k+1} = \phi_\gamma(x_k)$ is connected to $\phi_\gamma(x_0) = x_1$, i.e. $y_1 = x_{k+1}$. Further, $x_{k+2} = \psi_\gamma(x_{k+1})$ is connected to $\psi_\gamma(x_1) = x_2$, i.e. $y_2 = x_{k+2}$. By induction, we have $y_i = x_{k+i}$. Therefore $x_0$ is connected to $x_k$ and $x_k$ is connected to $x_{2k}$. Since the connection relation is one-to-one, this implies that $2k = \ell$.

  We denote this setting as a **Type-S** cycle. Each element $x_i$ is connected to $x_{i+\ell/2}$. Thus, if we represent the cycle as a circle, the connections between the elements would all cross in its center, just like **S**pokes, as can be seen in Figure 10.3b.

- If $k$ is odd, we have $x_{k-1} = \phi_\gamma(x_k)$. Since $x_k = y_0$ is connected to $x_0$, $x_{k-1} = \phi_\gamma(x_k)$ is connected to $\phi_\gamma(x_0) = x_1$, i.e. $y_1 = x_{k-1}$. Further, $x_{k-2} = \psi_\gamma(x_{k-1})$ is connected to $\psi_\gamma(x_1) = x_2$, i.e. $y_2 = x_{k-2}$. By induction, we have $y_i = x_{k-i}$.

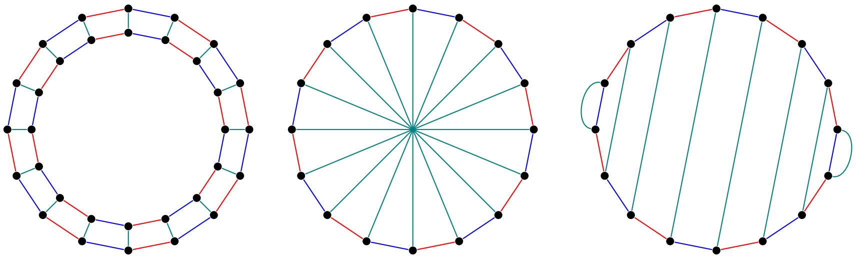
  We denote this setting as a **Type-P** cycle. If we represent the cycle as a circle, the connections between the elements would all be **P**arallel to each other as can be seen in Figure 10.3c.

  In particular, there at exactly two pairs $(x_i, x_{i+1})$ such that $x_i$ and $x_{i+1}$ are connected. Indeed, we have $x_{i+1} = y_i$ if and only if $i + 1 \equiv k - i \bmod \ell$ i.e. $i \equiv (k-1)/2 \bmod \ell/2$. As a consequence, the existence of $w$ connected pairs $(x, x')$ with $x' = \phi_\gamma(x)$ or $x' = \psi_\gamma(x)$ implies the existence of $w/2$ Type-P cycles.

  In addition, Type-P cycles can only exist if either $S_1$ or $S_3$ are not bijections. Indeed, if $(a||b)$ and $(a \oplus \gamma||b)$ are connected then the difference in position $D$ cannot be zero unless $S_1$ can map a difference of $\gamma$ to zero. If it is a permutation, this is impossible. The situation is identical for $S_3$.

  Furthermore, each value $c$ such that $S_1(c) = S_1(c \oplus \gamma)$ implies the existence of $2^n$ values $(a||b)$ connected to $\phi_\gamma(a||b)$ as $b$ can be chosen arbitrarily and $a$ computed from $b$ and $c$. Again, the situation is identical for $S_3$. Thus, if $S_1(x) = S_1(x \oplus \gamma)$ has $w_1$ solutions and if $S_3(x) = S_3(x \oplus \gamma)$ has $w_3$ solutions then there are $(w_1 + w_3) \cdot 2^{n-2}$ Type-P cycles.

In Section 10.4.3.1 (p. 192), experimental examples of the structures of the functional graphs of $\phi_\gamma$ and $\psi_\gamma$ are provided for small $n$.



(a) Type-D cycles. (b) A Type-S cycle. (c) A Type-P cycle.

Figure 10.3: All the types of cycles that can be encountered. $\phi_\gamma$ is a blue line, $\psi_\gamma$ is a red one and connection is a green one (remember that $\phi_\gamma$ and $\psi_\gamma$ are involutions).

We denote by $\Pr[\mathrm{S}]_n$ the probability that a Type-S cycle exists for a specific $\gamma$ for a 5-round Feistel network built out of bijective Feistel functions. When averaged over all such Feistel networks, this probability does not depend on $\gamma$. A discussion about its value is given in Section 10.4.3.2.

### 10.4.3 Experimental Results About Cycles

To better understand and illustrate the behavior of these different types of cycles, we made some experiments. In Section 10.4.3.1 (p. 192), we investigate their distributions.

#### 10.4.3.1 Examples of Cycle Distribution

To illustrate the different types of cycles and their behavior, we plotted the functional graphs of $\phi_\gamma$ and $\psi_\gamma$ along with connection in $\gamma$ for three different Feistel networks with $n = 3$. The functional graphs of $\phi_\gamma$ and $\psi_\gamma$ for different 5-round Feistel networks are represented in Figures 10.4, 10.5 and 10.6. In those, $\phi_\gamma$ is in blue, $\psi_\gamma$ is red and connection in $\gamma$ is green.

First, we considered the case of a 5-round $\oplus$-Feistel built using only permutations. We denote it $\mathcal{E}_0$ and its Feistel functions are:

- $S_0$ = `[2, 0, 1, 3, 4, 7, 6, 5]`,
- $S_1$ = `[5, 6, 2, 1, 7, 0, 3, 4]`,
- $S_2$ = `[7, 2, 1, 3, 5, 6, 0, 4]`,
- $S_3$ = `[4, 1, 6, 2, 3, 7, 0, 5]`, and
- $S_4$ = `[6, 3, 0, 5, 2, 1, 4, 7]`.

The functional graph of $\phi_\gamma$ composed with $\psi_\gamma$ yields two Type-S cycles in this case, as shown in Figure 10.4.

Then, we looked at the case of a 5-round $\oplus$-Feistel $\mathcal{E}_1$ which is built out of functions, namely:

- $S_0$ = `[2, 6, 1, 0, 6, 3, 4, 6]`,
- $S_1$ = `[6, 6, 0, 6, 2, 5, 1, 2]`,

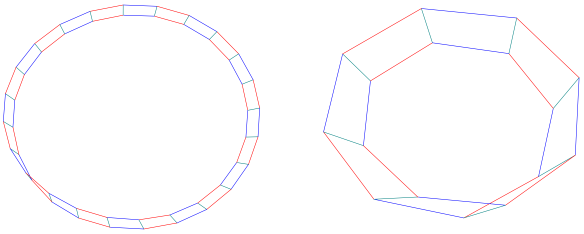


Figure 10.4: Case of $\mathcal{E}_0$ (permutations).

- $S_2$ = `[7, 5, 1, 5, 1, 2, 5, 4]`,
- $S_3$ = `[7, 1, 6, 2, 4, 3, 0, 1]`, and
- $S_4$ = `[4, 5, 2, 7, 6, 7, 7, 3]`.

Its second Feistel function ($S_1$) has two collisions for an input difference of $\gamma = 1$ while $S_3$ has none. Thus, it yields $2 \cdot 2^{n-2} = 4$ Type-P cycles. It also yields two Type-D ones, as can be seen in Figure 10.5).

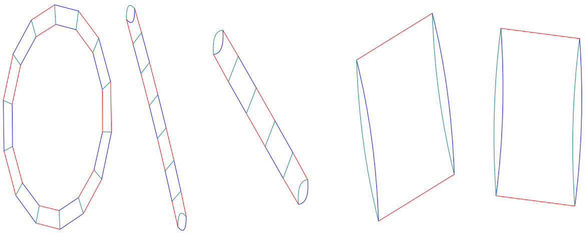


Figure 10.5: Case of $\mathcal{E}_1$ (2 collisions in $S_1$).

Finally, we considered another 5-round ⊕-Feistel, $\mathcal{E}_2$, which also uses functions. Those are

- $S_0$ = `[3, 7, 1, 1, 1, 5, 6, 2]`,
- $S_1$ = `[2, 2, 1, 5, 3, 2, 6, 0]`,
- $S_2$ = `[1, 0, 5, 4, 4, 4, 6, 4]`,
- $S_3$ = `[7, 3, 1, 1, 3, 4, 1, 7]`, and
- $S_4$ = `[0, 2, 1, 0, 5, 1, 3, 4]`,

and both $S_1$ and $S_3$ have two collisions for an input difference of $\gamma = 1$. Thus, it yields $(2 + 2) \cdot 2^{n-2} = 8$ Type-P cycles which are shown in Figure 10.6.

#### 10.4.3.2 Experimental Estimation of $\Pr[\mathrm{S}]_n$

We call $\gamma_{\mathrm{win}}$ the first value of $\gamma$ such that a large Type-S cycle is found. Those will be used in our attack against 5-round ⊕-Feistel in Section 10.3.3 (p. 188). Since $\Pr[\mathrm{S}]_n$ does not depend on $\gamma$, the probability distribution of $\gamma_{\mathrm{win}}$ is:

$$P[\gamma_{\mathrm{win}} = x] = (1 - \Pr[\mathrm{S}]_n)^{x-1} \cdot \Pr[\mathrm{S}]_n \,.$$

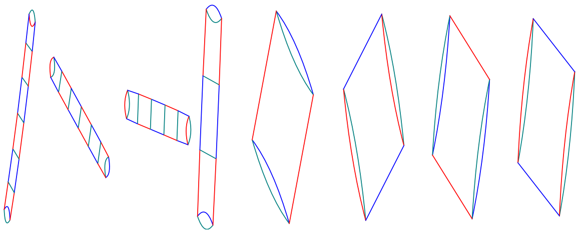


Figure 10.6: Case of $\mathcal{E}_2$ (2 collisions in $S_1$ and 2 collisions in $S_3$).

This is coherent with our experimental results. Indeed, we found that

$$P[\gamma_{\text{win}} = x] \approx Q \cdot \exp(-x/\tau).$$

Using this equation, $\Pr[\text{S}]_n$ can be estimated using the value of $Q$ and that of $\tau$. These two distinct estimations provide a sanity check.

The values of $Q$ and $\tau$ are given for different values of $4 \leq n \leq 11$ in the bijective and non-bijective case described in Section 10.4.3.2 (p. 193) in Tables 10.2a and 10.2b. The final result is Figure 10.7 which shows our estimation of $\Pr[\text{S}]_n$, obtained by averaging our two estimates, and its counterpart in the non-bijective case.

As we can see, there is a discrepancy: the probability is on average about 2.65 times smaller in the case of non-bijective Feistel functions. This can be explained by the massive presence of Type-P cycles. Let $w_i$ be the number of solutions of $S_i(x) = S_i(x \oplus \gamma)$. Then, as explained in Section 10.4.1, there are $(w_1 + w_3) \cdot 2^{n-2}$ Type-P cycles. If $w_1 + w_3 > 0$ then the large number of such cycles we obtain effectively "clogs" the cycle space and prevents the existence of large enough Type-S cycles. In fact, as a consequence of Theorem 9.1.1 (p. 160), $w_i/2$ follows a Poisson distribution with parameter 1/2 when $S_i$ is a random function mapping $n$ bits to $n$. Hence, the probability that $w_i = 0$ is $\exp(-1/2)$ and the probability that $w_1 + w_3 = 0$ is equal to $e^{-1}$, meaning that the probability is about 2.72 times smaller in the non-bijective case. This result is coherent with the ratio of 2.65 we found experimentally.

### 10.4.4 The Cycle-Based Yoyo Cryptanalysis

Exploiting a Type-$S$ cycle is a lot easier than exploiting a Type-$P$ or a pair of Type-$D$ cycles. Indeed, the connected pairs $(x_i, x_{i+\ell/2})$ can be immediately derived from the length $\ell$ of the cycle, while we have to guess a shift amount for connected pairs in a Type-$P$ cycle, or between two type $D$ cycles. Thus, it makes sense to target those specifically, for instance by implementing Algorithm 10.3.

This attacks requires $O(2^{2n}/n)$ blocks of memory to store which plaintexts were visited and $O(2^{2n})$ time. Indeed, at most all elements of the codebook will be evaluated and inspected a second time when attempting a yoyo cryptanalysis on each cycle large enough. Even though the attack must be repeated about $1/\Pr[\text{S}]_n$ times to be able to obtain a large enough Type-S cycle, $\Pr[\text{S}]_n$ increases with $n$ so that $1/\Pr[\text{S}]_n$ can be upper-bounded by a constant independent of $n$.[4] Special points can be used to obtain a time-memory tradeoff: instead of storing whether all plaintexts were visited or not, we only do so for those with, say, the first $\mathcal{B}$ bits equal to 0.

[4] As shown in Section 10.4.3.2 (p. 193), a lower bound of 0.1 is more than sufficient even for $n$ as small as 4.

| $n$ | $Q$ | $\tau$ | $\Pr[\mathrm{S}]_n$ (from $Q$) | $\Pr[\mathrm{S}]_n$ (from $\tau$) | $\Pr[\mathrm{S}]_n$ (avg.) |
|---|---|---|---|---|---|
| 4 | 0.584 | 2.16 | 0.369 | 0.371 | 0.370 |
| 5 | 1.110 | 1.349 | 0.526 | 0.524 | 0.525 |
| 6 | 1.662 | 1.020 | 0.624 | 0.625 | 0.625 |
| 7 | 1.867 | 0.9558 | 0.651 | 0.649 | 0.650 |
| 8 | 2.833 | 0.7427 | 0.739 | 0.740 | 0.740 |
| 9 | 3.254 | 0.6833 | 0.765 | 0.769 | 0.767 |
| 10 | 3.880 | 0.6338 | 0.795 | 0.794 | 0.794 |
| 11 | 4.943 | 0.5679 | 0.832 | 0.828 | 0.830 |

(a) Feistel functions are bijective.

| $n$ | $Q$ | $\tau$ | $\Pr[\mathrm{S}]_n$ (from $Q$) | $\Pr[\mathrm{S}]_n$ (from $\tau$) | $\Pr[\mathrm{S}]_n$ (avg.) |
|---|---|---|---|---|---|
| 4 | 0.173 | 6.549 | 0.147 | 0.142 | 0.144 |
| 5 | 0.2687 | 4.094 | 0.212 | 0.217 | 0.214 |
| 6 | 0.310 | 3.746 | 0.237 | 0.234 | 0.235 |
| 7 | 0.3386 | 3.411 | 0.253 | 0.254 | 0.254 |
| 8 | 0.3837 | 2.969 | 0.277 | 0.286 | 0.281 |
| 9 | 0.4096 | 2.851 | 0.291 | 0.296 | 0.293 |
| 10 | 0.4065 | 2.964 | 0.289 | 0.286 | 0.287 |
| 11 | 0.4019 | 2.964 | 0.287 | 0.286 | 0.286 |

(b) Feistel functions are not bijective.

Table 10.2: Experimentally found expression of $\Pr[\mathrm{S}]_n$.

In this case, the time complexity becomes $O(\mathcal{B} \cdot 2^{2n})$ and the memory complexity $O\left(2^{2n}/(n \cdot \mathcal{B})\right)$. Access to the hash table storing whether an element has been visited or not is a bottle-neck in practice so special points actually give a "free" memory improvement in the sense that memory complexity is decreased without increasing time. In fact, wall clock time may actually decrease. An attack against a $\oplus$-Feistel with $n = 14$ on a regular desktop computer[5] takes about 1 hour to recover both $S_0$ and $S_4$.

### 10.4.5 Attacking 6 and 7 Rounds

It is obviously possible to use the yoyo game to attack 6 and 7 rounds by first guessing the last or the last two Feistel functions and then validating this choice using a yoyo cryptanalysis on the remaining rounds. However, we show below that it is possible to perform attacks which, while having a double exponential complexity, remain much more efficient than the naïve approach just described.

[5] CPU: Intel core i7-3770 (3.40 GHz); 8 Gb of RAM. The program was compiled with g++ along with the optimization flag -O3.

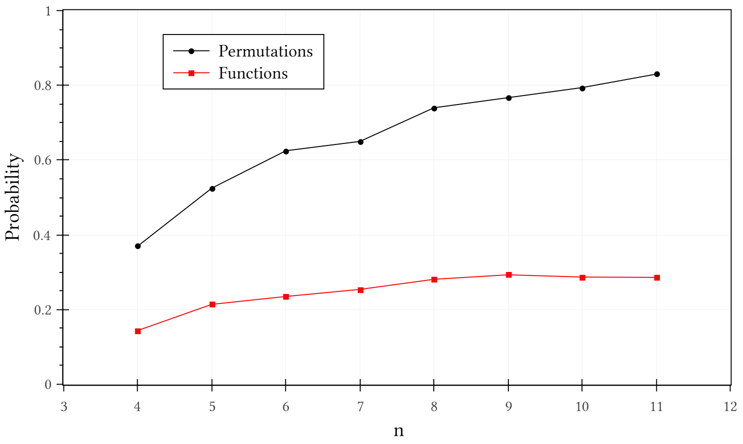


Figure 10.7: Experimentally found probability of having a large enough Type-S cycle when all Feistel functions are permutations (black) and when collisions are allowed (red).

**Algorithm 10.3** Cycle based yoyo cryptanalysis of a 5-round ⊕-Feistel.

```
for all $\gamma \in \{0,1\}^{2n} \backslash \{0\}$ do
    for all $s \in \{0,1\}^{2n}$ do
        if $s$ was not encountered before for this $\gamma$ then
            $C \leftarrow$ empty list
            $x \leftarrow s$
            repeat
                $x \leftarrow \phi(x)$; append $x$ to $C$
                $x \leftarrow \psi(x)$; append $x$ to $C$
            until $x = s$
            if $|C| \geq 2^{n+2}$ then
                Build yoyo game $\mathcal{Y} = (C[0,..,\ell-1], C[\ell,..,2\ell-1])$ with $\ell = \frac{|C|}{2}$
                Run yoyo cryptanalysis (Alg. 10.1) against $\mathcal{Y}$
                if yoyo cryptanalysis is a success then
                    return $S_0, S_4$
                end if
            end if
        end if
    end for
end for
```

#### 10.4.5.1 An Attack on 6 rounds

A naïve approach could consist in guessing all of the entries of $S_5$ and, for each guess, try running a cycle-based yoyo cryptanalysis. If it fails then the guess is discarded. Such an attack would run in time $O\left(2^{n2^n+2n}\right)$. However, it is possible to run such an attack at a cost similar to that of guessing only half of the entries of $S_5$, namely $O(2^{n2^{n-1}+2n})$ which means that it is $2^{n2^{n-1}}$ times faster.

Instead of guessing all the entries, this attack requires guessing the values of

$\Delta_5(x, \gamma) = S_5(x) \oplus S_5(x \oplus \gamma)$. Once these are know, we simply need to replace $\psi_\gamma$ by $\psi'_\gamma$ with

$$\left(\mathcal{E} \circ \psi'_\gamma \circ \mathcal{E}^{-1}\right)(g||h) = (g \oplus \gamma \;||\; h \oplus \Delta_5(x, \gamma))\,.$$

The cycle-based yoyo cryptanalysis can then be run as previously because, again, both $\phi_\gamma$ and $\psi'_\gamma$ preserve connection in $\gamma$. Once it succeeds, the top S-Box is known which means that it can be peeled of. The regular attack is then performed on the remaining 5 rounds. Note that if the yoyo cryptanalysis fails because of inner collisions in $S_1$ or $S_3$ then we can still validate a correct guess by noticing that there are $O(2^n)$ cycles instead of $O(2n)$ as would be expected[6].

In this algorithm, $2^{n-1}$ values of $[0, 2^n - 1]$ must be guessed and for each of those an attack with running time $O(2^{2n})$ must be run. Hence, the total running time is $O\left(2^{n2^{n-1}+2n}\right)$. The time necessary to recover the remainder of the Feistel functions is negligible.

#### 10.4.5.2 An Attack on 7 rounds

A $\oplus$-Feistel with 7 rounds can be attacked in a similar fashion by guessing both $\Delta_0(x, \gamma)$ and $\Delta_6(x, \gamma)$ for all $x$. These guesses allow the definition of $\phi''_\gamma$ and $\psi''_\gamma$, as follows:

$$\begin{aligned} \phi''_\gamma(a||b) &= (a \oplus \Delta_0(x, \gamma) \;||\; b \oplus \gamma) \\ \left(\mathcal{E} \circ \psi''_\gamma \circ \mathcal{E}^{-1}\right)(g||h) &= (h \oplus \Delta_6(x, \gamma) \;||\; g \oplus \gamma)\,. \end{aligned}$$

For each complete guess $((\Delta_0(x, \gamma_0), \forall x), (\Delta_6(x, \gamma_0), \forall x))$, we run a yoyo cryptanalysis. If it succeeds, we repeat the attack for a new difference $\gamma_1$. In this second step, we don't need to guess $2^{n-1}$ values for each $\Delta_0(x, \gamma_1)$ and $\Delta_6(x, \gamma_1)$ but only $2^{n-2}$ as $\Delta_i(x \oplus \gamma_0, \gamma_1) = \Delta_i(x, \gamma_0) \oplus \Delta(x \oplus \gamma_1, \gamma_0) \oplus \Delta_i(x, \gamma_1)$. We run again a cycle-based yoyo cryptanalysis to validate our guesses. The process is repeated $n - 1$ times in total so as to have $\sum_{k=0}^{n-1} 2^k = 2^n$ independent linear equations connecting the entries of $S_0$ and another $2^n$ for the entries of $S_6$. Solving those equations gives the two outer Feistel functions, meaning that they can be peeled off. We then run a regular yoyo cryptanalysis on the 5 inner rounds to recover the remainder of the structure.

Since $\sum_{k=0}^{n-1} 2^{n2^k+2n} = O\left(2^{n2^n+2n}\right)$, the total time complexity of this attack is $O\left(2^{n2^n+2n}\right)$, which is roughly the complexity of a naïve 6-round attack based on guessing a complete Feistel function and running a cycle-based yoyo cryptanalysis on the remainder.

## 10.5 The High-Degree Indicator Matrix of Feistel Networks

The HDIM as described in Definition 8.2.6 (p. 144) can be used to identify Feistel networks efficiently even if many rounds are used provided that the algebraic degree of the round functions is low enough. This section explains why such distinguishers are possible and how they could be used. Section 10.5.1 (p. 198) describes some artifacts which are always present in the HDIM of some Feistel networks. These are used to attack 4-round Feistel networks hidden using affine whitening in Section 10.5.2 (p. 201). Finally, some connections with integral distinguishers are drawn in Section 10.5.3 (p. 203).

[6] A random permutation of a space of size $N$ is expected to have about $\log_e(N)$ cycles.

In what follows, we denote $\mathsf{F}_d^r$ an $r$-round Feistel network with bijective Feistel functions of algebraic degree at most $d$.

### 10.5.1 Artifacts in the HDIM of Feistel Networks

The HDIM of a Feistel network may yield interesting patterns depending on the degree of its Feistel functions, whether they are bijections or not and its number of rounds. These are formalized by Theorem 10.5.1 and its corollary (Corollary 10.5.1). These results link the maximum degree $d$ of the Feistel functions, the number of rounds $r$ and the presence or not of some patterns using the function $\theta : \mathbb{N}^2 \rightarrow \mathbb{N}$ defined by

$$\theta(d,r) = d^{\lfloor r/2 \rfloor - 1} + d^{\lceil r/2 \rceil - 1},$$

where $\lfloor 2k \rfloor = \lfloor 2k+1 \rfloor = 2k$ and $\lceil 2k \rceil = \lceil 2k-1 \rceil = 2k$, so that $\lfloor k \rfloor + \lceil k \rceil = 2k$ for all $k$.

**Theorem 10.5.1.** *Let $F$ be a $2n$-bit $\mathsf{F}_d^r$. Then the HDIM of $F$ is such that $\hat{H}(F)[i,j] = 0$ if $i < n$ **or** $j < n$ under the following conditions:*

- *if the Feistel functions are bijections and $\theta(d,r) < 2n$, or*
- *if the Feistel functions are not bijections and $\theta(d,r+1) < 2n$.*

The general idea of the proof is to express the sum corresponding to coefficient $\hat{H}(F)[i,j]$ using well-chosen variables $(\alpha,\beta)$ located in the middle of the encryption. The value of $F(x)$ is then a function of degree $d^{\lceil r/2 \rceil - 1}$ of $(\alpha,\beta)$ and that of $x$ is a function of degree $d^{\lfloor r/2 \rfloor - 1}$. The coefficients can thus be written as

$$\hat{H}(F)[i,j] = \bigoplus_{(\alpha,\beta) \in (\mathbb{F}_2^n)^2} \left(e_i \cdot F(x(\alpha,\beta))\right) \left(e_j \cdot x(\alpha,\beta)\right)$$

and the result is equal to 0 if $\theta(d,r) = d^{\lfloor r/2 \rfloor - 1} + d^{\lfloor r/2 \rfloor - 1} < 2n$. If the Feistel functions are not bijective then a "trick" used to slightly decrease the degree in $(\alpha,\beta)$ of the output cannot be used, hence the small discrepancy in this case.

In order to formally prove the theorem, we first state the following lemma which can be derived by simply tracking the evolution of the algebraic degree of each branch of the Feistel network.

**Lemma 10.5.1** (Feistel network algebraic degree)**.** *Let $F : x \mapsto F_\ell(x) || F_r(x)$ be a $2n$-bit $\mathsf{F}_d^r$ and let $G : x \mapsto G_\ell(x) || G_r(x)$ be a $2n$-bit permutation such that* $\deg(G_r) = d_G$ *and* $\deg(G_\ell) \leq d \times d_G$*. Then the degree of the left and right words of $F \circ G$ are bounded as follows:*

$$\deg(F_\ell \circ G) \leq d^{r+1} \times d_G \text{ and } \deg(F_r \circ G) \leq d^r \times d_G \ .$$

*Proof of Theorem 10.5.1.* The inverse of a $\mathsf{F}_d^r$ is also a $\mathsf{F}_d^r$. As the HDIM of the functional inverse of a permutation is the transpose of its HDIM, it is sufficient to prove that $\hat{H}(F)[i,j] = 0$ for $i < n$. In this case, $\hat{H}(F)[i,j] = 0$ for $j < n$ is derived immediately by considering the $F^{-1}$. By Lemma 8.2.8, the coefficient $\hat{H}(F)[i,j]$ is equal to

$$\hat{H}(F)[i,j] = \bigoplus_{x \in \mathbb{F}_2^{2n}} \left(e_i \cdot F(x)\right) \left(e_j \cdot x\right) \ .$$

Our proof relies on expressing this sum using another set of variables and showing that the Boolean functions using these variables has an algebraic degree below $2n$, so that it always sums to 0.

Let $F$ be a $2n$-bit $\mathsf{F}_d^r$ built using Feistel functions $f_0, ..., f_{r-1}$. We denote $c$ the input of $f_{\lfloor r/2 \rfloor}$, $a$ the other input of round $\lfloor r/2 \rfloor$ and $b = a \oplus f_{\lfloor r/2 \rfloor}(c)$ the output of round $\lfloor r/2 \rfloor$ which is not equal to $c$ (see Figure 10.8).

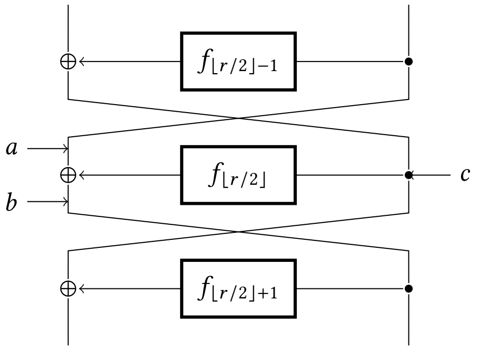

$f_{\lfloor r/2 \rfloor -1}$
$a$
$f_{\lfloor r/2 \rfloor}$
$c$
$b$
$f_{\lfloor r/2 \rfloor +1}$

Figure 10.8: The variables $a, b$ and $c$.

Let us denote $(x_\ell(a,b), x_r(a,b))$ the left and right side of the input of $F$ such that the input of round $\lfloor r/2 \rfloor$ is $(c,a)$ and $(y_\ell(a,b), y_r(a,b))$ the corresponding output. The coefficients of the HDIM of $F$ for $i < n$ can thus be expressed as

$$\bigoplus_{a||b \in \mathbb{F}_2^{2n}} (e_i \cdot y_r(a,b)) \times \left(e_j \cdot x_\ell(a,b) \oplus e_j \cdot x_r(a,b)\right). \qquad (10.3)$$

It is therefore sufficient to find a bound $B$ on the degree in $(a,b)$ of this expression and show that it is below $2n$ to prove the theorem. We will achieve this by looking separately at the degree of $y_r$ and that of $x_\ell || x_r$. For $r = 3$, we have:

$$\begin{cases} x_\ell(a,b) = f_0(a) \oplus c, x_r(a,b) = a \\ y_\ell(a,b) = f_1(b) \oplus c, y_r(a,b) = b. \end{cases}$$

Our bounds are different depending on whether the Feistel functions are bijective or not. We define $B_b(r) = \deg(y_r) + \deg(x_\ell || x_r)$ (bijective case) and $B_c(r) = \deg(y'_r) + \deg(x'_\ell || x'_r)$ (collisions are allowed).

**Bijective case.** If the functions are bijections, we compute $c$ using $c = f_1^{-1}(a \oplus b)$ so that the degrees of $x_\ell, x_r, y_\ell$ and $y_r$ are upper bounded respectively by $d, 1, d$and 1. If $r = 2k + 1$ is odd, we add $k - 1$ Feistel rounds before and after $x_\ell || x_r$ and $y_\ell || y_r$. In this case, Lemma 10.5.1 implies that the degrees become $d^k, d^{k-1}, d^k, d^{k-1}$ so that $B_b(2k+1) \leq d^k + d^{k-1} = \theta(d, 2k+1)$. If $r = 2k$, we add $k - 2$ rounds at the top and $k - 1$ at the bottom which means that the degrees become $d^{k-1}, d^{k-2}, d^k, d^{k-1}$, so that $B_b(2k) \leq d^{k-1} + d^{k-1} = \theta(d, 2k)$. Thus, if the functions are bijections and if $\theta(d, r) < 2n$, then $\hat{H}(F)[i,j] = 0$ for $i < n$.

**Non-bijective case.** If the Feistel functions are not bijections, then we compute $b$ using $b = a \oplus f_1(c)$, sum over $(a||c)$ and look at functions $x'_\ell, x'_r, y'_\ell$ and $y'_r$ taking as input $a$ and $c$ instead of $a$ and $b$.

The degrees of $x'_\ell, x'_r, y'_\ell, y'_r$ are upper bounded respectively by $d, 1, d^2, d$. The same reasoning as above applies, so that if $r = 2k+1$ then we add $k-1$ rounds above and below and the degrees become $d^k, d^{k-1}, d^{k+1}, d^k$. We deduce that $B_c(2k+1) \leq d^k + d^k = \theta(d, 2k+2)$. Similarly, if $r = 2k$ then we add $k-1$ rounds at the top and $k-2$ at the bottom which implies that the degrees become $d^{k-1}, d^{k-2}, d^{k+1}, d^k$. In this case, we deduce that $B_c(2k) \leq d^{k-1} + d^k = \theta(d, 2k+1)$. In the end, we obtain that if the functions are not bijections and if $\theta(d, r+1) < 2n$, then $\hat{H}(F)[i,j] = 0$ for $i < n$. □

**Corollary 10.5.1.** *Let $F$ be a $2n$-bit $\mathsf{F}^r_d$. The* HDIM *of $F$ is such that $\hat{H}(F)[i,j] = 0$ if $i < n$* ***and*** *$j < n$ under the following conditions:*

- *if the Feistel functions are bijections and $\theta(d, r-1) < 2n$, or*
- *if the Feistel functions are not bijections and $\theta(d, r) < 2n$.*

*Proof.* Let $r$ and $d$ be such that $\mathsf{F}^{r-1}_d$ fits the hypothesis of Theorem 10.5.1. The right word of the output of a $\mathsf{F}^r_d$ structure is the left word output by a $\mathsf{F}^{r-1}_d$ structure. As each line of the HDIM corresponds to one output bit, the top $n$ rows of the HDIM of the $r$-round Feistel network are equal to the bottom $n$ rows of the same permutation reduced to $(r-1)$ rounds. Because of Theorem 10.5.1, this bottom half is such that the first $n$ columns are all 0. Thus, the first $n$ columns of the first $n$ rows of the HDIM of a $\mathsf{F}^r_d$ are all equal to 0. □ □

**Observation 10.5.1.** *In Theorem 10.5.1 and Corollary 10.5.1, a distinction is made between the case where the Feistel functions are bijections and the case where they are not. Actually, due to how the proofs work, it would be sufficient to look at the Feistel function at round $k+1$ if the permutation has $2k+1$ rounds and at the Feistel functions of rounds $k$ and $k+1$ if the permutation has $2k$ rounds.*

To illustrate these theorems, we give the HDIM of a 4- and a 5-round Feistel network with 3-bit bijective Feistel functions picked uniformly at random. Since $\theta(2,4) = 2^1 + 2^1 = 4 < 6$, these HDIMs must exhibit the patterns described in the theorems above. It is the case, as we can see below. The zeroes caused by Theorem 10.5.1 and Corollary 10.5.1 are represented in grey:

$$\hat{H}(\mathsf{F}^4) = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 \end{bmatrix}, \hat{H}(\mathsf{F}^5) = \begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}. \tag{10.4}$$

Recall that the HDIM and the congruence modulo 4 of LAT coefficients are linked by Lemma 8.2.9. Thus, the patterns in these HDIM imply the existence of other patterns in the LAT of these functions. These are discussed in Section 12.2.2 (p. 229).

Even though a $\mathsf{F}^r_d$ structure has an algebraic degree of $2n-1$ in the conditions of Theorem 10.5.1, the way in which this high degree is achieved is very structured: only half of the output bits actually have a maximum degree and the monomials of degree $2n-1$ can not contain the product of $n-1$ bits from the right side of the input.

These patterns lead to the existence of distinguishers as long as the conditions necessary for Corollary 10.5.1 are satisfied. Table 10.3 shows the value of the number of rounds for which the conditions of Corollary 10.5.1 are satisfied for different

| $(d, 2n)$ | Feistel functions | $r_{\max}(d, n)$ | Instance |
|---|---|---|---|
| $(2, 32)$ | 1-to-1 | 10 | — |
| | collisions | 9 | SIMON-32 [BSS+13] |
| $(5, 64)$ | 1-to-1 | 7 | — |
| | collisions | 6 | DES [U.S99] |
| $(31, 64)$ | 1-to-1 | 5 | MISTY1/KASUMI [Mat97] |
| | collisions | 4 | — |
| $(n-1, 2n)$ | 1-to-1 | 5 | — |
| | collisions | 4 | — |

Table 10.3: If $r = r_{\max}(d, 2n)$ then the $2n$-bit permutation $\mathsf{F}_d^r$ exhibits an artifact of size $n^2$ in its HDIM.

values of $d, r$ and $n$ in both the 1-to-1 case and the case where collisions in the Feistel functions are allowed. If real ciphers correspond to these parameters, we specify them. Note that the rotation applied to one of the branches in the round function of LBlock [WZ11] does not change anything. The key-dependent linear FL layers in MISTY1 [Mat97] do not protect from our distinguisher as well and may be included from any side for free.

### 10.5.2 Bypassing Affine Whitening

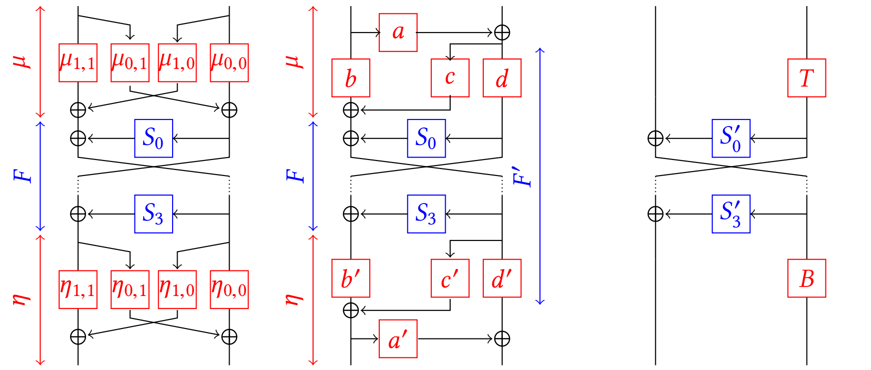
In the context of component reverse-engineering/white-box cryptography, it may not be sufficient to be able to attack a generic Feistel structure. Indeed, simply whitening a generic structure with secret affine layers can prevent many attacks from succeeding at small cost for the designer. For example, applying affine layers before and after a 5-round Feistel network would prevent the profitable use of the yoyo game used in Section 10.3. Similarly, the attacks against ASASA are much more sophisticated than the attack against SASAS (see Chapter 11).

As a consequence, we study the generic construction denoted $\mathsf{AF}_d^r\mathsf{A}$ consisting in a $\mathsf{F}_d^r$ construction with secret Feistel functions preceded and followed by the application of independent and secret linear layers[7]. We published a first attack against this structure in [BPU16] but our attack is significantly more efficient than the one presented in this section.

It is worth mentioning that such structures are used in practice. Indeed, one of the S-Boxes used by the stream cipher ZUC [ETS11] has this structure: it is a 3-round Feistel network composed with a bit rotation. Another decomposition method against this S-Box is presented in Section 12.2.4.3 (p. 235). Let us show how the HDIM and the artifacts we identified in the previous section can be used to attack permutations with $\mathsf{AF}_d^r\mathsf{A}$ structures.

Our attack works for a subset of all possible linear layers. We define $G = \eta \circ F \circ \mu$ where $F$ has a $\mathsf{F}_d^r$ structure satisfying the conditions of Theorem 10.5.1 and $\mu$ and $\eta$

[7] We note that adding constants to make the layers affine is equivalent to replacing the Feistel functions by other ones with identical properties.



(a) $G = \eta \circ F \circ \mu$.

(b) $G$ (alt. representation).

(c) $F'$ (alt. representation).

Figure 10.9: The target of our attack, its result and its alternative representation. In Figure 10.9c, $S'_i$ is affine equivalent to $S_i$.

are linear layers. The layer applied first must have a decomposition as follows:

$$\mu = \begin{bmatrix} \mu_{0,0} & \mu_{0,1} \\ \mu_{1,0} & \mu_{1,1} \end{bmatrix} = \begin{bmatrix} d & 0 \\ c & b \end{bmatrix} \times \begin{bmatrix} I & a \\ 0 & I \end{bmatrix} = \begin{bmatrix} d & d \times a \\ c & b + c \times a \end{bmatrix},$$

and the layer applied last must have a similar one:

$$\eta = \begin{bmatrix} \eta_{0,0} & \eta_{0,1} \\ \eta_{1,0} & \eta_{1,1} \end{bmatrix} = \begin{bmatrix} I & a' \\ 0 & I \end{bmatrix} \times \begin{bmatrix} d' & 0 \\ c' & b' \end{bmatrix} = \begin{bmatrix} d' + a' \times c' & a' \times b' \\ c' & b' \end{bmatrix}.$$

It is sufficient for such a decomposition of the first layer to exist that $\mu_{0,0}$ is invertible. Indeed, we can then simply set $d = \mu_{0,0}$, $c = \mu_{1,0}$, $a = d^{-1} \times \mu_{0,1}$ and $b = \mu_{1,1} - c \times a$. The matrix $b$ has to be invertible since $\mu$ is invertible. Similarly, it is sufficient that $\eta_{1,1}$ is invertible to decompose the final layer. Using these decompositions, we define $F'$ and $G$ as follows:

$$G = \begin{bmatrix} I & a' \\ 0 & I \end{bmatrix} \circ \underbrace{\begin{bmatrix} d' & 0 \\ c' & b' \end{bmatrix} \circ F \circ \begin{bmatrix} d & 0 \\ c & b \end{bmatrix}}_{F'} \circ \begin{bmatrix} I & a \\ 0 & I \end{bmatrix}.$$

A graphical representation of the relation between $F$, $F'$ and $G$ is provided in Figures 10.9a and 10.9b. As $F$ satisfies the condition of Theorem 10.5.1, its HDIM is such that $\hat{H}(F)[i,j] = 0$ if $i < n$ or $j < n$. The HDIM of a permutation composed with a linear permutation is given by Lemma 8.2.11. We deduce that the HDIM of $F'$ is equal to

$$\hat{H}(F') = \begin{bmatrix} d' & 0 \\ c' & b' \end{bmatrix} \times \hat{H}(F) \times \begin{bmatrix} d & c \\ 0 & b \end{bmatrix}^{-1} = \begin{bmatrix} 0 & 0 \\ 0 & h' \end{bmatrix} \text{with } h' = b' \times h \times b^{-1},$$

$h$ being the bottom-right part of $\hat{H}(F)$. Like in $\hat{H}(F)$, it holds that $\hat{H}(F')[i,j] = 0$ if $i < n$ or $j < n$. Another way to see why this holds is shown in Figure 10.9c. Indeed, $F'$ can be written as a $\mathsf{F}_d^r$ structure, like $F$, where $n$-bit linear permutations are applied only on two branches and where the Feistel functions $f'_i$ are obtained from compositions of $b, b', d, d'$ and $f_i$, as well as the addition of $c$ and $c'$ for the first and last rounds. We deduce that if $G$ indeed has a $\mathsf{AF}_d^r\mathsf{A}$ structure satisfying the conditions for Theorem 10.5.1, then the following equation, which has as unknowns the $n \times n$ binary matrices $a$ and $a'$, must have at least one solution:

$$\begin{bmatrix} I & a' \\ 0 & I \end{bmatrix} \times \hat{H}(G) \times \begin{bmatrix} I & 0 \\ a & I \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 0 & h_{1,1} \end{bmatrix},$$

where $h_{1,1}$ is the bottom right corner of $\hat{H}(G)$. This system has $2n^2$ unknowns and $3n^2$ equations, meaning that it is unlikely to have solutions if $G$ is a random permutation. However, if it does have a solution then we deduce both that $G$ has an $\mathsf{AF}_d^r\mathsf{A}$ structure and the expression of parts of the linear layers. We summarize these results in the following attack.

**Attack 10.5.1** (Partial Recovery Against $\mathsf{AF}_d^r\mathsf{A}$)**.** *Let $G$ be a $2n$-bit permutation. It is necessary for $G$ to be in $\mathsf{AF}_d^r\mathsf{A}$ for some $(r,d)$ satisfying Theorem 10.5.1 that the equation*

$$\begin{bmatrix} I & a' \\ 0 & I \end{bmatrix} \times \hat{H}(G) \times \begin{bmatrix} I & 0 \\ a & I \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 0 & h_{1,1} \end{bmatrix},$$

*where $h$ is an unknown $n \times n$ matrix, has at least one solution. The unknowns are the coefficients of the $n \times n$ matrices $a$ and $a'$, so that $2n^2$ Boolean variables must satisfy $3n^2$ equations corresponding to the zeroes in the right hand side.*

This distinguisher requires the full code-book and as much time as is needed to compute the HDIM and solve a system of equations. Since the system is small, the bottle-neck is the computation of the HDIM which can be done in time $O(n2^{2n})$ where $n$ is the branch size.

We can use the exact same reasoning to attack one more round if the decomposition of $\eta$ and $\mu$ involve the same "linear Feistel function" $a$. This happens in particular if $\eta = \mu^{-1}$. In this case, we can use the distinguisher obtained from the following attack.

**Attack 10.5.2** (Partial Recovery Against $\mathsf{A}^{-1}\mathsf{F}_{r+1}^d\mathsf{A}$)**.** *Let $G$ be a $2n$-bit permutation. In order for $G$ to be in $\mathsf{AF}_d^r\mathsf{A}$ for some $(r,d)$ satisfying Corollary 10.5.1 in such a way that the linear layers are the inverse of one another, it is necessary that the equation*

$$\begin{bmatrix} I & a \\ 0 & I \end{bmatrix} \times \hat{H}(G) \times \begin{bmatrix} I & 0 \\ a & I \end{bmatrix} = \begin{bmatrix} 0 & h_{0,1} \\ h_{1,0} & h_{1,1} \end{bmatrix},$$

*where $h_{0,1}, h_{1,0}$ and $h_{1,1}$ are unknown $n \times n$ matrices, has at least one solution. The unknowns are the coefficients of the $n \times n$ matrices $a$, so that $n^2$ Boolean variables must satisfy $n^2$ equations corresponding to the zero in the right hand side.*

Note that if there is a single whitening affine layer applied at some side, we have a similar system with $n^2$ unknowns. If we consider one more round, we will have $n^2$ equations as well. Therefore we can attack $\mathsf{F}_d^r\mathsf{A}$, where $r$ is the maximum number of rounds satisfying Corollary 10.5.1.

### 10.5.3 Relationship Between HDIM-based and Integral/Zero-Sum Distinguishers

The HDIM has a simple integral interpretation. Indeed, the HDIM coefficients correspond to ANF coefficients which can be computed using the Möbius transform:

$$\hat{H}(F)[i,j] = \bigoplus_{x \preccurlyeq \overline{e_j}} F_i(x)$$

where $\overline{e_j}$ is the vector where all elements are equal to 1 except in position $j$. Due to the computation method, coefficients equal to 0 effectively correspond to zero-sum distinguishers much like those obtained in an integral attack. This has two consequences. First, we can compute the HDIM of an $n$-bit permutation in time $O(n2^{n-1})$, and, second, zeroes in column $j$ imply the existence of an integral distinguisher.

In light of this, we state the following corollary of Corollary 10.5.1.

**Corollary 10.5.2** (Zero-sum Distinguisher for $\mathsf{F}_d^r$)**.** *Let $F$ be a $2n$-bit $\mathsf{F}_d^r$ and suppose that one of the following conditions holds:*

- *the Feistel functions are bijections and $\theta(d, r-1) < 2n$, or*
- *the Feistel functions are not bijections and $\theta(d, r) < 2n$.*

*Then there exists a zero-sum distinguisher with data and time complexity $2^{2n-1}$ for this structure, namely*

$$\bigoplus_{x \preccurlyeq \overline{e_j}} (e_i \cdot F(x)) = 0$$

*for all $i < n$ and $j < n$. In other words, the sum of the right words of $F(x)$ is equal to 0 over a cube where one bit of the input right word is fixed to 0.*

We notice a relation between our attacks and the so-called *division property*. This tool for finding integral attacks was introduced by Todo in [Tod15b] and later used by the same author to attack full-round MISTY1 [Tod15a]. In his seminal paper, Todo gives some integral distinguishers against Feistel networks for various block sizes, number of rounds, degree of the Feistel functions for both bijective and non-bijective Feistel functions. Interestingly, his results are extremely similar to ours! Indeed, while there is no generic formula in Todo's paper, the application of his algorithm shows the existence of linear spaces of dimension $2n-1$ whose sum is equal to 0 for a number of rounds identical to the ones we predicted. In fact, results about the division property of the output of a Feistel network can be extracted from its HDIM. To explain this, we first recall the definition of the division property.

**Definition 10.5.1** (Division Property)**.** *Let $\mathbb{X}$ be a multiset of $\mathbb{F}_2^n$ and $k$ be an integer of $[0, n]$. We say that $\mathbb{X}$ has the division property $\mathcal{D}_k^n$ if, for all $u$ in $\mathbb{F}_2^n$ such that $hw(u) \leq k$, $\bigoplus_{x \in \mathbb{X}} x^u = 0$.*

This property is further generalized into the *collective division property* whose description we simplify to only encompass the case of 2-branched Feistel networks.

**Definition 10.5.2** (Collective Division Property (for Feistel networks))**.** *Let $\mathbb{X}$ be a multiset of $(\mathbb{F}_2^n)^2$ and $k^L, k^R$ be integers of $\{0, ..., n\}$. We say that $\mathbb{X}$ has the* collective division property *$\mathcal{D}_{(k^L, k^R)}^n$ if, for all $u, v$ in $\mathbb{F}_2^n$ such that $hw(u) \leq k^L$ and $hw(v) \leq k^R$, $\bigoplus_{(x,y) \in \mathbb{X}} x^u y^v = 0$.*

In particular, Todo applied his technique to $2n$-bit $\mathsf{F}_d^r$. The integral distinguisher against the highest number of rounds corresponds to integrals over linear spaces of dimension $2n-1$ where the constant bit has to be on the left side.[8] As we have seen, summing over such a space is equivalent to computing half of the lines of the HDIM of the function.

[8] It is actually on the right side in Todo's paper. Unlike in our paper, the Feistel function is XORed in the right branch in his case.

Let $F$ be a $2n$-bit $\mathsf{F}^r_d$, $x$ denote the left input bits, $y$ denote the right ones and $F_L$ and $F_R$ denote its left and right output halves so that $F(x||y) = F_L(x||y) \;||\; F_R(x||y)$. Suppose that the top left corner of the HDIM of $F$ is all zero. We deduce that the following holds for any $i \leq n$ and for any linear space $C_k$ of dimension $2n-1$ where the bit at index $k \leq n$ is fixed: $\bigoplus_{x \in C_k} F(x) \cdot e_i(x) = 0$. This can also be written as

$$\bigoplus_{x \in C_k} (F_L(x))^{u_i} \, (F_R(x))^0 = \bigoplus_{x \in C_k} (F_L(x))^{u_i} = 0 \;,$$

where $u_i$ is the element of $\mathbb{F}_2^n$ equal to 0 except at position $i$ where it is equal to 1. In other words, for all $u$ in $\mathbb{F}_2^n$, $\mathrm{hw}\,(u) \leq 1$ implies that $\bigoplus_{x \in C_k} (F_L(x))^u = 0$, which means that the image of $C_k$ has vectorial division property $\mathcal{D}^n_{1,0}$. The HDIM of Feistel networks can thus be interpreted as describing the vectorial division property of each output half.

Chapter 11

# Structural Attacks Against Spns

A popular choice for building S-Boxes is the Substitution-Permutation Network structure. It has also been used to build large incompressible permutations in the context of white-box crypto, e.g. by Biryukov et al. [BBK14] and by Bogdanov et al. [BIT16]. In this chapter, I present several methods that can be used to decompose such permutations. These can be used either to decompose an unknown S-Box or to attack a white-box block cipher.

The notation used throughout this chapter is described in Section 11.1 (p. 207). Section 11.2 (p. 208) summarizes the results about the decomposition of Substitution-Permutation Networks with secret components. The rest of the chapter deals with these attacks. General results about spn with two non-linear layers are extracted in Section 11.3 (p. 209). The attacks from the literature against schemes with 2- and 3-non-linear rounds are reinterpreted using these results in Section 11.4 (p. 210). Then, Section 11.5 (p. 212) presents a generalization of the distinguisher presented in Section 11.3 (p. 209) which can be used to attack more rounds when the S-Boxes are small enough. Finally, some connections between these attacks and the division property are presented in Section 11.6 (p. 222).

## 11.1 Notation

We consider block ciphers operating on $n$ bits by alternating $n$-bit linear permutations operating on the full state and $n$-bit S-Box layers consisting of the parallel application of $n/m$ permutations of $m$ bits.

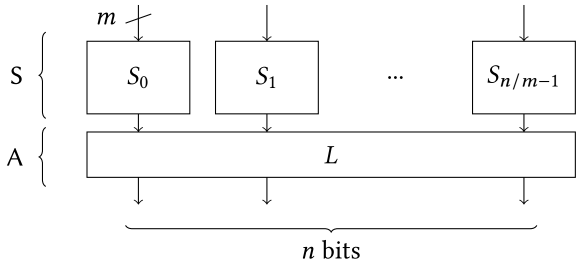


Figure 11.1: A secret spn round SA.

The linear layers may be different. We use A to denote any such function. Similarly, the S-Box layers may be different in each round and, during each round, the $n/m$ permutations used may also be different from one another. However, we assume that they always have the maximum algebraic degree of $m-1$. The composition of such A and S is denoted

$$\mathrm{A}(\mathrm{SA})^q = \mathrm{A}\underbrace{\mathrm{SASA}\ldots\mathrm{SA}}_{2q\text{layers}} .$$

An affine space $C$ is obtained from a vector space $\mathcal{L}$ and a constant $a$ as follows: $C = \{a \oplus x, x \in \mathcal{L}\}$. The set of all affine spaces of $\{0,1\}^n$ of dimension $c$ is denoted $\mathbb{A}_n(c)$ and the union $\cup_{\ell=c}^{n} \mathbb{A}_n(\ell)$ is denoted $\mathbb{A}_n(\geq c)$.

For a predicate $P$ we write

$$[P] = \begin{cases} 1, & \text{if } P \text{ is true;} \\ 0, & \text{if } P \text{ is false .} \end{cases}$$

The coefficients in the algebraic normal form (ANF) of function $f : \mathbb{F}_2^n \rightarrow \mathbb{F}_2$ are denoted $a_u^f$ so that

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

The attacks presented in this chapter may have complexities close to $2^n$ but the aim here is not to recover a secret key as long as the block size, as would be the case when attacking a block cipher. The structures attacked contain far more entropy as all of their components are secret.

## 11.2 Overview of the Structural Attacks Against SPNs

As early as 1997, Patarin and Goubin proposed building a public key crypto-system out of ASASA structures [PG97]. In the same paper, they presented efficient attacks against ASA. In their scheme, the non-linear layers S were not bijections. Biham quickly found an attack against this scheme [Bih00].

Shortly after, Biryukov and Shamir [BS01, BS10] presented a generic attack against the SASAS structure when the S-Boxes are bijective. This attack is described in Section 11.4.1 (p. 210).

This type of structure regained interest following the publication of white-box schemes based on ASASA by Biryukov *et al.* at Asiacrypt'2014 [BBK14]. In this paper, two types of schemes are proposed. The first consists of public key schemes using expanding S-Boxes which were subsequently attacked by Gilbert *et al.* [GPT15]. The second consists of white-box schemes,[1] that is, encryption algorithms whose implementation requires significantly large tables and cannot be compressed unless a secret is known. They use bijective components. Two independent teams found practical attacks against them: Minaud *et al.* [MDFK15] and Dunkelman *et al.* [DDKL15].

Dmitry Khovratovich, with the help of Alex Biryukov and myself [BKP17], found generalizations of these attacks to more rounds when the S-Box size is small enough compared to the block size.

[1] This notion is discussed in Chapter 15 (p. 303). As is explained there, this type of white-box encryption is *asymmetrically code-hard.*

## 11.3 Analysis of SASA

In this section, I present a very efficient distinguisher for the SASA structure. A simple zero-sum distinguisher for AS is described in Section 11.3.1 (p. 209) and then extended essentially for free to the SASA structure in Section 11.3.2 (p. 209). Finally, an equivalence between different instances of secret SPNs is discussed in Section 11.3.3 (p. 210).

### 11.3.1 A Simple Degree-based Zero-sum

The following lemma is well known and is easily derived using the Möbius transform described in Definition 8.2.3.

**Lemma 11.3.1** (Degree-based 0-sum)**.** *Let $f : \mathbb{F}_2^n \rightarrow \mathbb{F}_2^n$ be a function with degree* $\deg(f) = d$*. Let $C$ be an affine space of dimension at least $d + 1$. Then the following sum is equal to zero for any $C$ in $\mathbb{A}_n(\geq d + 1)$:*

$$\bigoplus_{x \in C} f(x) = 0 \ .$$

A direct consequence of Lemma 11.3.1 is that $\bigoplus_{x \in C} \mathrm{ASA}(x) = 0$ for all $C$ in $\mathbb{A}_n(\geq m)$. Indeed, since S consists of the parallel application of $n/m$ functions of degree $m - 1$, it has degree $m - 1$ as well. This algebraic degree is not changed by the compositions with A. It is therefore particularly easy to distinguish a permutation based on ASA from a random one.

### 11.3.2 Free S-Box Layer Addition

The vector space consisting of all possible inputs for an S-Box is mapped to itself. Similarly, an affine space built from this vector space is mapped to another affine space built from the same vector space. Using terminology from [BS01], this observation corresponds to the propagation of the "permutation" pattern through an S-Box. This observation can be generalized.

**Lemma 11.3.2** (Free S-Box Layer)**.** *Let $G$ be a permutation of degree* $\deg(G) < k \times (m - 1) < n$ *and let $S$ be a layer consisting of $n/m$ parallel $m$-bit S-Boxes. If $C$ is an affine space of dimension $k \times (m - 1)$ which covers the inputs of $k$ distinct S-Boxes, then*

$$\bigoplus_{x \in C} (G \circ S) = 0.$$

*Proof.* Because the S-Boxes are bijections, an affine space $C = \mathcal{L} + a$ as specified in the statement is mapped to $\mathcal{L} + b$. Since it is an affine space of dimension $k \times (m - 1) > \deg(G)$, because of Lemma 11.3.1, the sum $\bigoplus_{x \in C} G(x)$ is equal to 0. □

We remark that, in Lemma 11.3.2, the integer $k$ does not need to be the smallest possible. It works for all $k$ such that $\deg(G) < k \times (m - 1) < n$.

The algebraic degree of ASA with $m$-bit full degree S-Boxes is equal to $m - 1$. We can thus apply Lemma 11.3.2 to deduce a distinguisher against SASA.

**Lemma 11.3.3** (Distinguisher for SASA)**.** *Consider a SASA construction and let $k > 0$ be an integer. For all affine spaces $C$ of $\mathbb{A}_n (\geq k \times m)$ covering the input of $k$ S-Boxes, the following holds:*

$$\bigoplus_{x \in C} SASA(x) = 0.$$

Unlike in Lemma 11.3.1, the exact structure of the affine spaces matters in this case. Indeed, if the affine space does not correspond to S-Box inputs, the sum may not be equal to 0. This creates an asymmetry between affine spaces of equal size: those corresponding to S-Box inputs are bound to yield zero-sums while the others can sum to anything.

### 11.3.3 Equivalences in Secret spns

A given permutation with a SA structure has several equivalent decomposition with this overall structure. Indeed, it is possible to compose each S-Box with any $m$-bit affine permutation and compose the affine layer with its inverse to get a functionally equivalent SA structure.

Moreover, in an ASA structure, we can add an affine permutation before each S-Box in the same fashion. It is also possible to permute the order of the S-Box provided that the inverse permutation is factored in both the input and the output linear layer.

In what follows, I present methods recovering the components of a secret SNP. However, those methods only recover one equivalent representations.

## 11.4 Component Recovery Attacks

While they were not introduced in this fashion by their authors, the attack against SASAS from [BS01] and the integral against ASASA from [DDKL15] can be interpreted as adding another round before or after SASA and leveraging Lemma 11.3.3. These are explained in Sections 11.4.1 (p. 210) and 11.4.2 (p. 211) respectively.

### 11.4.1 S-Box Recovery Against SASAS

The attack against SASAS described in [BS01, BS10] uses the SASA distinguisher provided by Lemma 11.3.3. Let $\mathcal{E}$ be a SASAS construction with secret component, let $S_i$ denote the $i$-th S-Box of the final layer and let $(y_0^j, ..., y_{n/m-1}^j)$ denote the image of the plaintext $(j, 0, ..., 0)$.

Because of Lemma 11.3.3, the following holds for all $i < n/m$:

$$\bigoplus_{j=0}^{2^m} S_i^{-1}(y_i^j) = 0.$$

Indeed, this sum is equal to the sum over a part of the output of a SASA construction, as illustrated in Figure 11.2. We deduce a first linear equation connecting the entries of $S_i^{-1}$ to one another for each $i$. We then iterate this process by replacing the constants equal to 0 by others. Doing so about $2^m$ times gives us $2^m$ equations connecting the entries of $S_i^{-1}$ to one another. Solving the corresponding system gives us the look-up table of $S_i^{-1}$ for all $i < n/m$ and, thus, the last S-Box layer of this construction.

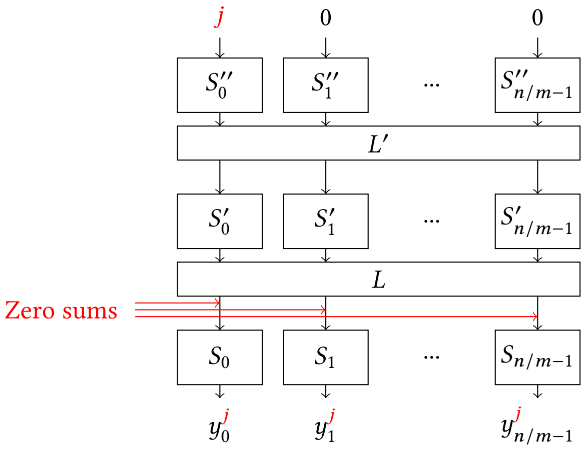


Figure 11.2: The principle of the attack against SASAS.

### 11.4.2 Linear-layer Recovery Against ASASA

Two main attacks target this structure. The first is presented by Dunkelman *et al.* in [DDKL15] and it exploits the distinguisher against SASA of Lemma 11.3.3. The second, by Minaud *et al.* [MDFK15], is based on a simple algebraic observation differentiating linear masks related to the last linear layer from others.

Both attacks aim at recovering the vector spaces $V_i$ of dimension $m$ corresponding to the image of the output of the S-Box $S_i$ by the last linear layer. Once those have been recovered, the last linear layer is deduced.

**Integral Approach.** Algorithm 1 from [DDKL15] starts from a random vector space of size $d > m$. If the permutation sums to 0 over it, it looks for smaller subspaces that still sum to 0 until one of size $d$ is found. This procedure is repeated enough times so as to recover $n/m$ distinct vector spaces $V_i$.

This technique works as long as $m > \sqrt{n}$ and, as is always the case in this chapter, if the S-Boxes have maximum algebraic degree.

**Algebraic Approach.** The approach of Minaud *et al.* is different. The key observation behind their work is a difference in the algebraic degree of the scalar product of two output bits from the same S-Box and from different ones.

**Lemma 11.4.1** (Lemma 2 of [MDFK15]). *Let $s_u$ and $s_v$ be two output bits of the $m$-bit permutation $s$. Then the algebraic degree of $s_u \cdot s_v : x \mapsto s_u(x) \cdot s_v(x)$ is at most $m - 1$.*

For the sake of completeness, here is a proof of this lemma.

*Proof.* Since $s$ has $m$ inputs, the algebraic degree of $s_u \cdot s_v$ is at most $m$. To prove that it is not equal to $m$, it is sufficient to prove that $x_0 ... x_{m-1}$ does not appear in its ANF which, by the Möbius transform, is equivalent to $\bigoplus_{x \in \mathbb{F}_2^m} s_u(x) \cdot s_v(x) = 0$. The left hand side is equal to

$$\bigoplus_{x \in \mathbb{F}_2^m} s_u(x) \cdot s_v(x) = \bigoplus_{x \in \mathbb{F}_2^m, s_v(x)=1} s_u(x).$$

As $s$ is a permutation, the function $s_v$ restricted to the preimage of 1 by $s_v$ is balanced. If it were not the case, then $s$ would have inner collisions. Thus, it holds that $\bigoplus_{x \in \mathbb{F}_2^m} s_u(x) \cdot s_v(x) = 0$. □

This bound contrasts with the one that exists for the scalar product of outputs from two different S-Boxes. As those have different inputs and have algebraic degree at most $m-1$, such a product has degree at most $2(m-1)$. By composing S with SA, we obtain the result that the algebraic degree of the scalar product of two outputs from the same S-Box is at most equal to $(m-1)^2$ while it is bounded by $2(m-1)^2$ if the outputs are from different S-Boxes.

Let $e_u$ and $e_v$ be two vectors of $\mathbb{F}_2^n$ with only one non-zero coordinate at position $u$ and $v$ respectively. Let also $\mathcal{E}$ be an ASAS instance. If $e_u$ and $e_v$ have their non-zero component over the output of the same S-Box, then $x \mapsto (e_u \cdot \mathcal{E}(x)) \times (e_v \cdot \mathcal{E}(x))$ sums to zero over any space of $\mathbb{A}_n\big(\geq (m-1)^2+1\big)$. If $e_u$ and $e_v$ cover different S-Box outputs, it is not the case.

Let $L$ be some affine permutation of $n$ bits. Let us consider the ASASA construction $\mathcal{E}' = L \circ \mathcal{E}$ and let $C$ be some vector space of $\mathbb{A}_n\big(\geq (m-1)^2+1\big)$. The following holds:

$$\begin{aligned} 0 &= \bigoplus_{x \in C} (e_u \cdot \mathcal{E}(x)) \times (e_v \cdot \mathcal{E}(x)) \\ &= \bigoplus_{x \in C} \Big(e_u \cdot (L^{-1} \circ \mathcal{E}')(x)\Big) \times \Big(e_u \cdot (L \circ \mathcal{E}')(x)\Big) \\ &= \bigoplus_{x \in C} \Big((L^{-1})^t(e_u) \cdot \mathcal{E}'(x)\Big) \times \Big((L^{-1})^t(e_u) \cdot \mathcal{E}'(x)\Big). \end{aligned}$$

The principle of the attack then consists of finding vectors $a_i$ and $b_i$ such that $\bigoplus_{x \in C} (a_i \cdot \mathcal{E}'(x)) \times (b_i \cdot \mathcal{E}'(x)) = 0$ for $C$ in $\mathbb{A}_n\big((m-1)^2+1\big)$. Those give information about $(L^{-1})^t(e_u)$ for all $u$ and, thus $L$. The exact details of this linear layer recovery can be found in [MDFK15].

It needs to generate $n(n-1)/2$ vector spaces of dimension $(m-1)^2+1$. Those are obtained by fixing any 2 variables from the input to 0, so it is necessary that $(m-1)^2+1 \leq n-2$. Thus, this attack requires that $(m-1)^2 < n-2$.

## 11.5 Attacking More Rounds

The recovery attacks described in Section 11.4.1 (p. 210) rely on the small algebraic degree of $(\mathrm{SA})^2$. Different heuristics are then used to attack the additional final S-layer of a SASAS construction or the final A-layer of an ASASA scheme. In this section, we bound the algebraic degree of $(\mathrm{SA})^q$ for higher values of $q$ and deduce attacks recovering the components of secret SPNs with more rounds provided that the S-Box size is small enough.

These results are obtained from a result by Boura *et al.* which is recalled in Section 11.5.1 (p. 213). Then, we extract a closed formula for the number of rounds that can be attacked in Section 11.5.2 (p. 213). Said attacks are described in Section 11.5.3 (p. 218).

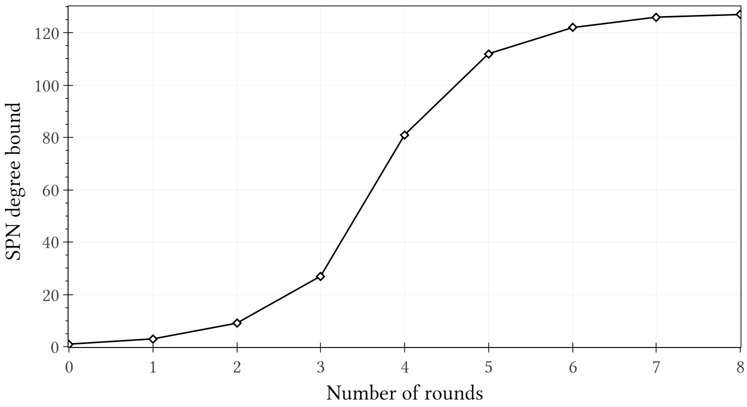


Figure 11.3: Evolution of the maximum algebraic degree of an SPN with 128-bit blocks and 4-bit S-Boxes as bounded by Proposition 11.5.1.

### 11.5.1 The Recursive Degree Bound of Boura *et al.*

**Proposition 11.5.1** ([BCD11]). *Let $P$ be an arbitrary function on $\mathbb{F}_2^n$. Let $S$ be an S-Box layer of $\mathbb{F}_2^n$ corresponding to the parallel application of $m$-bit bijective S-Boxes of degree $m-1$. Then*

$$\deg(P \circ S) \leq n - \left\lceil \frac{n - \deg(P)}{m-1} \right\rceil .$$

This proposition captures the influence of the fact that an S-Box layer consists of the parallel application of several smaller functions. Note in particular that if $m = n$, which corresponds to the case where one S-Box is applied to the full state, this bound does not give new information: it merely states that $\deg(P \circ S) \leq n - 1$, which is obviously the case since it is a permutation. When the S-Box layers consists of several smaller S-Boxes however, it implies a degree discrepancy as illustrated in Figure 11.3 which shows the evolution of the maximum degree of an SPN with $m = 4$ and $n = 128$: starting from $r = 4$, the degree increase is much slower. It reaches the maximum of 127 only after 8 rounds, meaning that a simple integral distinguisher exists for up to 7 rounds.

The principle behind this bound was first used to derive zero-sum distinguishers for the permutation updating the internal state of Keccak [BC11]. Its statement was then refined into its current form in [BCD11].

### 11.5.2 Closed Formulas for a Degree Bound

In what follows, we use $\overline{x_\ell ... x_0}^d$ to denote the base $d$ expansion of $x$, where $\ell = \lfloor \log_d(x) \rfloor$:

$$x = \overline{x_\ell ... x_1 x_0}^d, \text{ where } x = \sum_{i=0}^{\ell} x_i d^i .$$

The following theorem allows us to study the evolution of the algebraic degree of an SPN based on Proposition 11.5.1.

**Theorem 11.5.1.** *For all $\ell \leq \lfloor \log_d(n) \rfloor$, it holds that:*

$$\deg\left(A(SA)^{\ell+r}\right) \leq n - \left(\psi_r + \left\lfloor \frac{n}{d^r} \right\rfloor - \frac{d^\ell}{d^r}\right) ,$$

*where*

$$\psi_i = \begin{cases} 0 \text{ if } n_{i-1} = n_{i-2} = \ldots = n_0 = 0, \\ 1 \text{ otherwise.} \end{cases}$$

*Furthermore, if $\ell = \lfloor \log_d(n) \rfloor$ and $n_\ell \ldots n_1 n_0$ is the base $d$ expansion of $n$, we have*

$$\left\lfloor \frac{n}{d^r} \right\rfloor = \sum_{i=r}^{\ell} n_i d^{i-r} ,$$

*so that if we need $\deg\left(A(SA)^{\ell+r}\right) \leq n - k$, then it is sufficient that*

$$\psi_r + \left\lfloor \frac{n}{d^r} \right\rfloor - \frac{d^\ell}{d^r} \geq k \text{ or, equivalently, } (n_\ell - 1)d^{\ell-r} + \sum_{i=r}^{\ell-1} n_i d^{i-r} \geq k - \psi_r .$$

*Proof.* We bound the algebraic degree of $r$ SPN rounds using $\theta_r$: $\deg(A(SA)^r) \leq \theta_r$. Obviously, $\theta_0 = 1$ holds as $(SA)^0$ is the identity. For larger values, $\theta_r$ is bounded in three different ways: the natural bounds $d^r$ and $n-1$, and the one from Proposition 11.5.1:

$$\theta_r \leq n - \left\lceil \frac{n - \theta_{r-1}}{d} \right\rceil .$$

As long as the first bound prevails, the expression of $\theta_r$ is very simple: $\theta_r = d^r$.

We now consider a larger number of rounds. Let $\ell$ be such that $\ell \leq \log_d(n)$. It holds that $\theta_{\ell+1} \leq n - \left\lceil (n - d^\ell)/d \right\rceil$, which, using the base $d$ expansion of $n$, is equal to

$$\theta_{\ell+1} \leq n - \left\lceil \frac{\sum_{i=0}^{\infty} n_i d^i - d^\ell}{d} \right\rceil .$$

Because all coefficients in the numerator except $n_0$ can be divided by $d$, this quantity is equal to:

$$\theta_{\ell+1} \leq n - \left(\sum_{i=1}^{\infty} n_i d^{i-1} - d^{\ell-1} + \left\lceil \frac{n_0}{d} \right\rceil\right) .$$

Finally, we note that $\sum_{i=1}^{\infty} n_i d^{i-1} = \lfloor n/d \rfloor$ and conclude that

$$\theta_{\ell+1} \leq n - \left(\left\lfloor \frac{n}{d} \right\rfloor - d^{\ell-1} + \left\lceil \frac{n_0}{d} \right\rceil\right) .$$

In fact, we can generalize this equality using a simple induction for $r \leq \ell$. To simplify its writing, we define $\psi_i$ inductively using $\psi_1 = \lceil n_0/d \rceil$ and $\psi_{i+1} = \lceil (n_i + \psi_i)/d \rceil$. Our induction hypothesis is then

$$\theta_{\ell+r} \leq n - \left(\left\lfloor \frac{n}{d^r} \right\rfloor - d^{\ell-r} + \psi_r\right) , \tag{11.1}$$

and we have established that it holds for $r = 1$. Suppose now that it holds for some $r$. Using Proposition 11.5.1, we deduce that

$$\theta_{\ell+r+1} \leq n - \left\lceil \frac{n - \left(n - \lfloor n/d^r \rfloor + d^{\ell-r} - \psi_r\right)}{d} \right\rceil ,$$

which implies

$$\theta_{\ell+r+1} \leq n - \left\lceil \frac{\sum_{i=r}^{\infty} n_i d^{i-r} - d^{\ell-r} + \psi_r}{d} \right\rceil .$$

Using again that all $d^{i-r}$ for $i \geq r$ are divisible by $d$ except for $d^0$, we can rewrite this inequality as

$$\theta_{\ell+r+1} \leq n - \left( \sum_{i=r}^{\infty} n_i d^{i-r-1} - d^{\ell-r-1} + \left\lceil \frac{n_r + \psi_r}{d} \right\rceil \right).$$

We simplify this expression using that $\sum_{i=r}^{\infty} n_i d^{i-r-1} = \lfloor n/d^{r+1} \rfloor$ and the definition of $\psi_{r+1}$ and obtain

$$\theta_{\ell+r+1} \leq n - \left( \left\lfloor \frac{n}{d^{r+1}} \right\rfloor - d^{\ell-r-1} + \psi_{r+1} \right).$$

Let us simplify this expression. First, the quantity $\lfloor n/d^r \rfloor - d^{\ell-r}$ can be written using the base $d$ expansion of $n$:

$$\lfloor n/d^r \rfloor - d^{\ell-r} \leq \sum_{i \geq r,\ i \neq \ell} n_i d^{i-r} + (n_r - 1) d^{\ell-r}.$$

Furthermore, all $n_i$ for $i > \ell$ are equal to 0. Using this, the inequality becomes:

$$\lfloor n/d^r \rfloor - d^{\ell-r} \leq \sum_{i=r}^{\ell-1} n_i d^{i-r} + (n_r - 1) d^{\ell-r}.$$

Second, we can easily compute $\psi_i$ using the base $d$ expansion of $n$. We again proceed inductively using the following hypothesis:

$$\psi_i = \begin{cases} 0 \text{ if } n_{i-1} = n_{i-2} = \ldots = n_0 = 0, \\ 1 \text{ otherwise.} \end{cases}$$

The equality obviously holds for $i = 0$ as $\lceil n_0/d \rceil = 0$ if and only if $n_0 = 0$, otherwise it is equal to 1 because $n_j < d$ for all $j$. Assuming the equality holds for $i$, let us now compute $\psi_{i+1}$. By definition,

$$\psi_{i+1} = \left\lceil \frac{n_i + \psi_i}{d} \right\rceil ,$$

which, given that $n_i < d$ and $\psi_i \leq 1$, is at most equal to 1. Thus, $\psi_{i+1} = 1$ if and only if either $\psi_i$ or $n_i$ is strictly greater than 0. This concludes the induction and thus the proof of the theorem. □

The best bounds are derived for $\ell = \lfloor \log_d(n) \rfloor$ but the theorem holds for any $\ell \leq \lfloor \log_d(n) \rfloor$.

Unless $n$ is a power of $d$ we have that $\psi_r$ is equal to 1 at least for $r \geq \ell - 1$. Furthermore, it is likely to be equal to 1 even for lower values of $r$. The algebraic degree of $r + \ell$ SPN rounds is bounded by $n + d^{\ell-r} - \lfloor n/d^r \rfloor$. This observation has some interesting corollaries.

**Corollary 11.5.1.** *Let $\ell = \lfloor \log_d(n) \rfloor$, $n = \overline{n_\ell ... n_0}^d$ be the block size and $m = d + 1$ be the size of the S-Boxes. Assume that there exists $i < \ell - 2$ such that $n_i \neq 0$. Then the maximum number of rounds for which the degree is at most $n - 2$ is equal to*

$$\begin{cases} 2\ell & \text{if } n_\ell > 1, \\ 2\ell - 1 & \text{if } n_\ell = 1,\ n_{\ell-1} \geq 1, \\ 2\ell - 2 & \text{if } n_\ell = 1,\ n_{\ell-1} = 0,\ n_{\ell-2} \geq 1. \end{cases}$$

*The same results can be expressed using intervals rather than the expansion in base $d$ of $n$. The maximum number $r_s$ of rounds for which* $\deg(A(SA)^{r_s - 1}) \leq n - 2$ *is equal to*

$$\begin{cases} 2\ell & \text{if } 2d^\ell < n < d^{\ell+1}, \\ 2\ell - 1 & \text{if } d^\ell + d^{\ell-1} < n \leq 2d^\ell, \\ 2\ell - 2 & \text{if } d^\ell + d^{\ell-2} < n \leq d^\ell + d^{\ell-1}. \end{cases}$$

*Proof.* Using the assumptions of the corollary along with Theorem 11.5.1, we deduce that the degree is at most $n - 2$ if $\lfloor n/d^r \rfloor - d^{\ell - r} \geq 1$, which can also be written $\sum_{i=r}^{\ell-1} n_i d^{i-r} + (n_\ell - 1)d^{\ell-r} \geq 1$.

- If $r = \ell$, then we need that $n_\ell - 1 \geq 1$, which implies the first case.
- If $n_\ell = 1$ then the inequality becomes $\sum_{i=r}^{\ell-1} n_i d^{i-r} \geq 1$. For $r = \ell - 1$, it is equivalent to $n_{\ell-1} \geq 1$. For $r = \ell - 2$ and $n_{\ell-1} = 0$, it is equivalent to $n_{\ell-2} \geq 1$.

These results are easily turned into intervals. For example, $n_\ell > 1$ and $\psi_\ell = 1$ if and only if $n > 2d^\ell$. Furthermore, if $n = d^\ell$ then $r_s = 2\ell - 1$ as in this case $\psi_\ell = 0$. The other intervals are deduced identically. □

We deduce from Corollary 11.5.1 that a good rule of thumb to estimate the number of SPN rounds necessary to achieve full degree is to use $2 \times \lfloor \log_{m-1}(n) \rfloor$ rounds. Interestingly, this result is very similar to what is stated about Feistel networks in Theorem 10.5.1. Roughly speaking, in order to prevent the existence of an integral distinguisher, it is necessary that $d^{r/2} \geq n$ where $d$ is the degree of the Feistel function and $n$ is the block size.

Corollary 11.5.1 gives us a bound on the number of rounds for which the maximum algebraic degree can not be reached. It is also worth looking at round bounds for smaller degrees. Let us look for the maximum number of rounds $r_s$ such that $\deg(\mathrm{A}(\mathrm{SA})^{r_s - 1}) \leq n - (m + 1)$. It is then possible to attack $(\mathrm{SA})^{r_s}$ by fixing $d + 1 = m$ bits corresponding to an S-Box input, as in Lemma 11.3.2.

**Corollary 11.5.2.** *Let $\ell = \lfloor \log_d(n) \rfloor$, $n = \overline{n_\ell ... n_0}^d$ be the block size and $m = d + 1$ be the size of the S-Boxes. Assume that there exists $i < \ell - 3$ such that $n_i \neq 0$. Then the maximum number $r_s$ of rounds for which* $\deg(A(SA)^{r_s - 1}) \leq n - (m + 1)$ *is equal to*

$$\begin{cases} 2\ell & \text{if } n_\ell > 2 \text{ or } n_\ell = 2,\ n_{\ell-1} \geq 1, \\ 2\ell - 1 & \text{if } n_\ell = 2,\ n_{\ell-1} = 0 \text{ or } n_\ell = 1, n_{\ell-1} \geq 2 \text{ or } n_{\ell-1} = 1, n_{\ell-2} \geq 1 \\ 2\ell - 2 & \text{if } n_\ell = 1, n_{\ell-1} = 0,\ n_{\ell-2} \geq 1. \end{cases}$$

*The same results can be expressed using intervals rather than the expansion in base $d$ of $n$. The maximum number $r_s$ of rounds for which* $\deg(A(SA)^{r_s - 1}) \leq n - (m + 1)$ *is* *equal to*

$$\begin{cases} 2\ell & \text{if } 2d^{\ell}+d^{\ell-1} < n < d^{\ell+1}, \\ 2\ell-1 & \text{if } d^{\ell}+d^{\ell-1}+d^{\ell-2} < n \leq 2d^{\ell}+d^{\ell-1} \\ 2\ell-2 & \text{if } d^{\ell}+d^{\ell-2} < n \leq d^{\ell}+d^{\ell-1}+d^{\ell-2}. \end{cases}$$

*Proof.* We want $\lfloor n/d^r \rfloor - d^{\ell-r} > d$, which is equivalent to

$$\sum_{i=r}^{\ell} n_i d^{i-r} \geq d + d^{\ell-r} + 1.$$

- We cannot have $r = \ell$ as this would imply $n_\ell > d + 1$, which is impossible because $n_i < d$ for all $i$.
- In order to have $r = \ell - 1$, it is necessary and sufficient to have $dn_\ell + n_{\ell-1} > 2d$. It is the case if and only if $n_\ell \geq 3$ or $n_\ell = 2$ and $n_{\ell-1} > 0$.
- If it is not the case, then we may have $r = \ell - 2$. In order for this to happen, we need $d^2 n_\ell + dn_{\ell-1} + n_{\ell-2} > d + d^2$. It is the case if $n_\ell = 2$. Otherwise, as $n_\ell = 1$, we need either $n_{\ell-1} \geq 2$ or both $n_{\ell-1} = 1$ and $n_{\ell-2} > 0$.

This concludes the proof for the base $d$ expansions. Intervals are deduced from those in the same fashion as for Corollary 11.5.2. □

In most cases, the approach relying on fixing the input of a whole S-Box to leverage a distinguisher on $q - 1$ rounds to attack $q$ rounds leads to the best attacks. It is usually true that $r_s = r_i$, meaning that both distinguishers cover an equal number of rounds. Since the data complexity of the second approach is lower as the whole input of an S-Box is fixed instead of just 1 bit, it is a better attack.

However, there are cases where the simpler distinguisher based on a degree bound of $n - 2$ covers one more round. Using Corollary 11.5.1 and Corollary 11.5.2, we can see that the case $n_\ell = 2, n_{\ell-1} = 0$ yields such a case. Indeed, for such values, $r_i = 2\ell$ which means that $2\ell$ rounds have algebraic degree at most $n - 2$, but $r_s = 2\ell - 1 = r_i - 1$. This actually occurs with $d = 7$ and $n = 104 = \overline{206}^7$. For these values, the progression of the bound on the degree as deduced from Proposition 11.5.1 is $1 \rightarrow 7 \rightarrow 49 \rightarrow 96 \rightarrow 102 \rightarrow 103$. Since $96 = 108 - 8$, it is impossible to extend a 4-round distinguisher using the fixed-S-Box method. And yet, since $102 < 103$, a simple distinguisher on 5-round exists. Similarly, for $d = 3$ and $n = 512 = \overline{200222}^3$, we have that $r_s = 9$, and $r_i = 10$. Indeed, the last steps of the progression of the algebraic degree are $502 \rightarrow 508 \rightarrow 510 \rightarrow 511$ and 508 is too high to allow a fixed-S-Box integral distinguisher with a 4-bit S-Box.

We applied these corollaries to several S-Box size/block size combinations and obtained the results in Table 11.1. We denote $r_i$ the maximum number of rounds obtained with Corollary 11.5.1 such that $\deg(\mathsf{A}(\mathsf{SA})^{r_i}) \leq n - 2$. The actual value of $\deg(\mathsf{A}(\mathsf{SA})^{r_i})$ is also given: it can be computed either directly from Theorem 11.5.1 or by recursively applying the formula from Proposition 11.5.1. We also computed the number $r_s$ of rounds having a degree at most equal to $n - (m + 1)$ using Corollary 11.5.2. We then compute $\deg(\mathsf{A}(\mathsf{SA})^{r_s})$ and deduce the minimum dimension of an affine space $c_{\min}$ summing to zero over $\mathsf{SA}(\mathsf{SA})^{r_s}$ by rounding $\deg(\mathsf{A}(\mathsf{SA})^{r_s-1})$ up to its closest multiple of $m$.

| S-box size $m$ | Block size $n$ | $(n_\ell, n_{\ell-1}, n_{\ell-2})$ | $r_i$ | $\deg(A(SA)^{r_i})$ | $r_s$ | $c_{\min}$ |
|---|---|---|---|---|---|---|
| 4 | 16 | (1, 2, 1) | 3 | 13 | 3 | 12 |
| | 24 | (2, 2, 0) | 4 | 22 | 4 | 20 |
| | 32 | (1, 0, 1) | 4 | 29 | 4 | 24 |
| | 48 | (1, 2, 1) | 5 | 45 | 5 | 44 |
| | 64 | (2, 1, 0) | 6 | 62 | 6 | 60 |
| | 128 | (1, 1, 2) | 7 | 126 | 7 | 124 |
| | 512 | (2, 0, 0) | 10 | 510 | 9 | 504 |
| 8 | 64 | (1, 2, 1) | 3 | 61 | 3 | 56 |
| | 104 | (2, 0, 6) | 4 | 102 | 3 | 56 |
| | 128 | (2, 4, 2) | 4 | 126 | 4 | 120 |
| | 256 | (5, 1, 4) | 4 | 251 | 4 | 232 |
| | 512 | (1, 3, 3) | 5 | 508 | 5 | 488 |

Table 11.1: Theorem 11.5.1 and its Corollaries 11.5.1 and 11.5.2 for some $m, n$.

### 11.5.3 Attack Against Secret SPNs

Our attacks differ depending on whether the first layer is an affine layer or an S-Box layer. The former case is dealt with in Section 11.5.3.1 (p. 218) and the latter in Section 11.5.3.2 (p. 219). Another distinguisher based on the rank of the HDIM is described in Section 11.5.3.3 (p. 220). The limits of the ASASA construction in light of Theorem 11.5.1 are explored in Section 11.5.3.4 (p. 222). Finally, some experimental results are presented in Section 11.5.3.5 (p. 222).

#### 11.5.3.1 Decomposition Attacks on Schemes Starting with an A-layer

**Attack 11.5.1.** *The $(AS)^{2q+1}$ scheme with secret affine transformations over $\mathbb{F}_2^n$ and with secret bijective (possibly different) S-boxes of degree $m-1$ such that $3(m-1)^q+1 \le n$ can be decomposed with data and time complexity*

$$C_{(AS)^{2q+1}} \le 2^n .$$

**Recovery of the final S-layer.** In Theorem 11.5.1, we set $l = r = q$. By the condition of the attack we have either $\psi_q = 1$ and $n/d^q \ge 3$, or $\psi_q = 0$ and $n/d^q \ge 4$. In both cases we have

$$\deg(\mathrm{A(SA)}^{2q}) = D \le n-3 .$$

Therefore, an affine space of dimension $D+1$ is encrypted to ciphertexts that sum to 0. We can then duplicate the approach used in [BS01] and explained in Section 11.4.1 (p. 210) consisting of deriving $2^m$ systems of $2^m$ equations corresponding to the entries of $S_i^{-1}$. These systems are then solved to recover the last layer of S-Boxes.

**Complexity.** The complexity of peeling off the final S-layer is determined by the number of encryptions, which is upper bounded by $2^{D+m+1}$. However, this bound can be improved significantly. Consider an affine space $C$ of dimension $D' > D+1$, where $(n-D')$ variables are fixed and the others are free. Let us compute how many linearly independent equations linking the entries of $S_i^{-1}$ we can obtain using only the plaintexts from this space.

Linearly independent equations correspond to linearly independent indicator functions of the plaintext sets. For example, in 3-dimensional space the 2-dimensional sets $\{(*,*,0)\}, \{(*,*,1)\}, \{(0,*,*)\}$ and $\{(1,*,*)\}$ are linearly dependent. However, any three of them are linearly independent.

We can guarantee the linear independence as follows. Consider subspaces of $C$ of dimension $D'-1$, which are formed by fixing any variable (out of $D'$) to 1. These $D'$ subspaces are linearly independent. Within each, we can select $(D'-1)$ subspaces of dimension $(D'-2)$ in the same fashion. Therefore, a space of dimension $(D+3) \le n$ contains at least $D^2+5D$ spaces of dimension $D+1$. For all $d, m, n$ that we consider the condition $D^2+5D > 2^m$ holds as $D \approx n$ and $m \approx \log_q(n)$, so the total complexity of the first step is upper bounded by $2^{D+3} \le 2^n$.

**Recovery of the A-layers.** We are left with the subcipher $\mathrm{A(SA)}^{2q}$ which has incomplete degree $D \le n-3$. The affine layers can be recovered with the technique from [MDFK15] recalled in Section 11.4.2 (p. 211) provided that we can find enough equations.

We have already demonstrated that the full codebook contains at least $(n-3)^2 + 5(n-3) \ge n^2/2$ linearly independent affine spaces of dimension $n-2$, so the total complexity of the affine-recovery step does not exceed $2^n$. For smaller $D$ the complexity is around $2^{D+3}$.

**Attack 11.5.2.** *The $(AS)^{2q}$ scheme with secret affine transformations over $\mathbb{F}_2^n$ and secret bijective S-boxes of degree $m-1$ (possibly different) such that $2(m-1)^q+1 \le n$ can be attacked with data and time complexity*

$$C_{(AS)^{2q}} \le 2^{n-m+3}.$$

The argument in this case is similar to the previous one except that $l = q, r = q-1$. Briefly, we have a bound $\deg(AS)^{(2q-1)} \le n-(1+2(m-1)-(m-1)) = n-m$, which is smaller by $(m-3)$ than in Attack 11.5.1. This difference is deducted from the complexity exponent in $2^n$.

#### 11.5.3.2 Decomposition Attacks on Schemes Starting with an S-layer

**Attack 11.5.3.** *The $S(AS)^{2q+1}$ scheme with secret affine transformations over $\mathbb{F}_2^n$ and secret bijective (possibly different) m-bit S-boxes of degree $m-1$ such that $(m+1)(m-1)^q+1 \le n$ can be decomposed with complexity*

$$C_{S(AS)^{2q+1}} \le 2^n.$$

*Proof.* We have that $n \ge (m+1)(m-1)^q+1 = (m-1)^{q+1}+2(m-1)^q+1$. Thus, by applying Corollary 11.5.2 with $q = \ell-1$, we deduce that $\deg(A(SA)^{2q}) = D \le n-m-1$. Therefore, it is sufficient to encrypt affine spaces of dimension $n-m$. Those can be produced before the affine layer by fixing the input to a single S-box and varying the others.

The rest of the attack is identical to the decomposition of SASAS: the input of one of the S-boxes is fixed successively to each possible values while the others take all possible values at once. We deduce and solve $2^m$ linear equations. The total data complexity is about $2^n$ encryptions. □

We summarize our attacks for small $q$ and some interesting $m, n$ in Table 11.2 and give the equivalent key size for the AS pair of layers, that is, about $(n - 1.45n/m)2^m + n^2$ bits.

| $m$ | $n$ | Key size | ASASAS | SASASAS | ASASASAS | SASASASAS |
|---|---|---|---|---|---|---|
| 4 | 12 | 270 | $2^{11}$ | - | - | - |
| | 16 | 420 | $2^{11}$ | $2^{15}$ | $2^{15}$ | - |
| | 24 | 1060 | $2^{11}$ | $2^{15}$ | $2^{15}$ | $2^{24}$ |
| 6 | 12 | 728 | $2^{12}$ | - | - | - |
| | 18 | 1200 | $2^{17}$ | - | - | - |
| | 24 | 1744 | $2^{21}$ | - | - | - |
| | 36 | 3048 | $2^{28}$ | $2^{36}$ | $2^{36}$ | - |
| | 120 | $2^{14}$ | $2^{28}$ | $2^{36}$ | $2^{106}$ | $2^{114}$ |
| 8 | 128 | $2^{15}$ | $2^{52}$ | $2^{64}$ | $2^{118}$ | $2^{128}$ |
| | 256 | $2^{17}$ | $2^{52}$ | $2^{64}$ | $2^{230}$ | $2^{240}$ |

Table 11.2: Summary of the complexity of our decomposition attacks with concrete $m, n$ and different number of rounds $q$.

#### 11.5.3.3 Exploiting Lower-Degree Linear Combinations

Even if the targeted primitive has maximal degree, it can still be attacked under some circumstances. We effectively add one more affine layer to a generic SPN structure vulnerable to the decomposition attacks described above. We present a distinguisher that exposes a property that is unlikely to occur in a random permutation.

The first observations allowing our attack is the following lemma.

**Lemma 11.5.1.** *Let $f$ be an $n$-bit function and $z(f, n)$ be the number of non-zero $b$ in $\mathbb{F}_2^n$ such that* $\deg\left(x \mapsto b \cdot f(x)\right) \leq n - 2$*. Then the expected value of $z(f, n)$ is* $1 - 2^{-n} \approx 1$.

*Proof.* We consider the HDIM of $f$, as defined in Definition 8.2.6: it is a $n \times n$ binary matrix where the coefficient at line $i$ and column $j$ is equal to 1 if and only if the monomial $\prod_{k \neq j} x_k$ of degree $n - 1$ is present in the ANF of the $i$-th output bit of $f$.

We consider each of the coordinates independently and assume that a monomial appears in the ANF of a coordinate of a random permutation with probability 1/2. The expected number of solutions $b$ of the equation $M \times b = 0$ where $M$ is such a matrix is given by Theorem 3.2.4 and the preceding comments of [Kol99]: it is equal to $1 - 2^{-n}$ and thus converges to 1 as $n$ goes to infinity.

Such solutions correspond to linear combinations of the coordinates of $f$ such that the monomials of degree $n-1$ cancel each other. In other words, such $b$ are such that $\deg\left(x \mapsto b \cdot f(x)\right) \leq n - 2$ and their expected number is $1 - 2^{-n}$. □

Consider now an $n$-bit permutation $P$ with degree $n - m + 1$ and a layer $S$ of $m$-bit S-Boxes with degree $m - 1$. Then $S \circ P$ has degree $n - 1$. However, because of Lemma 11.5.1, we can expect each of the S-Boxes to have a linear combination of its coordinates with lower degree. Hence, we expect the existence of about $n/m$ linearly independent linear combinations of the coordinates of $S$ with an algebraic degree at most equal to $m-2$. They are denoted using elements $v_i$ of $\mathbb{F}_2^n$ such that $x \mapsto v_i \cdot S(x)$ has degree $m-2$ and such that the $v_i$ are linearly independent.

Using the following lemma, we will deduce that the degree of $x \mapsto v_i \cdot S(P(x))$ is at most equal to $n-2$.

**Lemma 11.5.2.** *Let $P$ be a permutation of $\mathbb{F}_2^n$ such that* $\deg(P^{-1}) \leq n-m+1$ *for $m < n$ and let $F$ be a function mapping $\mathbb{F}_2^n$ to $\mathbb{F}_2$ of degree at most $m-2$. Then the degree of the composition of $F$ and $P$ is bounded as follows:*

$$\deg(F \circ P) \leq n-2 .$$

*Proof.* The degree of $F \circ P$ is at most equal to $n-1$. Indeed, $\deg(F) \leq m-2 < n$, which means that $\sum_{x \in \mathbb{F}_2^n} F(P(x)) = \sum_{y \in \mathbb{F}_2^n} F(y) = 0$. We deduce that that the degree of $F \circ P$ is strictly smaller than $n$.

Using the Möbius transform (see Definition 8.2.3 (p. 142)), we see that the degree of $F \circ P$ is equal to $n-1$ if and only if there exists a vector space $\mathcal{U} = \{x \preccurlyeq u, \forall x \in \mathbb{F}_2^n\}$ for some $u$ with $\text{hw}(u) = n-1$ such that

$$\bigoplus_{x \in \mathcal{U}} F(P(x)) = 1 .$$

This sum can be re-written using the function $\mathbb{I}_{\mathcal{U}} : \mathbb{F}_2^n \mapsto \mathbb{F}_2$ which is such that $\mathbb{I}_{\mathcal{U}} = 1$ if and only if $x$ is in $\mathcal{U}$:

$$\bigoplus_{x \in \mathcal{U}} F(P(x)) = \bigoplus_{x \in \mathbb{F}_2^n} F(P(x)) \times \mathbb{I}_{\mathcal{U}}(x) = \bigoplus_{y \in \mathbb{F}_2^n} F(y) \times \mathbb{I}_{\mathcal{U}}\left(P^{-1}(y)\right) ,$$

where $y = P(x)$.

Since $\mathcal{U}$ is the set of all $x$ such that $x \preccurlyeq u$ where $u_i = 1$ for all $i \neq k$, its indicator function is simply $\mathbb{I}_{\mathcal{U}} : x \mapsto 1 \oplus x_k$. In particular, it has algebraic degree 1. Thus, $F \circ P$ has algebraic degree $n-1$ if and only if

$$\bigoplus_{y \in \mathbb{F}_2^n} \underbrace{F(y)}_{\deg \leq m-2} \times \underbrace{\mathbb{I}_{\mathcal{U}}\left(P^{-1}(y)\right)}_{\deg \leq n-m+1} = 1 ,$$

which is impossible because $\deg\left(y \mapsto F(y) \times \mathbb{I}_{\mathcal{U}}\left(P^{-1}(y)\right)\right) \leq n-1$, meaning that it sums to 0 over the whole space $\mathbb{F}_2^n$. We conclude that $\deg(F \circ P) \leq n-2$. □

The low degree coordinates of $F \circ P$ can be detected using the following method. For all $n$ vector spaces $C_i$ where only bit $i$ is fixed to 0, compute the sum $\bigoplus_{x \in C_i} (S \circ P)(x) = s_i$. Then, build the $n \times n$ matrix where row $i$ is equal to $s_i$, i.e. the HDIM of $S \circ P$.

If a linear combination of the output bits has algebraic degree $n-2$, then the corresponding linear combination of the rows of this matrix is equal to the all-zero row because each row is a sum over a space of size $2^{n-1}$ which is equal to 0 for the lower-degree linear combinations. Hence, the rank of this matrix will be close to $n - n/m$ while the rank of a random binary matrix is expected to be $n-1$. As the number of S-Boxes in the layer increases, the rank of this matrix decreases. Furthermore, the application of an affine layer after $S \circ P$ does not change the presence of low-degree linear combinations, it merely shuffles them. Thus, the same discrepancy in rank would be observed in $A \circ S \circ P$.

**Attack 11.5.4.** *Let $PSA$ denote a construction where $P$ is a secret permutation with* $\deg(P) = n - m + 1$, *$S$ is a secret layer of $m$-bit (possibly different) S-Boxes with degree $m - 1$ and $A$ is a secret affine transformation. This permutation can be distinguished from a random one with high probability with complexity*

$$C_{PSA} \leq n2^{n-1}.$$

#### 11.5.3.4 Why ASASA Can Not Be Secure

An ASASA structure with equal-size S-boxes cannot reach a full degree. Indeed, even if we take an ASASA instance with maximum degree, namely one with two S-Box layers where each consists of two S-Boxes of size $m = n/2$, then the decomposition of $n$ in base $d$ is simply $n = 2d + 2$. This implies that the algebraic degree is at most $n - 2$ as this puts us in the situation of Corollary 11.5.1.

**Corollary 11.5.3.** *The $n$-bit ASASA scheme with equal-size S-Boxes has algebraic degree at most $n - 2$.*

This can be seen in several of the S-Boxes listed in Table 9.7. The S-Boxes of Clefia ($S_0$), Crypton, Enocoro, and the two of Twofish use a SAS structure possibly composed with additional linear layers. All of them have algebraic degree $n - 2 = 6$.

As a result of Corollary 11.5.3, we deduce a distinguisher on the ASASA scheme for any $m$ with a complexity of $2^{n-1}$: the sum over any cube of this size must be equal to 0. For comparison, the best attack in [DDKL15, MDFK15] has complexity $2^{3n/2}$. However, it is a recovery attack while we only have a distinguisher.

#### 11.5.3.5 Experimental Verification

We have verified our attack experimentally. We considered the ASASASAS scheme with 16-bit block and four 4-bit S-boxes. The inputs to the last S-layer have degree 13, thus they sum to zero over any affine space of dimension 14.

We need $2^4$ linearly independent equations to recover the S-box. We encrypted $2^{15}$ plaintexts that start with the zero bit. Within this structure, we consider 15 substructures $\{\mathcal{S}_i\}$, where $i$-th bit is zero in $\mathcal{S}_i$. We obtained a system of 15 equations with rank 11 (in most cases). We assigned arbitrary distinct values to 5 unknowns and solved the resulting system. As a result, we got an S-box which is affine-equivalent to the original one. If we use the true values of these unknowns, the S-box is recovered precisely.

We also tried the rank based distinguisher against the ASASASASA scheme (addition of 1 secret affine layer) with the same parameters. As the degree of ASASASA is equal to 13 and $m = 4$, we expect the presence of 4 linear combinations of the output bits with algebraic degree 14 instead of 15. We ran the matrix based method to count these linear combinations. Over ten ASASASASA schemes' constructions, we found that the average number of low-degree linear combinations is 4.5 while the average of this quantity is equal to 0.7 for the same number of permutations generated with a Knuth shuffle.

## 11.6 Links with the Division Property

The similarity between our results derived using Theorem 11.5.1 and those obtained by Todo using his division property is described in Section 11.6.1 (p. 223). We show in

Section 11.6.2 (p. 223) that the division property can be reinterpreted as the algebraic degree of the indicator function of a set. The evolution of this degree is investigated in Section 11.6.3 (p. 224).

### 11.6.1 Similarity of the Results

The division property was introduced by Todo [Tod15b]. It was recalled in Definition 10.5.1.

If we set $k$ bits to take all possible values and the other to constant, we get a multiset with division property $\mathcal{D}_k^n$ or, in other words, an affine space of dimension $k$. If the multiset sums to 0 over all $n$ bits, it has division property $\mathcal{D}_2^n$. Todo found distinguishers of the form

$$\mathcal{D}_k^n \mapsto \mathcal{D}_2^n$$

for generic SPN constructions with $n$-bit block and $m$-bit S-boxes of degree $m-1$. Those are given in Table 11.3. Since these attacks outperform the existing degree bounds so far, it seems that the division property method is more effective than the algebraic one.

| n | m | r | k | Target |
|---|---|---|---|---|
| 64 | 4 | 6 | 60 | Present |
| 128 | 4 | 7 | 124 | Serpent |
| 128 | 8 | 4 | 120 | Aes |
| 256 | 4 | 8 | 252 | Minalpher |
| 512 | 4 | 10 | 509 | Prost-512 |
| 512 | 8 | 5 | 488 | Whirlpool |

Table 11.3: Todo's best integral distinguisher against SPNs.

The same results can be found using the algebraic degree bounds from Theorem 11.5.1 and Lemma 11.3.2. The idea is first to demonstrate that the algebraic degree of the $(r-1)$-round primitive is at most $n-m-1$. Therefore, the encryptions of any cube of dimension $n-m$ sum to 0 over $r-1$ rounds. We then apply the technique illustrated by Lemma 11.3.2 and fix the input of one $m$-bit S-Box while the others take all $2^{n-m}$ possible values. In this case, the sum after ciphertexts after $r$ rounds is equal to 0.

This procedure applies for all cases in Table 11.3. For instance, we have

$$2 \cdot 3^3 + 3^2 + 1 \le 64,$$

which, by Theorem 11.5.1, implies that the 5-round Present has degree 59, which is exactly what we require. The same holds for the other primitives.

### 11.6.2 Algebraic View of the Division Property

In order to demonstrate why the division property covers as many rounds as the algebraic distinguisher, we introduce an equivalent definition of the division property.

It might seem that checking the division property requires the evaluation of the entire multiset for every $u$. However, it is much easier for multisets with a compact description. The definition of the division property ignores the order of the elements of the multiset. Moreover, it is unimportant how many times an element occurs; it matters only whether this number is odd or even[2].

Let us define the *multiset indicator* Boolean function, which is true if and only if its input $y$ is present in the multiset an odd number of times:

$$\mathbb{I}_{\mathcal{X}}(y) = \bigoplus_{x \in \mathcal{X}} [x = y]$$

**Lemma 11.6.1.** *The multiset $\mathcal{X}$ has division property $\mathcal{D}_k^n$ if and only if its indicator function has degree at most $n - k$.*

*Proof.* Suppose that a multiset $\mathcal{X}$ has division property $\mathcal{D}_k^n$. Consider the dual multiset $\overline{\mathcal{X}}$

$$x \in \overline{\mathcal{X}} \Leftrightarrow \overline{x} \in \mathcal{X},$$

where $\overline{x}$ denotes the negation of $x$. The algebraic degrees of $\mathbb{I}_{\mathcal{X}}$ and $\mathbb{I}_{\overline{\mathcal{X}}}$ are the same. Now consider the ANF $\bigoplus_u a_u^{\mathbb{I}_{\overline{\mathcal{X}}}} x^u$ of $\mathbb{I}_{\overline{\mathcal{X}}}$ and some coefficient $a_u$ with $\mathrm{hw}(u) > n - k$. From the Moebius transform, we get

$$a_u^{\mathbb{I}_{\overline{\mathcal{X}}}} = \bigoplus_{y \preccurlyeq u} \mathbb{I}_{\overline{\mathcal{X}}}(y).$$

This sum is equal to the parity of the size of the intersection of $\{y \in \mathbb{F}_2^n, y \preccurlyeq u\}$ with $\overline{\mathcal{X}}$. It is thus equal to

$$a_u^{\mathbb{I}_{\overline{\mathcal{X}}}} = \bigoplus_{x \in \overline{\mathcal{X}}} [x \preccurlyeq u] \ .$$

The predicate $[x \preccurlyeq u]$ is equivalent to $[\overline{u} \preccurlyeq \overline{x}]$, so that

$$a_u^{\mathbb{I}_{\overline{\mathcal{X}}}} = \bigoplus_{x \in \mathcal{X}} [\overline{u} \preccurlyeq x] = \bigoplus_{x \in \mathcal{X}} x^{\overline{u}} \ .$$

As $\mathrm{hw}(\overline{u}) < k$, the division property of $\mathcal{X}$ implies that this last quantity is equal to 0. Thus, all coefficients of the ANF of $\mathbb{I}_{\mathcal{X}}$ corresponding to terms of degree higher than $n - k$ are equal to 0, meaning that $\deg(\mathbb{I}_{\mathcal{X}}) \leq n - k$. □

Thus, looking at the division property of a multiset boils down to studying the algebraic degree of the indicator function of the multiset. The decrease of $k$ in $\mathcal{D}_k^n$ as a multiset goes through the rounds of the cipher has therefore a natural algebraic explanation: the multiset description becomes more sophisticated and is described by a function of increasing algebraic degree.

### 11.6.3 Evolution of the Multiset Degree

Suppose now that the multiset $\mathcal{X}$ corresponding to the indicator function $\mathbb{I}_{\mathcal{X}}$ goes through an S-box $S$ of degree $d$ and becomes $\mathcal{Y} = S(\mathcal{X})$. The ANF of its indicator

[2] A similar approach was independently taken in [BC16].

function $\mathbb{I}_{\mathcal{Y}}$, expressed as $\bigoplus_v a_v^{\mathbb{I}_y} y^v$, is such that

$$\begin{aligned} a_v^{\mathbb{I}_y} &= \bigoplus_{y \leq v} \mathbb{I}_{\mathcal{Y}}(y) \\ &= \bigoplus_{y \leq v} \bigoplus_{x \in \mathbb{F}_2^n} [S(x) = y] \mathbb{I}_X(x) \\ &= \bigoplus_{x \in \mathbb{F}_2^n} [S(x) \leq v] \mathbb{I}_X(x) \; . \end{aligned}$$

The Boolean function $x \mapsto [S(x) \leq v]$ has degree $d(n - \mathrm{hw}(v))$ as $x \mapsto S(x)$ has degree $d$ and $y \mapsto [y \leq v]$ has degree $(n - \mathrm{hw}\,(v))$. Therefore, for

$$d(n - \mathrm{hw}(v)) + \deg(\mathbb{I}_X) < n,$$

the function $x \mapsto [S(x) \leq x] \mathbb{I}_X(x)$ has incomplete degree in $x$, so it sums to 0 over $\mathbb{F}_2^n$. We deduce that if $v$ is such that

$$\mathrm{hw}\,(v) > n - \left\lfloor \frac{n - \deg(\mathbb{I}_X)}{d} \right\rfloor \; ,$$

then the ANF coefficient $a_v^{\mathbb{I}_y}$ is equal to zero. As a consequence,

$$\deg(\mathbb{I}_Y) \leq n - \left\lceil \frac{n - \deg(\mathbb{I}_X)}{d} \right\rceil, \qquad (11.2)$$

which is equivalent to $\mathcal{D}_k^n$ becoming $\mathcal{D}^n_{\lceil \frac{k}{d} \rceil}$.

Equation (11.2) is the same as in Proposition 11.5.1! Therefore, the multiset degree grows at the same speed as the algebraic degree of the primitive. We conclude that the evolution of the division property is the same process as the algebraic degree growth.

Chapter 12

# Pollock Representation and TU-Decomposition

Not all S-Box structures correspond to simple mathematical objects or block cipher structures such as those attacked in the previous chapters. The methods presented in Chapters 10 and 11 may not be sufficient.

In order to both distinguish an S-Box from a random one and propose a decomposition of it, we introduce two different tools: the *Pollock representation* and the *TU-decomposition*. The first is a trick to look for visual patterns in the DDT/LAT of an S-Box. The second uses particular integral patterns to decompose S-Boxes with certain structures. These tools are related: the first step of a TU-decomposition may be the reconstruction of a visual pattern in the LAT of a function.

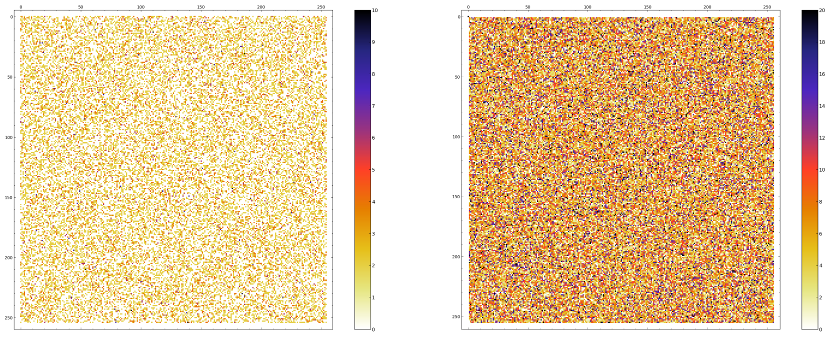
The Pollock representation is introduced in Section 12.1. In Section 12.2, I describe some of the patterns that can be expected in the Pollock representation of an S-Box depending on its structure. In particular, a connection between integral and zero-correlation attacks is used and some properties of Feistel networks are identified. The Pollock representations of the S-Boxes of Clefia, Enocoro and Zuc are studied in Section 12.2.4. I also describe an algorithm which generates an S-Box such that an arbitrary (small) picture is embedded in its DDT in Section 12.2.5. Finally, Section 12.3 presents the TU-Decomposition and shows a first simple application of its principle by decomposing the S-Box of Cmea.

The next two chapters are in fact devoted to the decomposition of two permutations for which the TU-decomposition was the first step: the S-Box of the last two GOST standards in symmetric cryptography in Chapter 13 (p. 245) and the only known APN permutation on an even number of variables in Chapter 14 (p. 267).

## 12.1 From an S-Box to a Picture and Back Again

In order to distinguish an S-Box from a random one we propose a new method which we call *Pollock's Pattern Recognition*[1]. It is based on turning the DDT and the LAT of the S-Box into a picture called its *Pollock representation* and then use the natural pattern finding power of the human eye to identify "not-random properties". For example, Figure 12.1 contains the Pollock representation of the DDT and the LAT of the "F-table" of Skipjack which was studied in Section 9.2 (p. 168).

[1] The pictures obtained in this fashion have a strong abstract feel to them, hence a name referring to the painter Jackson Pollock for this method.



(a) DDT. (b) LAT.

Figure 12.1: The two tables of the S-Box of Skipjack.

The aim of this process is to use the natural pattern finding abilities of the human eye. Furthermore, as shown in Section 12.2, several S-Box structures lead to the presence of strong patterns in the Pollock representations which allow the cryptographer to identify a hidden structure at a glance.

Unfortunately, this method did not yield any result for Skipjack. Should you find some pattern in either Figure 12.1a or 12.1b, let me know.

## 12.2 Expected Patterns in Pollock Representations

In this section, I list several of the visual patterns that are to be expected in the Pollock representation of several constructions. For each pattern, an explanation as to why it is present is provided.

### 12.2.1 Zero-Correlations and Integrals

In [BLNW12], Bogdanov *et al.* have identified some links between integral and zero-correlation distinguishers which are summarized by the following proposition.

**Proposition 12.2.1** (Proposition 1 in [BLNW12]). *Let $f : \mathbb{F}_2^m \rightarrow \mathbb{F}_2^n$ be a function. If the input and output linear masks $\alpha$ and $\beta$ are independent, then the following two statements are equivalent:*

- $\mathrm{LAT}[a,b] = 0$ *for all $a \preccurlyeq \alpha$ and $b \preccurlyeq \beta$, and*
- $x \mapsto f\Big((x\&\overline{\alpha}) \oplus a'\Big) \cdot b'$ *is balanced for all $a' \preccurlyeq \alpha$ and $b' \preccurlyeq \beta$,*

*where $\overline{x}$ is the complement of $x$ and $v = x\&m$ is a C-style notation representing the vector $v \in \mathbb{F}_2^n$ such that $v_i = \min(x_i, m_i)$.*

*The second statement means that, after fixing all bits covered by $\alpha$, all components of $f$ covered by $\beta$ are balanced (in the sense of Definition 8.1.4).*

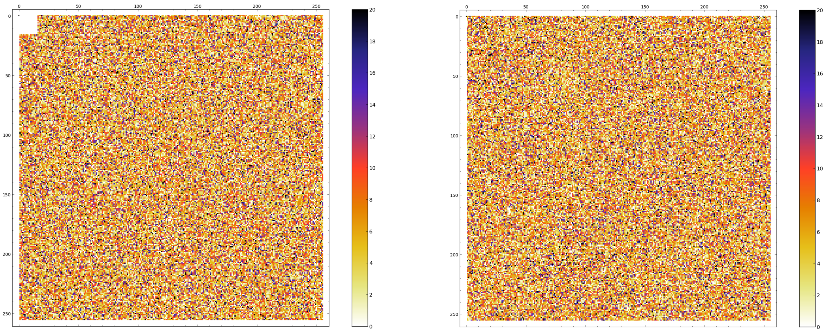
A direct consequence of this equivalence is the following lemma.

**Lemma 12.2.1** (White-square Lemma). *Let $f : \mathbb{F}_2^{2n} \rightarrow \mathbb{F}_2^{2n}$ be a function with two coordinates $f_h$ and $f_\ell$ so that $f(x||y) = f_h(x,y) \ || \ f_\ell(x,y)$, where $h$ stands for "high weight" and $\ell$ for "low weight". Then the following two statements are equivalent:*

- *$x \mapsto f_\ell(x||y)$ is a permutation of $\mathbb{F}_2^n$ for all $y$ in $\mathbb{F}_2^n$, and*
- $\text{LAT}[a,b] = 0$ *if $a < 2^n$ and $b < 2^n$.*

*In particular, if white pixels represent coefficients equal to 0 then the Pollock representation of the* LAT *of $f$ has a white-square filling the space of coordinates $(x,y)$ with $x < 2^n$ and $y < 2^n$.*

For example, Figure 12.2a contains the Pollock representation of the LAT of a function $f : \mathbb{F}_2^8 \to \mathbb{F}_2^8$ such that, using notation from [BS01], $(*,c)$ is mapped to $(?,*)$. That is, if the input nibble of high weight takes all possible values then the output nibble of low weight takes all possible values as well.



(a) $f$.

(b) $W \circ f$

Figure 12.2: The LATs of a function $f$ with a particular integral distinguisher and of $f$ composed with a word swap.

Lemma 12.2.1 implies that some integral patterns can be easily detected in the Pollock representation of an LAT. However, this pattern detection depends heavily on the representation of the field elements used when computing the index of the lines. For example, Figure 12.2b contains the Pollock representation of the LAT of the same function $f$ as in Figure 12.2a except that it has been composed with a 4-bit word swap $W : x||y \mapsto y||x$. The pattern is still present in the sense that $\text{LAT}[a||0, b||0] = 0$ but this cannot be spotted with the naked eye.

More generally, an algorithm capable of efficiently finding two non-trivial vector spaces $\mathcal{U}$ and $\mathcal{V}$ such that $\text{LAT}[u,v] = 0$ for all $(u,v)$ in $\mathcal{U} \times \mathcal{V}$ would prove extremely useful for decomposing S-Boxes as it could identify patterns more general than the white-square, which in turn would be very useful when trying to perform a TU-decomposition as described in Section 12.3.

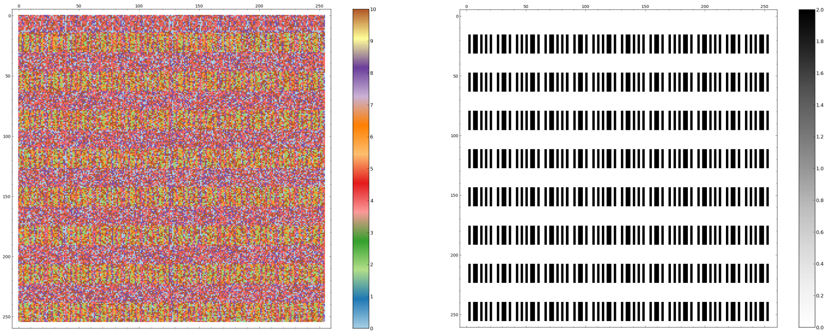
**Open Problem 12.2.1.** *Is there an algorithm which, given a list of elements $L$ of $(\mathbb{F}_2^n)^2$, can efficiently find the largest vector spaces $\mathcal{U} \subset \mathbb{F}_2^n$ and $\mathcal{V} \subset \mathbb{F}_2^n$ such that $(u,v)$ is in $L$ for all $u \in \mathcal{U}$ and $v \in \mathcal{V}$?*

### 12.2.2 The Role of the HDIM

In Section 8.2.3 (p. 142), I presented the HDIM. This matrix was introduced in a joint work with Aleksei Udovenko [PU16]. Lemma 8.2.9 showed that the congruence of the coefficients of the LAT of an $n$-bit permutation can be derived using the HDIM $\hat{H}(F)$ of this permutation as follows:

$$\mathrm{LAT}[a,b] \mod 4 \; = \; 2 \times \left( b^t \times \hat{H}(F) \times a \right) .$$

Since the function $(a,b) \mapsto \mathrm{LAT}[a,b] \mod 4$ is a bilinear form, we can expect to see a lot of regularity in the Pollock representation of the LAT modulo 4 of any permutation. Furthermore, any pattern in the HDIM will lead to the presence of patterns in this Pollock representation. A very good example of this is the S-Box of Zorro [GGNS13]. I noticed while working on [BP15] that the LAT of this S-Box had a strange striped pattern in it, as can be seen in Figure 12.3a.



(a) Without modulo.

(b) Modulo 4

Figure 12.3: The LAT of the S-Box of Zorro

After the publication of this paper, Anne Canteaut suggested looking at the LAT modulo $2^\ell$ for different powers of $\ell$ as this quantity is related to the algebraic degree of the components of the permutation. As we can see in Figure 12.3b, the LAT modulo 4 of Zorro is indeed strongly structured.[2] In hindsight, the explanation behind this pattern is very simple. The HDIM of the S-Box of Zorro is very sparse; it is equal to

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

This high number of zeroes explains the bar-code-like patterns in Figure 12.3b. Indeed, the only way for an LAT coefficient to be non-zero modulo 4 is if $a_4 \neq 0$, hence the 32 bit period of the pattern along the vertical axis. For the first 16 lines we have that $a_4 = 0$. Thus, no entry congruent to 2 modulo 4 exists. For the next 16 lines, $a_4 = 1$. However, it is then necessary that the sum $b_1 \oplus b_3 \oplus b_4 \oplus b_5$ is equal to 1. This pattern repeats itself every 32 lines and explains why its rows are all identical.

The patterns in the HDIM of Feistel networks described in Theorem 10.5.1 and in its Corollary 10.5.1 also lead to the existence of patterns in the Pollock representation of Feistel networks. These are discussed in Section 12.2.3.1

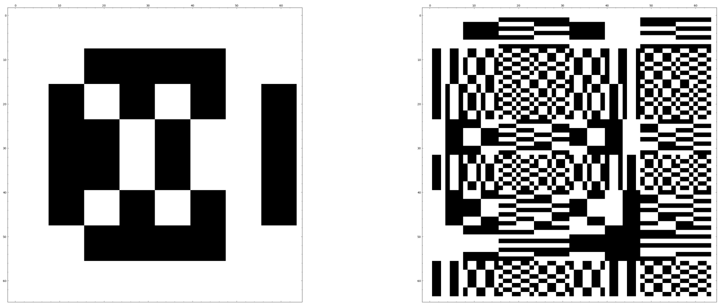
[2] This is one of the reasons which pushed Aleksei and myself to investigate the congruence modulo 4 of LAT coefficients, a study which eventually lead to [PU16].

### 12.2.3 Feistel Networks

Several attacks and distinguishers against Feistel networks have been introduced in Chapter 10 (p. 181). It turns out that their Pollock representations can also be used to identify them because of the strong patterns they contain. Those are described below.

#### 12.2.3.1 Patterns Caused by the HDIM

Figures 12.4a and 12.4b show the "Pollock representation" of the LAT modulo 4 of a 4- and a 5-round 6-bit Feistel networks for some bijective Feistel functions picked uniformly at random.



(a) $r = 4$

(b) $r = 5$

Figure 12.4: LAT of $r$-round Feistel networks (modulo 4).

As we can see, the congruence of the biases is constant in each square of dimensions $8 \times 8$ for the 4-round Feistel Networks. Furthermore, there seems to be linear patterns for the 5-round structure: if we divide the LAT into $8 \times 8$ squares as before then we find that each square at position $(i, j)$ is the sum of the squares at positions $(i, 0)$ and $(0, j)$ and a square-wise constant.

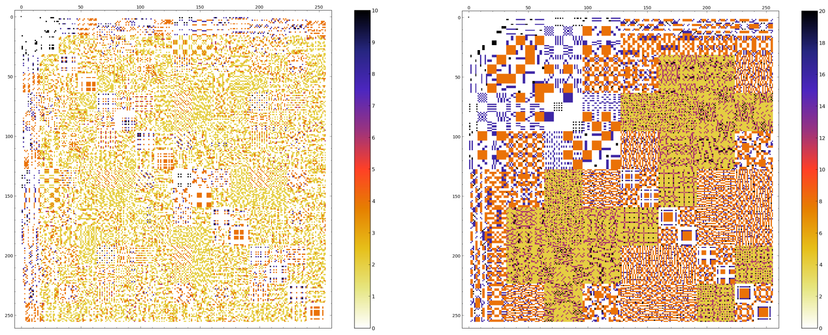
There are two reasons behind these patterns. The first reason is that the function $(a, b) \mapsto (\mathcal{L}[a, b] \mod 4)$ for the LAT $\mathcal{L}$ of a permutation is a bilinear form with the HDIM of the function as its matrix. The second aspect of the justification for the patterns is the probability 1 presence of zeroes in the HDIM of Feistel networks described in Theorem 10.5.1 and in its Corollary 10.5.1. Indeed, the HDIM of these permutations are:

$$\hat{H}(\mathsf{F}^4) = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 \end{bmatrix}, \hat{H}(\mathsf{F}^5) = \begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix},$$

where $\mathsf{F}^r$ denotes the $r$ round Feistel network and where the zeroes caused by Theorem 10.5.1 and in its Corollary 10.5.1 are represented in gray.

#### 12.2.3.2 Patterns in the LAT of a 3-Round Feistel Network

The LAT and DDT of 3-round Feistel networks have patterns that cannot be explained by their HDIM. For example, Figure 12.5 contains the Pollock representation of both the LAT[3] and the DDT of the S-Box $S$ of the block cipher iScream [GLS+14].



(a) DDT of $S$.

(b) LAT of $W \circ S \circ W$.

Figure 12.5: The two tables of the S-Box $S$ of iScream.

Both tables are symmetric along their top-left to bottom-right diagonal because $S$ is an involution. We also note that both share a white-square of $16 \times 16$ pixels in the top left corner. For the DDT, this is caused by the impossibility of the difference $(\delta, 0) \rightsquigarrow (\delta', 0)$ for all non-zero pairs $(\delta, \delta')$. Indeed, if the input difference is $(\delta, 0)$ then the right output difference after 3 rounds is the image of $\delta$ by the middle Feistel function. Since it is a permutation, it cannot be 0. For the LAT, it is caused by the presence of an integral distinguisher (see Section 12.2.1).

The chessboard-like pattern whereby the DDT and LAT seems divided into $2^n$ squares of side $2^{n/2}$ (where $n$ is the block size) has always been present for all the 3-round Feistel networks I generated at random. It is stronger when all Feistel functions are permutations but is mostly scrambled if the second Feistel function has inner-collisions.

The cause of this pattern remains unclear. A 3-round Feistel network may not have algebraic degree $n-1$, as is the case for iScream. This means that all coefficients in its HDIM are equal to 0, which implies that this structure is not the cause of this pattern.

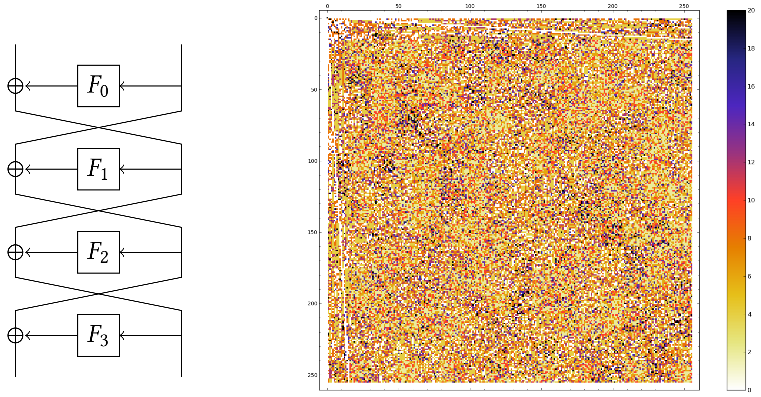
It is noteworthy that open butterflies, a structure introduced in Chapter 14, exhibit the same type of chessboard pattern. In particular, the APN permutation of Dillon *et al.* does if it is composed with an appropriate linear permutation, as shown in Figure 14.1b (p. 268). Since 3-round Feistel networks with bijective second Feistel functions are particular instances of the butterfly structure, it is likely that an explanation can be found at this level.

**Open Problem 12.2.2.** *What is the cause of the checkered pattern occurring in both the DDT and the LAT of a 3-round Feistel network (after composition with branch swaps)?*

#### 12.2.3.3 Patterns in the LAT of a 4-Round Feistel Network

Let $F_0, \ldots, F_3$ be four $n$-bit Boolean permutations. Figure 12.6a represents the 4-round Feistel network $f$ built using $F_i$ as its $i$-th Feistel function. Figure 12.6b is the Pollock representation of the LAT of an 8-bit Feistel network $f_{\text{exp}}$ built using four 4-bit permutations picked uniformly at random.

[3] The LAT is actually that of $S$ composed with two word swaps denoted $W$.



(a) Definition of $f$.

(b) The LAT (Pollock repr.) of $f_{\text{exp}}$.

Figure 12.6: A 4-round Feistel network and its LAT.

In Figure 12.6b, we see that the LAT $\mathcal{L}_{\text{exp}}$ of $f_{\text{exp}}$ contains both vertical and horizontal segments of length 16 which are made only of zeroes. These segments form two lines starting at $(0,0)$, one ending in $(15,255)$ and another one ending in $(255,15)$, where $(0,0)$ is the top left corner. The vertical segments are in columns 0 to 15 and correspond to entries $\mathcal{L}_{\text{exp}}[a_L||a_R, 0||a_L]$ for any $(a_L, a_R)$. The horizontal ones are in lines 0 to 15 and correspond to entries $\mathcal{L}_{\text{exp}}[0||a_R, a_R||a_L]$ for any $(a_L, a_R)$.

The coefficients which correspond to such vertical segments for any 4-round Feistel network $f$ with LAT $\mathcal{L}$ are such that

$$\begin{aligned} 2\mathcal{L}[a_L||a_R, 0||a_L] &= \sum_{x \in \mathbb{F}_2^{2n}} (-1)^{(a_L||a_R)\cdot x \oplus a_L \cdot f(x)} \\ &= \sum_{r \in \mathbb{F}_2^n} (-1)^{a_R \cdot r} \sum_{\ell \in \mathbb{F}_2^n} (-1)^{a_L \cdot \left(\ell \oplus f_R(\ell || r)\right)}, \end{aligned}$$

where $f_R(x)$ is the right word of $f(x)$. This quantity is equal to $\ell \oplus F_0(r) \oplus F_2\Big(r \oplus F_1(\ell \oplus F_0(r))\Big)$, so that $\mathcal{L}[a_L||a_R, 0||a_L]$ can be re-written as:

$$2\mathcal{L}[a_L||a_R, 0||a_L] = \sum_{r \in \mathbb{F}_2^n} (-1)^{a_R \cdot r} \sum_{\ell \in \mathbb{F}_2^n} (-1)^{a_L \cdot \left(F_0(r) \oplus F_2\left(r \oplus F_1(\ell \oplus F_0(r))\right)\right)}.$$

Since $\ell \mapsto F_2\Big(r \oplus F_1(\ell \oplus F_0(r))\Big)$ is balanced for all $r$, the sum over $\ell$ is equal to 0 for all $r$ (unless $a_L = 0$). This explains[4] the existence of the vertical "white segments". The existence of the horizontal ones is a simple consequence of the relationship between the LAT of a permutation and that of its inverse. As the inverse of $f$ is also a 4-round Feistel, its LAT must contain white vertical segments and since the LAT of $f$ is the transpose of the LAT of $f^{-1}$, these vertical white segments become the horizontal ones. We summarize this observation with the following lemma.

[4] In fact, our proof only requires that $F_1$ and $F_2$ are permutations. The pattern would still be present if the first and/or last Feistel functions had inner-collisions.

**Lemma 12.2.2** (LAT lines for 4-round FN)**.** *For a 4-round Feistel network operating on 2n bits with bijective second and third Feistel functions, the* LAT *is such that for all $a_L$ and $a_R$ in $\mathbb{F}_2^n$:*

$$\text{LAT}[a_L||a_R, 0||a_L] = \text{LAT}[0||a_L, a_L||a_L] = 0 .$$

### 12.2.4 Case Studies: Some Known S-Boxes

#### 12.2.4.1 An S-Box of Clefia

The block cipher Clefia [SSA+07] uses two different S-Boxes: $S_0$ and $S_1$. While $S_1$ is simply based on the inverse function, $S_0$ has a SAS structure: it uses two layers of 4-bit S-Boxes interleaved with a linear mixing layer based on the following matrix mapping $(\mathbb{F}_{2^4})^2$ to itself:

$$M = \begin{bmatrix} 1 & 2 \\ 2 & 1 \end{bmatrix} .$$

An integral pattern $(*, c)$ is mapped to $(*, *)$ by M; and so is $(c, *)$. This explains why the Pollock representation of its LAT has a white-square and horizontal "dents" on its left-most side which correspond to another white-square after its row indices have had their left and right words swapped. Since its inverse yields the same integral patterns, vertical "dents" are present at its top.

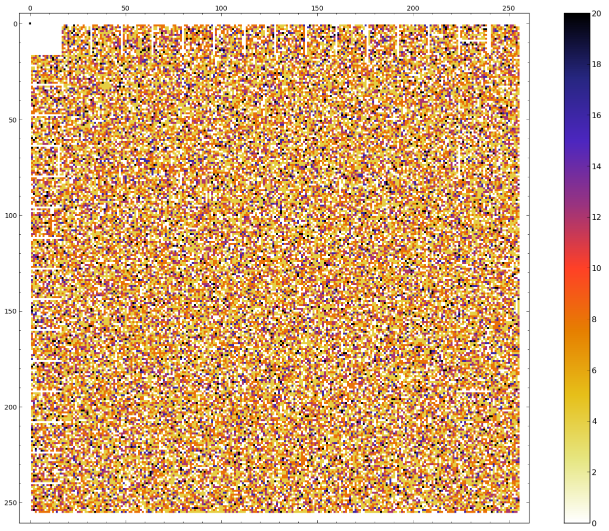


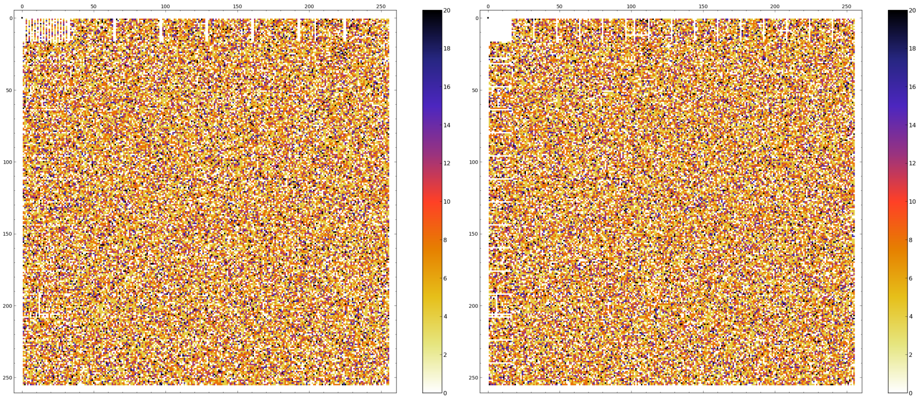
Figure 12.7: The LAT of the S-Box $S_0$ of Clefia.

With some practice, we can also notice that this Pollock representation has too few colors. This is due to the algebraic degree of the S-Box being equal[5] to 6 = 8-2, which implies that all LAT coefficients must be divisible by 4. Therefore, all colors corresponding to 2, 6, 10, etc. are absent in this case.

[5] In Corollary 11.5.3 (in Chapter 11), we have showed that this degree is in fact the best that can be achieved by such a Substitution-Permutation Network structure.

#### 12.2.4.2 The S-Box of Enocoro

The S-Box used by the stream cipher Enocoro, denoted $S_e$, has a structure similar to that of Clefia, except that a rotation by 1 bit to the left is applied after the SAS construction. It is sufficient to partially scramble the patterns present in its lat, as shown in Figure 12.8a.



(a) The lat of $S_e$, $\mathcal{L}$.

(b) $\mathcal{L}'$ where $\mathcal{L}'[a,b] = \mathcal{L}[a,(b \lll 1)]$.

Figure 12.8: The Pollock representation of $S_e$.

However, it is easy to recover this final rotation from Figure 12.8a. Indeed, we note that the top-left corner contains a specific pattern: instead of a $16 \times 16$ white-square, it contains a $16 \times 32$ rectangle where every second column is white. These columns are easily re-aligned to the left by applying a 1-bit rotation to the right on the column indices. The result is shown in Figure 12.8b.

By Lemma 8.2.4, the S-Box with the lat in Figure 12.8b is equal to $x \mapsto (S_e(x) \ggg 1)$. because transposing the binary matrix representing a rotation yields the representation of its inverse. Since $S_e$ consists of a SAS structure followed by a rotation to the left, composing it with a rotation to the right unsurprisingly yields a result very similar to the S-Box of Clefia.

As we can see, the Pollock representation is a convenient method for identifying the last linear layer of an S-Box if it consists of a simple rotation. This idea is further illustrated below by the S-Box of Zuc.

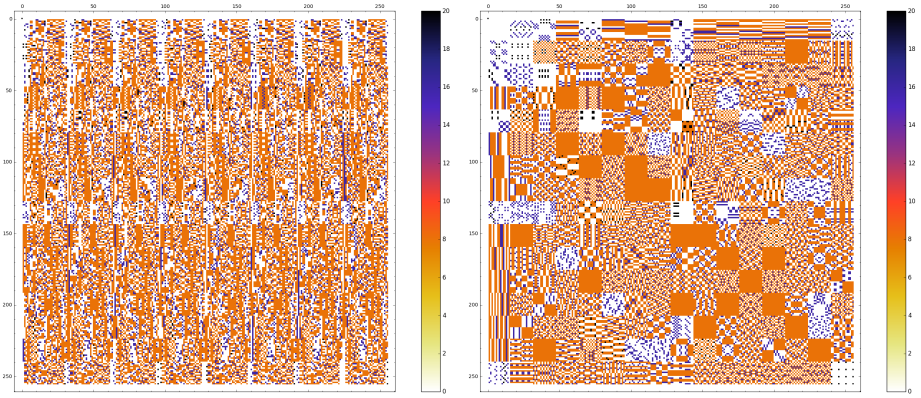
#### 12.2.4.3 The S-Box of Zuc

Zuc [ETS11] is a Chinese stream cipher which uses two different S-Boxes, $s_0$ and $s_1$. The latter is based on a finite field inversion, much like the one of the AES. However, $s_0$ has a structure which, while public, is easy to recover.[6]

The Pollock representation of its lat, denoted $\mathcal{L}$, is given in Figure 12.9a. It is obvious from it that the S-Box is structured: there are too many zeroes, too few colors

[6] In fact, the structure is not described in all the specifications of ZUC. The one Alex Biryukov, Aleksei Udovenko and I looked at first did not contain it. We thus decomposed it using the strategy outlined in this section. However, later, Gaëtan Leurent — who I thank for this — pointed out that this structure was released in another document.

(which indicates a low algebraic degree) and some form of period of length 32 along the horizontal axis.



(a) The LAT of $s_0$, $\mathcal{L}$.

(b) $\mathcal{L}'$ where $\mathcal{L}'[a,b] = \mathcal{L}[a,(b \lll 5)]$.

Figure 12.9: The Pollock representation of $s_0$.

It is thus natural to rotate the column indices by 5 bits to the right so as to cluster similar columns together. The result is given in Figure 12.9b and is typical of a 3-round Feistel network, as discussed above in Section 12.2.3.2 (p. 231).

As with Enocoro, the explanation is simple. Indeed, this S-Box was built by composing a 3-round Feistel network with a rotation by 5 bits to the left, as illustrated in Figure 10.1 in Section 10.3.1.1 of [ETS11]. As for Enocoro, the Pollock representation allows a cryptanalyst to easily peel off a linear layer.

### 12.2.5 Seurat's Steganography

In this section, we present an algorithm allowing the creation of a non-bijective S-Box such that the picture representation of its DDT contains a particular image. Since we draw this image dot after dot like in *pointillism* and since it hides said image, we call the method we present below *Seurat's Steganography*. The pictures we embed are black and white, the white parts corresponding to places where differentials are impossible and black parts to places where the differentials have non-zero probability.

#### 12.2.5.1 The Algorithm

We define white and black equations as those giving the corresponding pixel color in the Pollock representation of the DDT of an S-Box:

- white Equations: $\mathsf{W}_{a,b} : \forall x \in \{0,1\}^m,\ S(x+a) + S(x) \neq b$, and
- black Equations: $\mathsf{B}_{a,b} : \exists x \in \{0,1\}^m,\ S(x+a) + S(x) = b$.

The inputs considered in this Section are:

- B: the complete list of the black equations,

- $T_w$ : a table of Boolean variables of size $u \times v$ (the dimensions of the image) where $T_w[a,b]$ is false if and only if the pixel at $(a,b)$ cannot be white,
- $i$: the index of the equation in B for which we need to find a solution, and
- $S$: a partially unspecified S-Box such that all equations $\mathsf{B}_j$ for $j < i$ hold and such that none of the $W_j$ has a solution for any $j$.

We first need a sub-routine checking if adding an entry $S(x) = y$ to a partially assigned S-Box, i.e. an S-Box for which some of the outputs are unspecified, leads to at least one of the white equations no longer holding. It is described in Algorithm 12.1.

**Algorithm 12.1** `checkW`$(S,x,y,T_w)$.
**Inputs:** S-Box $S$, elements $x,y$, table of Boolean variables $T_w$;
**Outputs:** `true` or `false`

```
for all $a \in \{0,1\}^m$, if $S(x+a)$ is specified, do
    if $T_w[a, S(x+a) + S(x)]$ is false then return false
end for
return true
```

We now describe Seurat's Steganoraphy, namely Algorithm 12.2, which uses two lists of equations to iteratively build an S-Box such that a particular picture appears in its DDT. It works by first making a list $L$ of all the ways entries could be added to the S-Box in order to satisfy the black equation $\mathsf{B}_i$. If none are found, the function fails. The function is finally called recursively on the candidates found to look for a solution for the next equation. If no solution is found for the next equation, the function fails.

Some optimizations are possible. First of all, it is not necessary to write this algorithm using recursion. It is also not necessary to let $L$ be as large as possible. In fact $|L| \leq 2$ is sufficient, although $|L| = 1$ does not work unless the picture is very simple. It is also possible to allow some noise by tweaking CheckW$(S,x,y,T_w)$ to return `true` with low probability for pairs $(x,y)$ even if they blacken a white pixel.

Several outputs of this algorithm are presented in Section 12.2.5.3 (p. 239).

#### 12.2.5.2 Counting Possible S-Boxes

Let $S$ be a random function from $\{0,1\}^m$ to $\{0,1\}^n$. Then $\mathsf{W}_{a,b}$ holds if and only if DDT$[a,b] = 0$, which happens with probability $\Pr[\text{DDT}[a,b]=0] = \exp\left(-2^{m-n-1}\right)$ because the coefficients in the DDT of a random function follow approximately a Poisson distribution with parameter 1/2 (see Theorem 9.1.1). Hence, if we have $b$ black equations, $w$ white ones and if we consider that their having solutions are independent events, then the probability that an S-Box has the correct image at the center of its DDT is $P_{\text{success}} = \left(\exp(-2^{m-n-1})\right)^w \times \left(1 - \exp(-2^{m-n-1})\right)^b$. In the case where $m = n$, we use that $\log_2(\exp(-1/2)) \approx -1.35$ and that $\log_2(1 - \exp(-1/2)) \approx -0.72$ to approximate this probability by

$$P_{\text{success}} = 2^{-(0.72 \cdot w + 1.35 \cdot b)}.$$

As there are $2^{n2^n}$ possible $n \times n$ S-Boxes, we expect to have very roughly the following number of solutions:

$$N_{\text{Solutions}} = 2^{n2^n - (0.72 \cdot w + 1.35 \cdot b)}.$$

**Algorithm 12.2** `Seurat`$(S, B, T_w, i)$: Seurat's steganography.
**Inputs:** S-Box $S$, black equations $B$, table of Boolean $T_w$, counter $i$
**Output:** S-Box $S'$ satisfying all black equations or `FAIL`

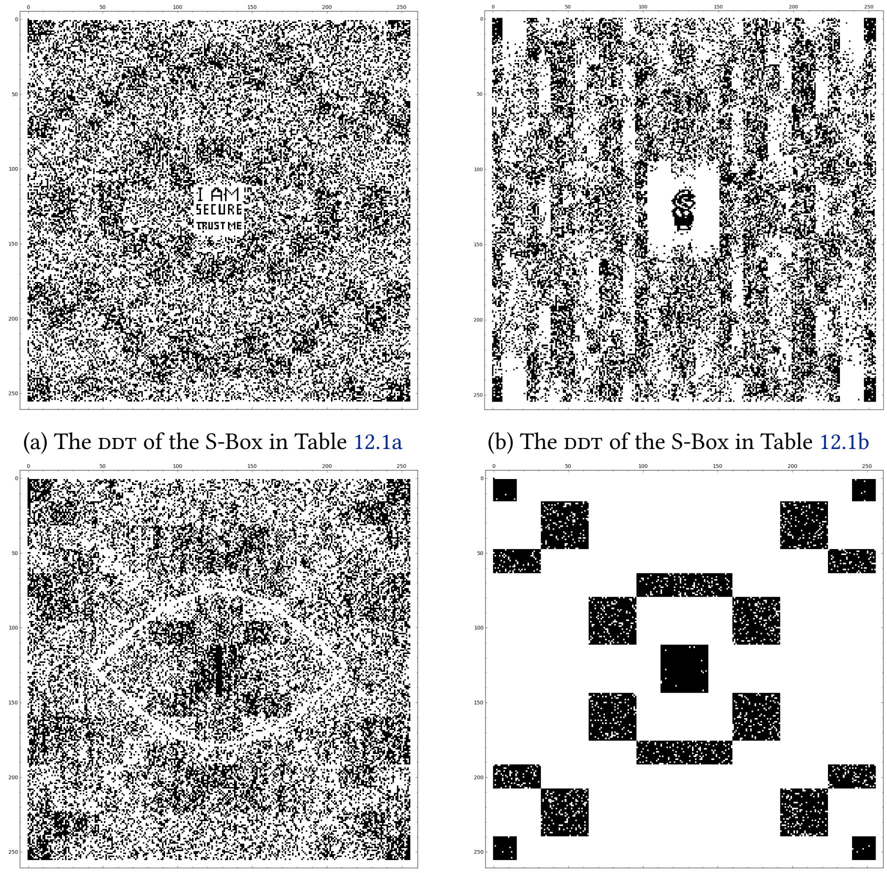
```
$\delta_{\text{in}}$ := input difference in $\mathsf{B}_i$
$\delta_{\text{out}}$ := output difference in $\mathsf{B}_i$
$L$ := empty list of S-Boxes
if $\mathsf{B}_i$ is already satisfied by $S$ then
    Append $S$ to $L$ and return $L$
end if
for all $x \in \{0,1\}^m$ do
    if $S(x)$ is not defined then
        if $S(x + \delta_{\text{in}})$ is defined then
            $y = S(x + \delta_{\text{in}}) + \delta_{\text{out}}$
            if CheckW$(S, x, y, T_w)$ then
                $S' = S$ ; $S'(x) = y$
                Append $S'$ to $L$
            end if
        else if $S(x + \delta_{\text{in}})$ is not defined then
            for all $y \in \{0,1\}^n$ do
                if CheckW$(S, x, y, T_w)$ and CheckW$(S, x + \delta_{\text{in}}, y + \delta_{\text{out}}, T_w)$ then
                    $S' = S$ ; $S'(x) = y$ ; $S'(x + \delta_{\text{in}}) = y + \delta_{\text{out}}$
                    Append $S'$ to $L$
                end if
            end for
        end if
    end if
end for
If $L$ is still empty then return FAIL
for all $S' \in L$ do
    If Seurat$(S', B, T_w, i + 1)$ does not fail then return $S'$
end for
return FAIL
```

Therefore, we need $0.72 \cdot w + 1.35 \cdot b < n2^n$ in order to have a non-empty set of S-Boxes with the image we want inside their DDT. Black pixels are about twice as expensive as white ones according to this model. However, in practice, it is only possible to build an S-Box such that its DDT contains a black square of size 22 × 22 or a white one of size 62 × 62 without any noise, meaning that black pixels are, from the point of view of our algorithm, about 8 times more expensive. Stirling's equation gives an approximate number of $2^{(n-1.44)\cdot 2^n}$ permutations of $\{0,1\}^n$, so we need that $0.72 \cdot w + 1.35 \cdot b < (n-1.44)2^n$ for permutations with the correct black/white pixels to exist with non negligible probability. However, our algorithm will require significant changes in order to search for permutations.

Since our algorithm does not require the pixels to be organized inside a square, we can also use it to force white or black pixels to appear anywhere in the DDT of an S-Box. This could be used to place a sort of trapdoor by for instance ensuring that a truncated differential compatible with the general structure of a cipher is present. Another possible use could be to "sign" an S-Box: Alice would agree with Bob to generate an S-Box for him and tell him beforehand where some black/white pixels will be. Bob can then check that they are placed as agreed.

#### 12.2.5.3 Some Results

The S-Boxes described in Tables 12.1a, 12.1b, 12.1c and 12.1d were built using the method described in Section 12.2.5. Note that these are not bijections. The picture representations of their DDT are given in Figures 12.10a, 12.10b, 12.10c and 12.10d respectively. Those clearly show the pictures we chose to embed in them. Unfortunately, the differential and linear properties of these S-Boxes are less than impressive.



(a) The DDT of the S-Box in Table 12.1a

(b) The DDT of the S-Box in Table 12.1b

(c) The DDT of the S-Box in Table 12.1c

(d) The DDT of the S-Box in Table 12.1d

Figure 12.10: The DDT of some outputs of Seurat's Steganography: $|d_{i,j}| = 0$ is in white, $|d_{i,j}| \geq 2$ is in black.

| | .0 | .1 | .2 | .3 | .4 | .5 | .6 | .7 | .8 | .9 | .a | .b | .c | .d | .e | .f |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0. | 6c | b7 | 4b | 27 | 72 | 72 | 5e | 5e | 41 | 6c | 38 | 0b | a1 | 22 | a3 | 5a |
| 1. | 70 | 57 | 20 | df | d7 | dc | b2 | 28 | da | 28 | f6 | f2 | db | 23 | ca | c5 |
| 2. | 96 | 38 | 3a | c2 | 35 | 8b | 3b | e4 | c7 | 59 | 14 | 4e | 34 | c5 | c6 | ca |
| 3. | 6c | c2 | b6 | 43 | 65 | 6c | f2 | 26 | e1 | c2 | 0a | 46 | 94 | 8d | 9b | 47 |
| 4. | aa | aa | f6 | 5d | 61 | 09 | a2 | 92 | ae | 93 | 0c | ae | f2 | 38 | 35 | 08 |
| 5. | 46 | 07 | 16 | 0a | 22 | d4 | 07 | f6 | 24 | fa | 06 | 08 | 0f | d0 | 22 | 26 |
| 6. | cc | 5b | 33 | 36 | ea | 7b | 84 | 7f | b6 | c5 | c9 | 34 | f6 | 72 | 19 | c0 |
| 7. | d2 | 9a | 69 | 96 | b4 | 40 | 45 | 6e | b4 | 9e | 6a | 6a | 8f | 3a | a4 | 51 |
| 8. | b6 | ae | be | 90 | 9a | 9a | 90 | 92 | 9c | 97 | b0 | ea | f0 | 67 | 9d | 54 |
| 9. | 89 | 8d | 78 | 83 | 32 | e4 | be | c9 | db | 28 | e4 | 2c | cc | cd | c1 | ec |
| a. | df | d4 | 2c | d1 | 0b | d3 | 6d | f6 | df | f4 | 3d | da | 0c | fd | 09 | fd |
| b. | 9c | f0 | c0 | c5 | 6c | 51 | 74 | 0f | 92 | 53 | 8c | 53 | 17 | 53 | a4 | a8 |
| c. | 9b | cc | ca | 61 | 6c | 0b | 5b | 00 | c6 | 06 | 20 | ae | b6 | 9b | 97 | 0f |
| d. | ca | ca | cb | 4e | 0b | bd | ba | d1 | 33 | 31 | 61 | 34 | c8 | d2 | 6c | f9 |
| e. | 3a | 20 | cf | 28 | 8d | 8f | 98 | 81 | d2 | 25 | d7 | 29 | 68 | dd | 8e | 0e |
| f. | 87 | 81 | 53 | 73 | 48 | 7c | 43 | b6 | ab | 95 | 5f | 1c | a5 | ba | f4 | 71 |

(a) "I am secure trust me".

| | .0 | .1 | .2 | .3 | .4 | .5 | .6 | .7 | .8 | .9 | .a | .b | .c | .d | .e | .f |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0. | 1b | 1e | e7 | 1b | 00 | 1b | 4f | e7 | 07 | a8 | b7 | 1c | 00 | 06 | 1c | 1c |
| 1. | 30 | a9 | ab | af | 54 | 50 | 36 | 57 | 65 | 01 | 17 | 7c | 53 | 99 | fb | 65 |
| 2. | 86 | b5 | 33 | 78 | c9 | 80 | f5 | 7f | 79 | 7d | 87 | 7a | 4d | 14 | 49 | 2b |
| 3. | 66 | d5 | c8 | 54 | a9 | 57 | 54 | ab | aa | 98 | a8 | a8 | 32 | 17 | d2 | cb |
| 4. | d4 | e7 | 73 | 1b | 51 | b3 | af | 50 | 51 | 68 | ac | 6b | d7 | 52 | 1b | d5 |
| 5. | 71 | 75 | 8a | 97 | c8 | 36 | 37 | 33 | 74 | ce | 75 | 4a | 77 | 88 | 8f | 77 |
| 6. | 1b | ff | e4 | b5 | ff | 1f | 1e | fa | b3 | 4a | b1 | 4c | fd | fc | 4b | 01 |
| 7. | ca | c8 | a0 | 5b | 5e | a1 | 5b | a6 | 9d | c8 | 98 | 84 | cb | 31 | ca | cb |
| 8. | 33 | ca | 33 | cc | 7b | 83 | 98 | cb | a2 | 7f | a3 | ce | 34 | 33 | cb | cd |
| 9. | e7 | fd | ff | 03 | 7f | 2d | 00 | b5 | 05 | e5 | ff | 02 | 03 | 06 | fc | 06 |
| a. | 88 | 8e | 74 | 8b | 8c | 8e | 8c | 51 | c9 | 03 | 88 | c9 | 8a | c9 | 70 | fc |
| b. | 94 | 2b | d4 | 29 | ae | 69 | 6b | af | b7 | 91 | b7 | b7 | 8b | 89 | d4 | 75 |
| c. | d1 | c9 | 98 | 99 | 61 | ab | aa | 61 | 99 | 66 | 12 | 65 | 15 | 2d | 2d | 33 |
| d. | b3 | b3 | 7c | 86 | 83 | 7a | 7f | 78 | cf | 98 | 81 | 30 | 7e | cf | c9 | c9 |
| e. | 01 | a9 | 57 | ad | e3 | 80 | ad | 61 | 56 | 53 | 53 | 28 | 56 | a8 | c8 | ae |
| f. | 18 | 1d | 00 | 06 | df | 52 | 52 | af | 1d | 61 | e2 | 60 | e2 | e6 | fa | e2 |

(b) "It's me, Mario!".

| | .0 | .1 | .2 | .3 | .4 | .5 | .6 | .7 | .8 | .9 | .a | .b | .c | .d | .e | .f |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0. | bb | 44 | 93 | 4a | a6 | 2f | af | ac | 77 | 66 | 55 | 47 | 8a | b0 | 21 | 18 |
| 1. | 96 | 99 | a3 | a7 | 5b | 3e | b3 | 5a | 87 | 66 | a6 | ab | ad | 09 | 71 | 04 |
| 2. | 2a | 20 | de | df | 21 | df | 21 | 21 | 20 | 21 | 20 | 5b | c5 | 7f | df | de |
| 3. | 28 | df | ce | f7 | 6f | 8d | 84 | 20 | 4d | a5 | 49 | 8b | 55 | ff | 47 | 32 |
| 4. | 7d | 52 | 28 | a0 | 5d | 45 | 8d | 7e | 4f | bf | 5c | 46 | 2a | 51 | a6 | a5 |
| 5. | a0 | a0 | a0 | a1 | a1 | a1 | a1 | a1 | a0 | cf | 5f | a1 | a0 | a0 | 98 | ab |
| 6. | c1 | 3d | b8 | 3b | 70 | 33 | 30 | 5f | 7c | c7 | db | c8 | 09 | 06 | 83 | c7 |
| 7. | f9 | 01 | f7 | 52 | 51 | d4 | 57 | e2 | 52 | 57 | f4 | b4 | f5 | 37 | 30 | 07 |
| 8. | 33 | 37 | c1 | 09 | c2 | 12 | 2b | 56 | 18 | 9a | 51 | ad | 5e | 08 | 95 | f6 |
| 9. | 40 | cb | 08 | 0b | b7 | 45 | 23 | 80 | 44 | 91 | 7d | 70 | 74 | e2 | 5e | 36 |
| a. | 5f | 5e | 5a | 5f | a6 | 58 | 5e | 5e | 53 | 5f | 51 | 5e | 5f | ab | ab | b9 |
| b. | 85 | ab | b2 | 86 | b2 | b3 | 48 | 45 | 49 | 8d | 23 | b9 | b7 | b5 | 8e | 2a |
| c. | c2 | 33 | 3f | 02 | 35 | fe | c8 | cc | 86 | 86 | fa | c9 | cd | 39 | c3 | 06 |
| d. | d8 | de | 7b | 20 | d7 | d5 | b4 | 26 | 44 | d0 | 21 | d2 | d0 | d7 | 23 | af |
| e. | 07 | f8 | 50 | f4 | 41 | ab | 7a | aa | 5e | a7 | c3 | 5e | a2 | 5c | 8b | 55 |
| f. | a6 | 51 | 4d | 42 | e8 | dc | 46 | 67 | 17 | 90 | ab | 51 | b4 | bd | 45 | a4 |

(c) The eye of Sauron.

| | .0 | .1 | .2 | .3 | .4 | .5 | .6 | .7 | .8 | .9 | .a | .b | .c | .d | .e | .f |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0. | bf | bd | b3 | 4c | 45 | 48 | b7 | bf | b3 | 45 | b8 | 48 | 45 | 4a | 4d | 45 |
| 1. | 9b | 9a | 62 | 96 | 60 | 91 | 91 | 93 | 60 | 62 | 6b | 65 | 97 | 9f | 9e | 61 |
| 2. | 80 | 76 | 7e | 82 | 7b | 84 | 81 | 87 | 81 | 77 | 85 | 74 | 73 | 82 | 70 | 76 |
| 3. | be | 43 | bc | b1 | 4f | be | 43 | bf | b0 | bc | bf | bb | bd | 4e | 49 | b3 |
| 4. | c9 | c6 | c4 | 33 | 39 | 3f | 37 | 33 | 3d | 32 | c6 | cd | 3a | 36 | c8 | 32 |
| 5. | 02 | 0a | 06 | f1 | fa | 02 | ff | 01 | 08 | f2 | fb | 08 | fe | 0e | fa | 02 |
| 6. | 14 | 1e | e7 | e2 | 16 | e7 | e1 | e7 | 1b | 1b | ed | eb | e6 | e7 | 15 | e9 |
| 7. | 37 | cc | c8 | c1 | 35 | 3f | 3a | c9 | cc | 39 | c5 | 38 | c4 | 35 | c9 | c7 |
| 8. | cb | 33 | cd | c2 | 32 | c8 | 32 | 3e | 33 | 3e | 33 | 30 | 3d | 34 | 30 | c8 |
| 9. | e7 | e8 | e6 | 11 | e4 | 15 | 13 | 18 | 1c | 14 | 1e | 13 | 1d | e2 | e8 | ed |
| a. | 02 | fe | f6 | 0b | 06 | 00 | f6 | fa | fd | 08 | 0f | 09 | f1 | f9 | 05 | ff |
| b. | 39 | 36 | c4 | 3b | cc | c5 | ce | cb | 3d | c4 | 32 | 39 | 3a | c4 | 30 | c8 |
| c. | bd | b3 | be | b1 | 40 | 45 | 41 | 4a | 45 | b1 | b6 | ba | b2 | 48 | ba | bf |
| d. | 7f | 83 | 79 | 77 | 82 | 7f | 85 | 7c | 7e | 88 | 79 | 8f | 71 | 89 | 7f | 85 |
| e. | 61 | 6f | 6b | 68 | 91 | 9e | 9f | 63 | 67 | 9b | 6e | 6c | 63 | 67 | 90 | 90 |
| f. | bf | 46 | b0 | b7 | 4a | bf | 4a | bd | b1 | bb | ba | 47 | 48 | b6 | 41 | ba |

(d) Forcing a central black square.

Table 12.1: Some outputs of the algorithm described in Section 12.2.5.

## 12.3 The TU-Decomposition

### 12.3.1 Principle

The *TU-decomposition* is a method allowing the decomposition of a function $f$ with a particular structure. It must consist in a core $g$ with the integral property described in the White-square Lemma (Lemma 12.2.1) composed with two affine permutations $\eta$ and $\mu$ so that $f = \eta \circ g \circ \mu$. The TU-decomposition recovers $T$,$U$, $\eta'$ and $\mu'$ such that $f$ has the decomposition presented in Figure 12.11.

The following lemma specifies a simple decomposition of a function $g$ having the integral property of Lemma 12.2.1.

**Lemma 12.3.1** (TU-core Decomposition)**.** *Let $g$ be a function mapping $\mathbb{F}_2^n \times \mathbb{F}_2^n$ to itself such that fixing the right input to any value and letting the left one take all $2^n$ possible values leads to the left output taking all $2^n$ possible values. Then $g$ can be decomposed using a keyed $n$-bit permutation $T$ and a keyed $n$-bit function $U$ (see Figure 12.11):*

$$g(x,y) \;=\; \Big(T_y(x), U_{T_y(x)}(y)\Big),$$

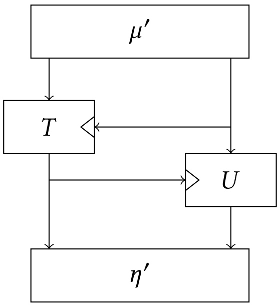


Figure 12.11: The TU-decomposition.

*Additionally, if $g$ is a permutation then $U$ is a keyed permutation.*

*Proof of Lemma 12.3.1.* We simply define $T_y(x)$ to be the left side of $g(x,y)$. Because of the multiset property, $T_y$ is a permutation for all $y$. We then define $U$ to be such that $U_k(y)$ is the right side of $g\big(T_y^{-1}(k),y\big)$.

If $g$ is a permutation then $(x,y) \mapsto g\big(T_y^{-1}(x),y)\big)$ is a permutation equal to $(x,y) \mapsto (x,U_x(y))$. In particular, it holds that $U_x$ is a permutation for all $x$, making it a keyed permutation. □

**Main Theorem 1** (The TU-Decomposition)**.** *Let $S : \mathbb{F}_2^n \to \mathbb{F}_2^n$ be an $n$-bit S-Box with* LAT *$\mathcal{L}$ and let $\eta$ and $\mu$ be linear permutations such that $\mathcal{L}'$ defined by $\mathcal{L}'[a,b] = [\mu(a),\eta(b)] = 0$ verifies $\mathcal{L}'[i,j] = 0$ for all $i < 2^{n/2}$ and $j < 2^{n/2}$. Then $S$ can be written as*

$$S = (\eta^t)^{-1} \circ \mathcal{U} \circ \mathcal{T} \circ \mu^t ,$$

*where $\mathcal{T}(x||y) = T_y(x)||y$ and $\mathcal{U}(x||y) = x||U_x(y)$. This corresponds to the structure in Figure 12.11, where $\mu' = \mu^t$ and $\eta' = (\eta^t)^{-1}$.*

*Proof.* Using Lemma 8.2.4, we derive that the function $s' = \eta^t \circ s \circ (\mu^{-1})^t$ has LAT $\mathcal{L}'$. Since the conditions of the theorem impose the presence of a white square in $\mathcal{L}'$, we can apply Lemma 12.3.1 and thus deduce that $s'$ can be decomposed into $\mathcal{U} \circ \mathcal{T}$. The result of the theorem is obtained by composing these decompositions. □

### 12.3.2 A Simple Example: the S-Box of Cmea

Examples of full-fledged TU-decompositions are provided in Chapters 13 (p. 245) and 14 (p. 267). Let us consider a simpler case where a previously unknown TU-core decomposition in the style of Lemma 12.3.1 can be obtained.

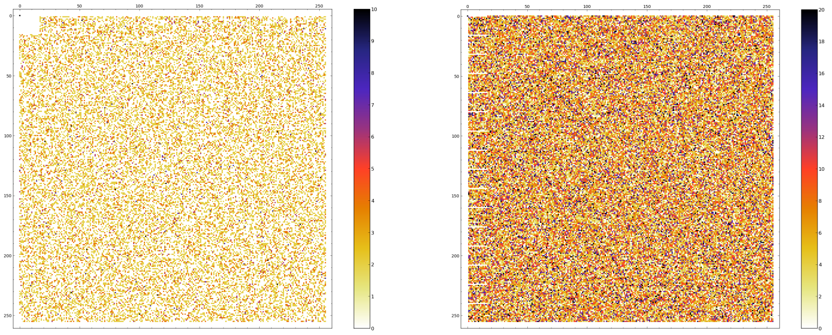
The S-Box of the block cipher Cmea is an 8-bit function which I denote $C$. Its look-up table is given in Table 12.2 and the cipher itself was briefly described in Section 2.2.3 (p. 38). As we can see, this S-Box has inner collisions such as $C(\mathtt{a5}) = C(\mathtt{bc}) = 0$. With the exception of that of Picaro, it is the only 8-bit non-injective S-Box listed in Section 8.3 (p. 146).

While the distribution of its linear and differential coefficients or on par with what would be expected of a random permutation, it yields a first algebraic pattern. Indeed, a random 8-bit Boolean function has algebraic degree 8 with probability 1/2. Yet, all of the 8 coordinates of $C$ have degree 7, an event which has probability $2^{-8}$.

|  | .0 | .1 | .2 | .3 | .4 | .5 | .6 | .7 | .8 | .9 | .A | .B | .C | .D | .E | .F |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0. | d9 | 23 | 5f | e6 | ca | 68 | 97 | b0 | 7b | f2 | 0c | 34 | 11 | a5 | 8d | 4e |
| 1. | 0a | 46 | 77 | 8d | 10 | 9f | 5e | 62 | f1 | 34 | ec | a5 | c9 | b3 | d8 | 2b |
| 2. | 59 | 47 | e3 | d2 | ff | ae | 64 | ca | 15 | 8b | 7d | 38 | 21 | bc | 96 | 00 |
| 3. | 49 | 56 | 23 | 15 | 97 | e4 | cb | 6f | f2 | 70 | 3c | 88 | ba | d1 | 0d | ae |
| 4. | e2 | 38 | ba | 44 | 9f | 83 | 5d | 1c | de | ab | c7 | 65 | f1 | 76 | 09 | 20 |
| 5. | 86 | bd | 0a | f1 | 3c | a7 | 29 | 93 | cb | 45 | 5f | e8 | 10 | 74 | 62 | de |
| 6. | b8 | 77 | 80 | d1 | 12 | 26 | ac | 6d | e9 | cf | f3 | 54 | 3a | 0b | 95 | 4e |
| 7. | b1 | 30 | a4 | 96 | f8 | 57 | 49 | 8e | 05 | 1f | 62 | 7c | c3 | 2b | da | ed |
| 8. | bb | 86 | 0d | 7a | 97 | 13 | 6c | 4e | 51 | 30 | e5 | f2 | 2f | d8 | c4 | a9 |
| 9. | 91 | 76 | f0 | 17 | 43 | 38 | 29 | 84 | a2 | db | ef | 65 | 5e | ca | 0d | bc |
| A. | e7 | fa | d8 | 81 | 6f | 00 | 14 | 42 | 25 | 7c | 5d | c9 | 9e | b6 | 33 | ab |
| B. | 5a | 6f | 9b | d9 | fe | 71 | 44 | c5 | 37 | a2 | 88 | 2d | 00 | b6 | 13 | ec |
| C. | 4e | 96 | a8 | 5a | b5 | d7 | c3 | 8d | 3f | f2 | ec | 04 | 60 | 71 | 1b | 29 |
| D. | 04 | 79 | e3 | c7 | 1b | 66 | 81 | 4a | 25 | 9d | dc | 5f | 3e | b0 | f8 | a2 |
| E. | 91 | 34 | f6 | 5c | 67 | 89 | 73 | 05 | 22 | aa | cb | ee | bf | 18 | d0 | 4d |
| F. | f5 | 36 | ae | 01 | 2f | 94 | c3 | 49 | 8b | bd | 58 | 12 | e0 | 77 | 6c | da |

Table 12.2: The look-up table of $C$. For example $C(\texttt{0x7A}) = \texttt{0x62}$.

However, the strongest patterns are in the Pollock representations of its DDT and LAT which are shown in Figures 12.12a and 12.12b respectively. There is a white-square at the top left corner of its DDT implying that all differentials $(0, \delta) \rightarrow (0, \delta')$ are impossible. Furthermore, its LAT contains white "dents" at rows $16 \times k$ on the 16 left-most columns. It means that composing it with a 4-bit branch swap $W : x||y \mapsto y||x$ of its input would yield a function with a TU-core decomposition.



(a) DDT.

(b) LAT.

Figure 12.12: The Pollock representations of the S-Box $C$ of CMEA.

It is indeed the case. The ciphers $T$ and $U$ obtained are given in Tables 12.3a and 12.3b respectively.

As expected since $C$ is not a permutation, the lines of $U$ are not permutations; but its columns are permutations. It means that the function $x \mapsto U_x(k)$ defined for any $k$ is a permutation. As a consequence, $C$ can be decomposed as shown in Figure 12.13a, where the code-book of the well defined mini-block cipher $U^t$ is obtained by transposing the Table 12.3b.

This decomposition can be further simplified into the one shown in Figure 12.13b, where $U'$ is the composition of $T$ and $U^t$.

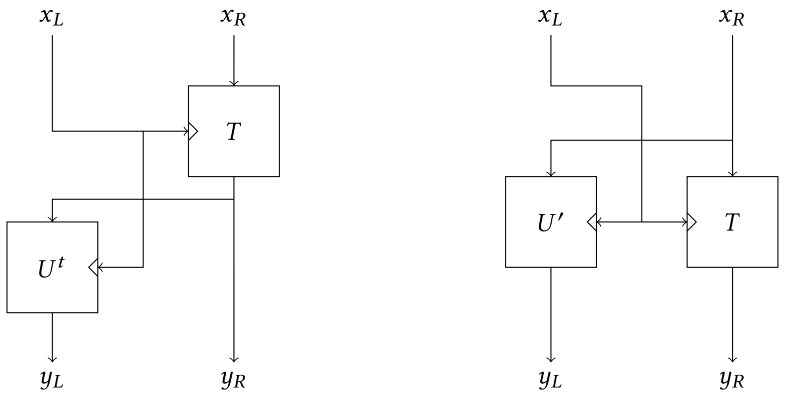
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | a | b | c | d | e | f |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $T_0$ | 9 | 3 | f | 6 | a | 8 | 7 | 0 | b | 2 | c | 4 | 1 | 5 | d | e |
| $T_1$ | a | 6 | 7 | d | 0 | f | e | 2 | 1 | 4 | c | 5 | 9 | 3 | 8 | b |
| $T_2$ | 9 | 7 | 3 | 2 | f | e | 4 | a | 5 | b | d | 8 | 1 | c | 6 | 0 |
| $T_3$ | 9 | 6 | 3 | 5 | 7 | 4 | b | f | 2 | 0 | c | 8 | a | 1 | d | e |
| $T_4$ | 2 | 8 | a | 4 | f | 3 | d | c | e | b | 7 | 5 | 1 | 6 | 9 | 0 |
| $T_5$ | 6 | d | a | 1 | c | 7 | 9 | 3 | b | 5 | f | 8 | 0 | 4 | 2 | e |
| $T_6$ | 8 | 7 | 0 | 1 | 2 | 6 | c | d | 9 | f | 3 | 4 | a | b | 5 | e |
| $T_7$ | 1 | 0 | 4 | 6 | 8 | 7 | 9 | e | 5 | f | 2 | c | 3 | b | a | d |
| $T_8$ | b | 6 | d | a | 7 | 3 | c | e | 1 | 0 | 5 | 2 | f | 8 | 4 | 9 |
| $T_9$ | 1 | 6 | 0 | 7 | 3 | 8 | 9 | 4 | 2 | b | f | 5 | e | a | d | c |
| $T_a$ | 7 | a | 8 | 1 | f | 0 | 4 | 2 | 5 | c | d | 9 | e | 6 | 3 | b |
| $T_b$ | a | f | b | 9 | e | 1 | 4 | 5 | 7 | 2 | 8 | d | 0 | 6 | 3 | c |
| $T_c$ | e | 6 | 8 | a | 5 | 7 | 3 | d | f | 2 | c | 4 | 0 | 1 | b | 9 |
| $T_d$ | 4 | 9 | 3 | 7 | b | 6 | 1 | a | 5 | d | c | f | e | 0 | 8 | 2 |
| $T_e$ | 1 | 4 | 6 | c | 7 | 9 | 3 | 5 | 2 | a | b | e | f | 8 | 0 | d |
| $T_f$ | 5 | 6 | e | 1 | f | 4 | 3 | 9 | b | d | 8 | 2 | 0 | 7 | c | a |

(a) $T$.

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | a | b | c | d | e | f |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $U_0$ | b | 1 | 0 | 7 | 2 | 1 | 8 | 3 | 3 | f | 0 | 0 | 6 | b | d | e |
| $U_1$ | 1 | f | 2 | d | f | f | d | b | 5 | 9 | 8 | 7 | 7 | 8 | 9 | 0 |
| $U_2$ | f | 6 | d | f | e | 6 | 1 | 6 | f | a | 4 | a | f | a | 2 | 1 |
| $U_3$ | 2 | b | e | 2 | 8 | 9 | f | c | 1 | 4 | 3 | 1 | c | e | 7 | c |
| $U_4$ | 3 | 3 | 6 | e | 4 | 7 | 5 | a | c | 8 | 1 | 4 | 0 | 0 | 3 | 9 |
| $U_5$ | a | a | 1 | 1 | 6 | 4 | 9 | 0 | e | 6 | 2 | c | b | 2 | 0 | f |
| $U_6$ | e | 4 | 9 | 5 | 7 | 8 | 2 | 9 | 8 | 7 | b | b | 9 | 6 | f | 3 |
| $U_7$ | 9 | 7 | 4 | 9 | c | a | 7 | 5 | 9 | 1 | e | 3 | d | c | 6 | 7 |
| $U_8$ | 6 | d | 3 | 8 | 3 | e | b | f | d | 3 | d | 8 | a | f | 1 | 5 |
| $U_9$ | d | c | 5 | 4 | 0 | 2 | e | 4 | a | 2 | c | d | 2 | 7 | 8 | 4 |
| $U_a$ | c | 0 | c | b | b | 0 | 3 | d | 7 | c | f | 5 | 5 | 4 | a | d |
| $U_b$ | 7 | 2 | 8 | c | a | c | 0 | 2 | b | d | a | 9 | 1 | 1 | c | 8 |
| $U_c$ | 0 | e | b | 3 | 1 | 3 | a | 7 | 6 | b | 7 | e | e | d | 5 | 6 |
| $U_d$ | 8 | 8 | 7 | 0 | 5 | b | 6 | e | 0 | 0 | 5 | 2 | 8 | 9 | 4 | b |
| $U_e$ | 4 | 5 | a | a | d | d | 4 | 8 | 4 | 5 | 9 | f | 4 | 3 | e | a |
| $U_f$ | 5 | 9 | f | 6 | 9 | 5 | c | 1 | 2 | e | 6 | 6 | 3 | 5 | b | 2 |

(b) $U$.

Table 12.3: The mini-block ciphers used to decompose $C \circ W$.



(a) Basic decomposition.

(b) Refined decomposition.

Figure 12.13: First decompositions of the S-Box $C$ of CMEA.

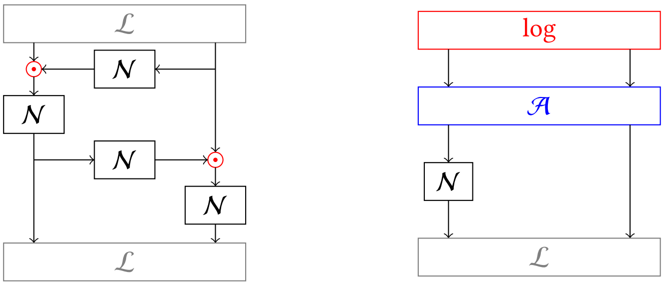
**Open Problem 12.3.1.** *Is there a decomposition of the mini-block ciphers $T$ and $U'$ used to build the S-Box of CMEA?*

Chapter 13

# Decomposing the GOST 8-bit S-Box

The Russian Federation has recently standardized two symmetric primitives: a hash function called Streebog [Fed12] and a block cipher called Kuznyechik [Fed15], which means “grasshopper” in Russian. These were developed by the Russian Technical Committee for standardization of “Cryptography and security mechanisms” (TC 26) which is supervised by the Russian Federal Security Service (FSB), i.e. the Russian counterpart of the American National Security Agency (NSA). These algorithms are specified in two different standards, GOST R 34.12–2015 for Kuznyechik and GOST R 34.11–2012 for Streebog. They should not be mistaken with the older GOST standard usually referred to as “GOST cipher” in the literature [Dol10a], a 64-bit Feistel network. Since this “GOST cipher” is still part of the Russian standard, GOST R 34.12–2015 calls the last iteration of this older cipher “Magma”.

In this chapter, I present two attempts made jointly with Alex Biryukov and Aleksei Udovenko at reverse-engineering the S-Box shared by these two algorithms. Our first decomposition is somewhat similar to a 2-round Feistel network with finite field multiplications instead of XORs. It is obtained in Section 13.2. The second one is based on a pseudo-exponential composed with a very weak and oddly structured. I explain how we obtained it in Section 13.3.



(a) First decomposition.

(b) Second decomposition.

Figure 13.1: A simplied view of our two decompositions of $\pi$.

Both decompositions are summarized in Figures 13.1a and 13.1b respectively. In these figures, linear (resp. nonlinear) functions are denoted $\mathcal{L}$ (resp. $\mathcal{N}$ ); $\odot$ denotes finite field multiplication; *log* is a finite field logarithm and $\mathcal{A}$ denotes a few simple integer arithmetic operations. Linear functions are represented in gray, finite field operations in red and integer operations in blue.

But first, I provide a brief description of the block cipher, Kuznyechik, the hash function Streebog and the properties of their S-Box in Section 13.1.

## 13.1 Preliminary

### 13.1.1 Description of Kuznyechik

It is a Substitution-Permutation Network which uses 9 rounds to encrypt a 128-bit block using a 256-bit key. The linear layer consists in multiplying the internal state by a $16 \times 16$ MDS matrix with elements in a finite field of size $2^8$. This matrix multiplication can be efficiently implemented using an LFSR.

The non-linearity is provided by an S-Box layer using an 8-bit S-Box $\pi$ given in Table 13.1. This S-Box is applied in parallel on the full state. There have been few third-party attacks of this cipher. The only one published to the best of our knowledge are a 5-round meet-in-the-middle attack [AY15a] and a 7-round integral/algebraic attack based on the zero-sum distinguishers described in Chapter 11 (p. 207) [BKP17]. This algorithm is sometimes named “Kuznechik”.

| | .0 | .1 | .2 | .3 | .4 | .5 | .6 | .7 | .8 | .9 | .A | .B | .C | .D | .E | .F |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0. | FC | EE | DD | 11 | CF | 6E | 31 | 16 | FB | C4 | FA | DA | 23 | C5 | 04 | 4D |
| 1. | E9 | 77 | F0 | DB | 93 | 2E | 99 | BA | 17 | 36 | F1 | BB | 14 | CD | 5F | C1 |
| 2. | F9 | 18 | 65 | 5A | E2 | 5C | EF | 21 | 81 | 1C | 3C | 42 | 8B | 01 | 8E | 4F |
| 3. | 05 | 84 | 02 | AE | E3 | 6A | 8F | A0 | 06 | 0B | ED | 98 | 7F | D4 | D3 | 1F |
| 4. | EB | 34 | 2C | 51 | EA | C8 | 48 | AB | F2 | 2A | 68 | A2 | FD | 3A | CE | CC |
| 5. | B5 | 70 | 0E | 56 | 08 | 0C | 76 | 12 | BF | 72 | 13 | 47 | 9C | B7 | 5D | 87 |
| 6. | 15 | A1 | 96 | 29 | 10 | 7B | 9A | C7 | F3 | 91 | 78 | 6F | 9D | 9E | B2 | B1 |
| 7. | 32 | 75 | 19 | 3D | FF | 35 | 8A | 7E | 6D | 54 | C6 | 80 | C3 | BD | 0D | 57 |
| 8. | DF | F5 | 24 | A9 | 3E | A8 | 43 | C9 | D7 | 79 | D6 | F6 | 7C | 22 | B9 | 03 |
| 9. | E0 | 0F | EC | DE | 7A | 94 | B0 | BC | DC | E8 | 28 | 50 | 4E | 33 | 0A | 4A |
| A. | A7 | 97 | 60 | 73 | 1E | 00 | 62 | 44 | 1A | B8 | 38 | 82 | 64 | 9F | 26 | 41 |
| B. | AD | 45 | 46 | 92 | 27 | 5E | 55 | 2F | 8C | A3 | A5 | 7D | 69 | D5 | 95 | 3B |
| C. | 07 | 58 | B3 | 40 | 86 | AC | 1D | F7 | 30 | 37 | 6B | E4 | 88 | D9 | E7 | 89 |
| D. | E1 | 1B | 83 | 49 | 4C | 3F | F8 | FE | 8D | 53 | AA | 90 | CA | D8 | 85 | 61 |
| E. | 20 | 71 | 67 | A4 | 2D | 2B | 09 | 5B | CB | 9B | 25 | D0 | BE | E5 | 6C | 52 |
| F. | 59 | A6 | 74 | D2 | E6 | F4 | B4 | C0 | D1 | 66 | AF | C2 | 39 | 4B | 63 | B6 |

Table 13.1: The look-up table of $\pi$. For example $\pi(\texttt{0x7A}) = \texttt{0xC6}$.

### 13.1.2 Description of Streebog

Streebog (also spelled Stribog) is a hash function standardized by Gost in 2012 [Fed12]. It uses a variant of the HAIFA construction [BD07] based on a 512-bit block cipher somewhat resembling the Aes. The overall structure of this algorithm is quite close to that of Whirlpool [BR00c].

The round constants of Streebog were chosen by feeding 12 different seeds into a round-constant-less version of the hash function with a modified linear layer [Rud15]. These seeds are given as hexadecimal strings of varying length which seem at first glance to lack any justification. However, they correspond to Russian names written backwards in Cyrillic and encoded in cp1251 as described in Table 13.2.

| $R$ | Hexadecimal seed | Name (Cyrillic) | Name (Latin) |
|---|---|---|---|
| 1 | `e2e5ede1e5f0c3` | Гребнев | Grebnev |
| 2 | `f7e8e2eef0e8ece8e4e0ebc220e9e5e3f0e5d1` | Сергей Владимирович | Sergej Vladimirovich |
| 3 | `f5f3ecc4` | Дмух | Dmukh |
| 4 | `f7e8e2eef0e4ede0f1eae5ebc020e9e5f0e4edc0` | Андрей Александрович | Andrej Aleksandrovich |
| 5 | `ede8e3fbc4` | Дыгин | Dygin |
| 6 | `f7e8e2eeebe9e0f5e8cc20f1e8ede5c4` | Денис Михайлович | Denis Mihajlovich |
| 7 | `ede8f5fef2e0cc` | Матюхин | Matjuhin |
| 8 | `f7e8e2eef0eef2eae8c220e9e8f0f2e8ecc4` | Дмитрий Викторович | Dmitrij Viktorovich |
| 9 | `e9eeeaf1e4f3d0` | Рудской | Rudskoj |
| 10 | `f7e8e2e5f0eee3c820f0e8ece8e4e0ebc2` | Владимир Игоревич | Vladimir Igorevich |
| 11 | `ede8eaf8e8d8` | Шишкин | Shishkin |
| 12 | `f7e8e2e5e5f1eae5ebc020e9e8ebe8f1e0c2` | Василий Алексеевич | Vasilij Alekseevich |

Table 13.2: The seeds used for each round constant of Streebog and the corresponding name.

### 13.1.3 Public and Basic Information about $\pi$

This S-Box is differentially 8-uniform with 25 occurrences of 8. The maximum LAT coefficient (in absolute value) is equal to 28 and occurs 14 times. By applying Equation (9.5) from Section 9.1.2.3 (p. 162), we see that the probabilities for a random 8-bit permutation to have differential and linear properties at least as good as those of $\pi$ are equal to, respectively, about $2^{-80.6}$ and $2^{-34.3}$. Thus, we can already rule out that this S-Box has been picked from a feasibly large set of random S-Boxes. It has to be the output of some generation algorithm, either based on hill-climbing or on a hidden structure.

It is worth noticing that all coordinates of $\pi$ have a maximum algebraic degree equal to 7. There is no specific pattern in its HDIM that I can see:

$$\hat{H}(\pi) = \begin{bmatrix} 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \end{bmatrix} .$$

There is very little public information available about this S-Box.

In a first presentation at RusCrypto'13 [Shi13] given by Shishkin on behalf of the FSB, some information about the design process of the S-Box was given. It is supposed not to have an analytic structure — even if that means not having optimal cryptographic properties unlike e.g. the S-Box of the AES — and to minimise the number of operations necessary to compute it so as to optimize hardware and vectorized software implementations. However, specifics about the method used to achieve this goal were not given.

One year later, at CTCrypt'14, the authors explained that one guideline they used to design Kuznyechik was: "only well examined constructions and transformations are used as building blocks" (slide 7 of [SDL+14]). As we will see, this seems to be contradicted by the structures we found. This might indicate that the structure they hid is one they have studied intensively without disclosing their results.

The most relevant data was gathered by Markku Saarinen during informal conversations with the designers held during CTCrypt'14 which he reports on in [SB15]:

> [The designers'] recollection was that the aim was to choose a "randomized" S-Box that meets the basic differential, linear, and algebraic requirements. Randomization using various building blocks was simply iterated until a "good enough" permutation was found. This was seen as an effective countermeasure against yet-unknown attacks [such as algebraic attacks].

## 13.2 A Feistel-like Decomposition

Our first decomposition resembles a 2-round Feistel network. We obtained it by first identifying visual artifacts in the LAT of $\pi$ using the method described in Section 13.2.1 (p. 248). We deduced a TU-decomposition of $\pi$ in the style of Theorem 1 (p. 241) and decomposed the mini-block cipher $T$ and $U$ we obtained. It is explained in Section 13.2.2 (p. 250). Finally, this first decomposition is discussed in Section 13.2.3 (p. 256).

### 13.2.1 Patterns in the LAT of Kuznyechik

As all the structural attacks presented in the previous chapter failed against this S-Box. However, the Pollock representation of its LAT contains interesting patterns. The Pollock representation of the absolute value of the coefficients of the LAT of $\pi$ is given in Figure 13.2a. While it may be hard to see on paper, blurry vertical lines appear when looking at a large enough version of this picture. In order to better see this pattern, we introduce the so-called $\oplus$-*texture*. It is a kind of auto-correlation.

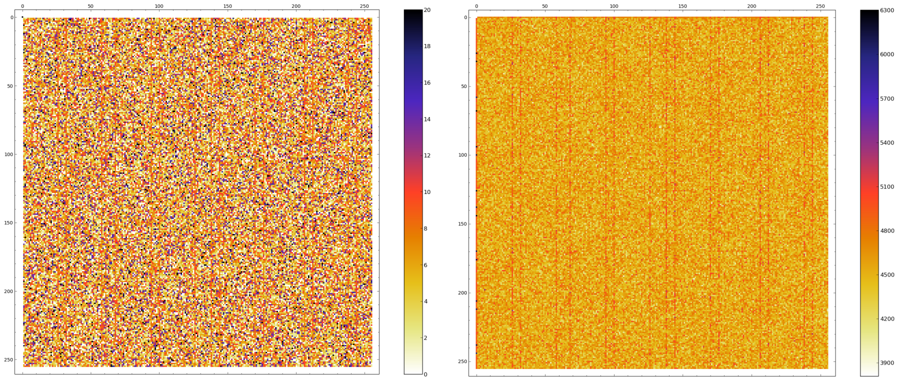
**Definition 13.2.1.** *We call* $\oplus$-texture *of the* LAT $\mathcal{L}$ *of an S-Box the matrix* $\mathcal{T}^{\oplus}$ *with coefficients* $\mathcal{T}^{\oplus}[i,j]$ *defined as:*

$$\mathcal{T}^{\oplus}[i,j] \;=\; \#\Big\{(x,y), |\mathcal{L}[x \oplus i, y \oplus j]| = |\mathcal{L}[x,y]|\Big\}.$$

The Jackson Pollock representation of the $\oplus$-texture of the LAT $\mathcal{L}_\pi$ of $\pi$ is given in Figure 13.2b. The lines are now much more obvious and, furthermore, we observe dark dots in the very first column. The indices of both the rows containing the black dots and the columns containing the lines are the same and correspond to a binary vector space $\mathcal{V}$ defined, using hexadecimal notation, as:

$$\mathcal{V} = \{\texttt{00},\texttt{1a},\texttt{20},\texttt{3a},\texttt{44},\texttt{5e},\texttt{64},\texttt{7e},\texttt{8a},\texttt{90},\texttt{aa},\texttt{b0},\texttt{ce},\texttt{d4},\texttt{ee},\texttt{f4}\}.$$

In order to cluster the columns together to the left of the picture and the dark dots to the top of it, we can apply a linear mapping $L$ to obtain a new table $\mathcal{L}'_\pi$ where $\mathcal{L}'_\pi[i,j] = \mathcal{L}_\pi[L(i), L(j)]$. We define $L$ so that it maps $i \in \mathbb{F}_2^4$ to the $i$-th element of $\mathcal{V}$



(a) Pollock representation.

(b) ⊕-texture.

Figure 13.2: The LAT of $\pi$ (absolute value).

and then complete it in a natural way to obtain a linear permutation of $\mathbb{F}_2^8$. It maps each bit as described below in hexadecimal notation:

$$\begin{cases} L(\texttt{01}) = \texttt{1a},\ L(\texttt{02}) = \texttt{20},\ L(\texttt{04}) = \texttt{44},\ L(\texttt{08}) = \texttt{8a}, \\ L(\texttt{10}) = \texttt{01},\ L(\texttt{20}) = \texttt{02},\ L(\texttt{40}) = \texttt{04},\ L(\texttt{80}) = \texttt{08}. \end{cases}$$

The Jackson Pollock representation of $\mathcal{L}'_\pi$ is given in Figure 13.3. As we can see, it is highly structured: there is a $16 \times 16$ square containing[1] only coefficients equal to 0 in the top left corner. Furthermore, the left-most 15 bits to the right of column 0, exhibit a strange pattern: each of the coefficients in it has an absolute value in $[4, 12]$ although the maximum coefficient in the table is equal to 28. This forms a sort of low-contrast "stripe". The low number of different values it contains implies a low number of color in the corresponding columns in $\mathcal{L}_\pi$, which in turn corresponds to the lines we were able to distinguish in Figure 13.2a.

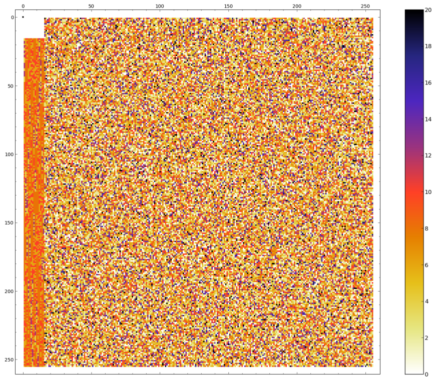


Figure 13.3: The Jackson Pollock representation of $\mathcal{L}'_\pi$, where $\mathcal{L}'_\pi[i,j] = \mathcal{L}_\pi[L(i), L(j)]$.

These column can also be identified using the method described in Section 9.1.2.4 (p. 163). It consists of looking at the variance of the absolute value of the coefficients

[1] Except of course in position $(0, 0)$ where the bias is equal to the maximum of 128.

in the rows and the columns of the LAT. The corresponding data for $\pi$ is provided in Figure 13.4. The columns for which the variance is abnormally low have their indices in $\mathcal{V}$.

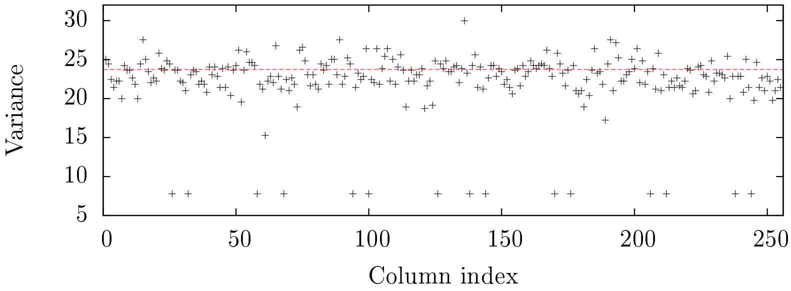


Figure 13.4: The variance of the columns of the LAT (absolute values) of $\pi$. The expected variance is represented with a red dashed line.

It is natural to try and build another S-Box from $\pi$ such that its LAT is equal to $\mathcal{L}'_\pi$. By applying Lemma 8.2.4, we obtain that $L^t \circ \pi \circ (L^t)^{-1}$ has $\mathcal{L}'_\pi$ as its LAT. The mapping $L^t$ consists of a linear Feistel round followed by a permutation of the left and right 4-bit nibbles (which we denote swapNibbles). To simplify the modifications we make, we remove the nibble permutation and define

$$\pi' = L^* \circ \pi \circ L^*$$

where $L^*$ is the Feistel round in $L^t$ and is described in Figure 13.5.

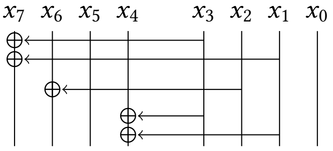


Figure 13.5: A circuit computing $L^*$ where its input is given in binary.

### 13.2.2 A TU-Decomposition of $\pi$

This affine-equivalent S-Box $\pi'$ is highly structured. First of all, the LAT of $\pi'$ composed with swapNibbles both before and after is $\mathcal{L}'_\pi$, with its white square in the top left and strange left side.

Furthermore, the presence of this white-square means we can apply Lemma 12.3.1. We deduce that $\pi'$ can be decomposed into the structure recalled in Figure 13.6, where $T$ and $U$ are mini-block ciphers with 4-bit block size and 4-bit keys. Their full code-books are given in Tables 13.3a and 13.3b respectively.

We decompose the mini-block ciphers $T$ and $U$ themselves in Section 13.2.2.1 and Section 13.2.2.2 respectively.

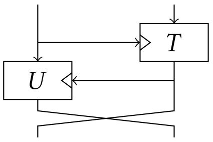


Figure 13.6: The high level structure of $\pi'^{-1}$.

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | a | b | c | d | e | f |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $T_0$ | e | f | 2 | 5 | 7 | b | 8 | 1 | 3 | c | d | a | 0 | 9 | 4 | 6 |
| $T_1$ | 2 | 9 | a | 4 | e | 6 | 7 | b | 1 | 8 | 3 | d | 0 | c | f | 5 |
| $T_2$ | e | f | 2 | 5 | 7 | b | 8 | 1 | 3 | c | d | a | 0 | 9 | 4 | 6 |
| $T_3$ | 5 | d | 4 | 2 | 6 | 7 | b | 8 | c | 1 | 9 | f | 0 | 3 | a | e |
| $T_4$ | 5 | e | 6 | 7 | 4 | 3 | f | a | 0 | 1 | d | 2 | 8 | b | c | 9 |
| $T_5$ | 9 | d | f | a | c | 6 | 8 | 1 | 0 | 5 | b | 3 | 2 | 4 | e | 7 |
| $T_6$ | 3 | 9 | d | f | 1 | e | b | 8 | 0 | 2 | 7 | c | 4 | a | 5 | 6 |
| $T_7$ | 5 | e | 6 | 7 | 4 | 3 | f | a | 0 | 1 | d | 2 | 8 | b | c | 9 |
| $T_8$ | 7 | b | 8 | 5 | 9 | d | c | 3 | 2 | e | a | f | 6 | 1 | 0 | 4 |
| $T_9$ | d | f | a | c | e | 6 | 2 | 5 | 1 | 3 | b | 7 | 9 | 4 | 0 | 8 |
| $T_a$ | e | 6 | 7 | 4 | c | 3 | 8 | 1 | a | 2 | d | 9 | 5 | b | 0 | f |
| $T_b$ | 4 | 2 | 5 | d | b | 8 | 6 | 7 | 9 | f | c | 1 | a | e | 0 | 3 |
| $T_c$ | 2 | 5 | a | 4 | 3 | 9 | d | 8 | c | f | 0 | 7 | b | 1 | 6 | e |
| $T_d$ | e | 6 | 2 | 5 | d | f | a | c | 9 | 4 | 0 | 8 | 1 | 3 | b | 7 |
| $T_e$ | 9 | d | c | 3 | 7 | b | 8 | 5 | 6 | 1 | 0 | 4 | 2 | e | a | f |
| $T_f$ | 8 | 1 | 7 | b | 2 | 5 | e | f | 4 | 6 | 0 | 9 | d | a | 3 | c |

(a) $T$.

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | a | b | c | d | e | f |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $U_0$ | 8 | f | 0 | 2 | d | 5 | 6 | 9 | e | 3 | 1 | 7 | c | b | 4 | a |
| $U_1$ | 8 | c | 7 | 3 | d | f | 2 | 0 | e | 4 | 1 | b | 6 | 5 | 9 | a |
| $U_2$ | 3 | 4 | e | 9 | d | 8 | 0 | 5 | 1 | 2 | c | f | 7 | b | a | 6 |
| $U_3$ | b | 8 | 9 | a | 0 | 7 | 2 | 5 | f | 6 | d | 4 | 1 | e | 3 | c |
| $U_4$ | c | 2 | 5 | b | e | 8 | 7 | 1 | 4 | f | d | 6 | 9 | 3 | 0 | a |
| $U_5$ | 4 | e | 2 | 8 | 3 | 7 | 5 | 1 | a | b | c | d | f | 6 | 9 | 0 |
| $U_6$ | f | 6 | b | 2 | 3 | 0 | 7 | 4 | 5 | d | 1 | 9 | e | 8 | a | c |
| $U_7$ | 7 | a | c | 1 | e | f | 5 | 4 | b | 9 | 0 | 2 | 8 | d | 3 | 6 |
| $U_8$ | a | f | b | e | c | 4 | d | 5 | 7 | 0 | 6 | 1 | 8 | 3 | 9 | 2 |
| $U_9$ | 2 | 3 | c | d | 1 | b | f | 5 | 9 | 4 | 7 | a | e | 6 | 0 | 8 |
| $U_a$ | 9 | b | 5 | 7 | 1 | c | d | 0 | 6 | 2 | a | e | f | 8 | 3 | 4 |
| $U_b$ | 1 | 7 | 2 | 4 | c | 3 | f | 0 | 8 | 6 | b | 5 | 9 | d | a | e |
| $U_c$ | 6 | d | e | 5 | 2 | c | a | 4 | 3 | f | b | 7 | 1 | 0 | 9 | 8 |
| $U_d$ | e | 1 | 9 | 6 | f | 3 | 8 | 4 | d | b | a | c | 7 | 5 | 0 | 2 |
| $U_e$ | 5 | 9 | 0 | c | f | 4 | a | 1 | 2 | d | 7 | 8 | 6 | b | 3 | e |
| $U_f$ | d | 5 | 7 | f | 2 | b | 8 | 1 | c | 9 | 6 | 3 | 0 | e | a | 4 |

(b) $U$.

Table 13.3: The mini-block ciphers used to decompose $L^* \circ \pi'^{-1} \circ L^*$.

#### 13.2.2.1 Reverse-Engineering T

The mini-block cipher $T'$ defined as $T'_k : x \mapsto T_k\big(x \oplus t_{\text{in}}(k) \oplus 0xC\big)$ for $t_{\text{in}}(k) = 0||k_2||k_3||0$ (see Table 13.4) is such that $T'_k(0) = 0$ for all $k$.

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | a | b | c | d | e | f |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $T'_0$ | 0 | 9 | 4 | 6 | 3 | c | d | a | 7 | b | 8 | 1 | e | f | 2 | 5 |
| $T'_1$ | 0 | c | f | 5 | 1 | 8 | 3 | d | e | 6 | 7 | b | 2 | 9 | a | 4 |
| $T'_2$ | 0 | 9 | 4 | 6 | 3 | c | d | a | 7 | b | 8 | 1 | e | f | 2 | 5 |
| $T'_3$ | 0 | 3 | a | e | c | 1 | 9 | f | 6 | 7 | b | 8 | 5 | d | 4 | 2 |
| $T'_4$ | 0 | 1 | d | 2 | 8 | b | c | 9 | 5 | e | 6 | 7 | 4 | 3 | f | a |
| $T'_5$ | 0 | 5 | b | 3 | 2 | 4 | e | 7 | 9 | d | f | a | c | 6 | 8 | 1 |
| $T'_6$ | 0 | 2 | 7 | c | 4 | a | 5 | 6 | 3 | 9 | d | f | 1 | e | b | 8 |
| $T'_7$ | 0 | 1 | d | 2 | 8 | b | c | 9 | 5 | e | 6 | 7 | 4 | 3 | f | a |
| $T'_8$ | 0 | 4 | 6 | 1 | a | f | 2 | e | c | 3 | 9 | d | 8 | 5 | 7 | b |
| $T'_9$ | 0 | 8 | 9 | 4 | b | 7 | 1 | 3 | 2 | 5 | e | 6 | a | c | d | f |
| $T'_a$ | 0 | f | 5 | b | d | 9 | a | 2 | 8 | 1 | c | 3 | 7 | 4 | e | 6 |
| $T'_b$ | 0 | 3 | a | e | c | 1 | 9 | f | 6 | 7 | b | 8 | 5 | d | 4 | 2 |
| $T'_c$ | 0 | 7 | c | f | 6 | e | b | 1 | a | 4 | 2 | 5 | d | 8 | 3 | 9 |
| $T'_d$ | 0 | 8 | 9 | 4 | b | 7 | 1 | 3 | 2 | 5 | e | 6 | a | c | d | f |
| $T'_e$ | 0 | 4 | 6 | 1 | a | f | 2 | e | c | 3 | 9 | d | 8 | 5 | 7 | b |
| $T'_f$ | 0 | 9 | 4 | 6 | 3 | c | d | a | 7 | b | 8 | 1 | e | f | 2 | 5 |

Table 13.4: A modified version $T'$ of the mini-block cipher $T$.

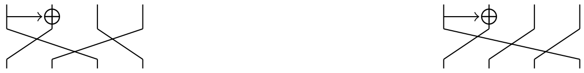
Furthermore, $T'$ is such that all lines of $T'_k$ can be obtained through a linear combination of $T'_6, T'_7, T'_8$ and $T'_9$ as follows:

$$\begin{array}{llllll} T'_0 & = T'_7 \oplus T'_9 & T'_1 & = T'_8 \oplus T'_9 & T'_2 & = T'_7 \oplus T'_9 \\ T'_3 & = T'_6 \oplus T'_7 & T'_4 & = T'_7 & T'_5 & = T'_7 \oplus T'_8 \\ T'_a & = T'_6 \oplus T'_7 \oplus T'_8 \oplus T'_9 & T'_b & = T'_6 \oplus T'_7 & T'_c & = T'_6 \oplus T'_7 \oplus T'_8 \\ T'_d & = T'_9 & T'_e & = T'_8 & T'_f & = T'_7 \oplus T'_9 . \end{array} \tag{13.1}$$

Besides, $T'_6, T'_7, T'_8$ and $T'_9$ are all affine equivalent. Indeed, the linear mapping $A$ defined by $A : 1 \mapsto 4, 2 \mapsto 1, 4 \mapsto 8, 8 \mapsto a$ (see Figure 13.7a) is such that:

$$\begin{array}{llll} T'_7 & = A & \circ & T'_6 \\ T'_8 & = A^2 & \circ & T'_6 \\ T'_9 & = A^3 & \circ & T'_6 . \end{array}$$

Let us denote swap2lsb the operation consisting of swapping the two least significant bits of a 4-bit word. By composing $A$ with swap2lsb before and after to obtain $\hat{A} = \text{swap2lsb} \circ A \circ \text{swap2lsb}$, we obtain a clear LFSR structure displayed in Figure 13.7b.



(a) Definition of $A$.

(b) A representation of $\hat{A}$.

Figure 13.7: The mapping used to generate $T'_7, T'_8$ and $T'_9$ from $T'_6$.

We deduce the LFSR polynomial to be $X^4 + X^3 + 1$. This points towards finite field multiplication and, indeed, the mapping $\hat{A}$ can be viewed as a multiplication by $X$ in $\mathbb{F}_{2^4} = \mathbb{F}_2[X]/(X^4 + X^3 + 1)$. To fit the swap into the original TU-decomposition, we modify $T'_6$ and the bottom linear layer. We use the fact that

$$A^i = (\text{swap2lsb} \circ \hat{A} \circ \text{swap2lsb})^i = \text{swap2lsb} \circ \hat{A}^i \circ \text{swap2lsb} \quad \text{for } i = 0, 1, \ldots,$$

to merge one swap2lsb into $T'_6$ and move the other swap2lsb through XORs outside $T'$. If we let $t = \text{swap2lsb} \circ T'_6$ then $\text{swap2lsb} \circ T'_k(x)$ is a linear combination of $X^i \odot t(x)$ for some exponents $i \in \{0, 1, 2, 3\}$ and $\odot$ is multiplication in the specified field. In the end, $T$ can be expressed as

$$T_k(x) = \text{swap2lsb}\left(f(k) \odot t\Big(x \oplus t_{\text{in}}(k) \oplus 0xC\Big)\right),$$

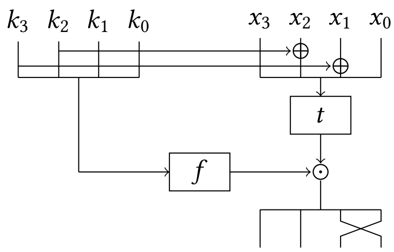
where $f$ captures the linear relations described in Equations (13.1). Both $f$ and $t$ are given in Figure 13.8b and a picture representing the structure of $T$ is given in Figure 13.8a.

Note that $f(x)$ is never equal to 0: if it were the case then the function would not be invertible. On the other hand, the inverse of $T_k$ is easy to compute: $f$ must be replaced by $1/f$ where the inversion is done in the finite field $\mathbb{F}_{2^4}$, $t$ by its functional inverse $t^{-1}$ and the order of the operations must be reversed.

#### 13.2.2.2 Reverse-Engineering U

We can easily deduce from Table 13.3b that for $k \neq 0$, $U_k$ can be expressed as

$$U_k(x) = \Big(k_3 \times U_8(x)\Big) \oplus \Big(k_2 \times U_4(x)\Big) \oplus \Big(k_1 \times U_2(x)\Big) \oplus \Big(k_0 \times U_1(x)\Big)$$



(a) The high-level structure of $T$.

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | a | b | c | d | e | f |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $f$ | a | c | a | 3 | 2 | 6 | 1 | 2 | 4 | 8 | f | 3 | 7 | 8 | 4 | a |
| $t$ | 2 | d | b | 8 | 3 | a | e | f | 4 | 9 | 6 | 5 | 0 | 1 | 7 | c |

(b) Mappings $f$ and $t$.

Figure 13.8: The components of our decomposition of the mini-block cipher $T$.

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | a | b | c | d | e | f |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $B_2$ | 5 | c | 0 | 9 | 2 | b | 7 | e | 3 | a | 6 | f | 4 | d | 1 | 8 |
| $B_4$ | 1 | d | 7 | b | f | 3 | 9 | 5 | c | 0 | a | 6 | 2 | e | 4 | 8 |
| $B_8$ | 5 | 6 | d | e | 0 | 3 | 8 | b | a | 9 | 2 | 1 | f | c | 7 | 4 |

Table 13.5: Affine functions such that $U_k = B_k \circ U_1$ for $k \in \{2, 4, 8\}$.

where $k = \sum_{i \leq 3} k_i 2^i$ and "$\times$" denotes a simple multiplication. We also notice that the permutations $U_2, U_4$ and $U_8$ can all be derived from $U_1$ using some affine functions $B_k$ so that $U_k = B_k \circ U_1$. The values of $B_k(x)$ are given in Table 13.5.

If we let $B(x) = B_4(x) \oplus 1$ then $B_2(x) = B^{-1}(x) \oplus 5$ and $B_8(x) = B^2(x) \oplus 5$. Thus, we can define a linear function $u_{\text{out}}$ such that

$$\begin{aligned}
U_1(x) &= B^0 &\circ\, U_1(x) &\oplus u_{\text{out}}(1)\\
U_2(x) &= B^{-1} &\circ\, U_1(x) &\oplus u_{\text{out}}(2)\\
U_4(x) &= B^1 &\circ\, U_1(x) &\oplus u_{\text{out}}(4)\\
U_8(x) &= B^2 &\circ\, U_1(x) &\oplus u_{\text{out}}(8).
\end{aligned} \tag{13.2}$$

Let $M_2$ be the matrix representation of the multiplication by $X$ in the finite field we used to decompose $T$, namely $\mathbb{F}_{2^4} = \mathbb{F}_2[X]/(X^4 + X^3 + 1)$. The linear mapping $u_f$ defined by $u_f : 1 \mapsto 5, 2 \mapsto 2, 4 \mapsto 6, 8 \mapsto 8$ is such that $B = u_f \circ M_2 \circ u_f^{-1}$ so that Equations (13.2) can be re-written as

$$\begin{aligned}
U_1(x) &= \left(u_f \circ M_2^0 \circ u_f^{-1} \circ U_1\right)(x) \oplus u_{\text{out}}(1)\\
U_2(x) &= \left(u_f \circ M_2^{-1} \circ u_f^{-1} \circ U_1\right)(x) \oplus u_{\text{out}}(2)\\
U_4(x) &= \left(u_f \circ M_2^1 \circ u_f^{-1} \circ U_1\right)(x) \oplus u_{\text{out}}(4)\\
U_8(x) &= \left(u_f \circ M_2^2 \circ u_f^{-1} \circ U_1\right)(x) \oplus u_{\text{out}}(8).
\end{aligned} \tag{13.3}$$

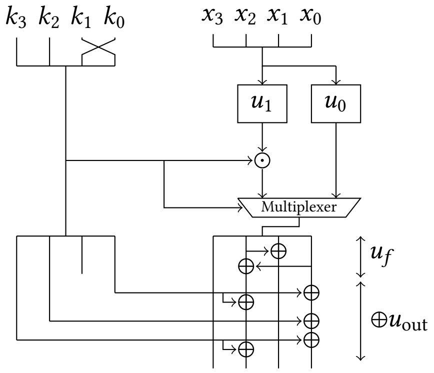
If we swap the two least significant bits of $k$, then the exponents of matrix $M_2$ will go in ascending order: $(-1, 0, 1, 2)$. Let $u_1 = M_2^{-1} \circ u_f^{-1} \circ U_1$. Since $M_2$ is the multiplication by $X$ in the finite field, we can write the following expression for $U_k$ (when $k \neq 0$):

$$U_k(x) = u_f\Big(u_1(x) \odot \mathsf{swap2lsb}(k)\Big) \oplus u_{out}(k). \tag{13.4}$$

The complete decomposition of $U$ is presented in Figure 13.9. It uses the 4-bit permutations $u_0$ and $u_1$ specified in Table 13.9b. We could not find a relation between $u_1$ and $u_0 = u_f^{-1} \circ U_0$ so there has to be a conditional branching: $U$ selects the result of

Equation (13.4) if $k \neq 0$ and the result of $u_0(x)$ otherwise before applying $u_f$. This is achieved using a multiplexer which returns the output of $u_0$ if $k_3 = k_2 = k_1 = k_0 = 0$, and returns the output of $u_1$ if it is not the case. In other words, $U$ can be computed as follows:

$$U_k(x) = \begin{cases} u_f\Big(u_1(x) \odot \mathsf{swap2lsb}(k)\Big) \oplus u_{\text{out}}(k), & \text{if } k \neq 0 \\ u_f\Big(u_0(x)\Big) & \text{if } k = 0. \end{cases}$$



(a) A high level view of $U$.

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | a | b | c | d | e | f |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $u_0$ | 8 | b | 0 | 2 | 9 | 1 | 4 | f | c | 5 | 7 | 3 | e | d | 6 | a |
| $u_1$ | 4 | 7 | d | e | 8 | 9 | 1 | 0 | 6 | 3 | f | a | 2 | c | b | 5 |

(b) The permutations $u_0$ and $u_1$.

Figure 13.9: The structure of the mini-block cipher $U$ and its components.

#### 13.2.2.3 A Structure for $\pi$

In Sections 13.2.2.1 and 13.2.2.2, we decomposed the two mini-block ciphers $T$ and $U$ which can be used to build $\pi'^{-1}$, the inverse of $L^* \circ \pi \circ L^*$. These mini-block ciphers are based on the non-linear 4-bit functions $f, t, u_0, u_1$, two finite field multiplications, a "trick" to bypass the non-invertibility of multiplication by 0 and simple linear functions. Let us now use the expressions we identified to express $\pi$ itself.

First, we associate the linear functions identified as parts of the decompositions of $T$ and $U$ with $L^*$ to form $\alpha$ and $\omega$, two linear permutations applied respectively at the beginning and the end of the computation.

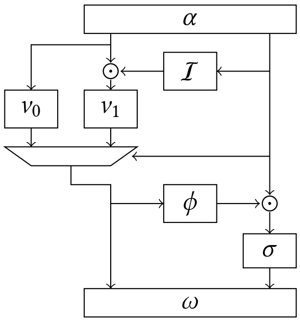
**Input layer $\alpha$.** First of all, we need to apply $L^*$ as well as the the swap of the left and right branches (swapNibbles) present in the high level decomposition of $\pi'^{-1}$ (see Figure 13.6). Then, we note that the key in $U$ needs a swap of its 2 bits of lowest weight (swap2lsb) and that the ciphertext of $T$ needs the same swap. Thus, we simply apply swap2lsb. Then, we apply the addition of $u_{\text{out}}$ and the inverse of $u_f$.

**Output layer $\omega$.** This function is simpler: it is the composition of the addition of $t_{\text{in}}$ and of $L^*$.

The matrix representations of these layers are

$$\alpha = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}, \omega = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}.$$

**Deducing $\pi$.** In order to invert $U$, we define $v_0 = u_0^{-1}$ and $v_1 = u_1^{-1}$. If $\ell = 0$, then the output of the inverse of $U$ is $v_0(r)$, otherwise it is $v_1\big(r \odot \mathcal{I}(\ell)\big)$, where $\mathcal{I} : x \mapsto x^{14}$ is the multiplicative inverse in $\mathbb{F}_{2^4}$. To invert $T$, we define $\sigma = t^{-1}$ and $\phi = \mathcal{I} \circ f$ and compute $\sigma\big(\phi(\ell) \odot r\big)$.



(a) High level structure.

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | a | b | c | d | e | f |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $\mathcal{I}$ | 0 | 1 | c | 8 | 6 | f | 4 | e | 3 | d | b | a | 2 | 9 | 7 | 5 |
| $v_0$ | 2 | 5 | 3 | b | 6 | 9 | e | a | 0 | 4 | f | 1 | 8 | d | c | 7 |
| $v_1$ | 7 | 6 | c | 9 | 0 | f | 8 | 1 | 4 | 5 | b | e | d | 2 | 3 | a |
| $\phi$ | b | 2 | b | 8 | c | 4 | 1 | c | 6 | 3 | 5 | 8 | e | 3 | 6 | b |
| $\sigma$ | c | d | 0 | 4 | 8 | b | a | e | 3 | 9 | 5 | 2 | f | 1 | 6 | 7 |

(b) Non-linear functions.

Figure 13.10: The first decomposition of $\pi$.

Figure 13.10a summarizes how to compute $\pi$ using these components. The non-linear functions are all given in Table 13.10b. A Sage [Dev16] script performing those computations can be downloaded on Github.[2] Pseudo-code computing $\pi$ using our components is provided in Algorithm 13.1.

**Algorithm 13.1** $\pi$ evaluation ;
**Input:** 8-bit block $x$ ;
**Output:** 8-bit block $y$

```
$x \leftarrow \alpha(x)$
$\ell \mid\mid r \leftarrow x$                                ▷ $\ell$ and $r$ are 4-bit long.
if $r = 0$ then
    $\ell \leftarrow v_0(\ell)$
else
    $\ell \leftarrow v_1\left(\ell \odot \mathcal{I}(r)\right)$
end if
$r \leftarrow \sigma\big(r \odot \phi(l)\big)$
$y \leftarrow \ell \mid\mid r$
return $y$
```

[2] https://github.com/picarresursix/GOST-pi

### 13.2.3 Studying this Decomposition of $\pi$

The knowledge of this first decomposition can give us more information about the properties of $\pi$. In Section 13.2.3.1 (p. 256), we list some properties of the non-linear 4-bit components we extracted. In Section 13.2.3.2 (p. 256), we comment on the overall structure used. Finally, we discuss the impact of this decomposition over the hardware implementation of $\pi$ in Sections 13.2.3.3 (p. 259).

#### 13.2.3.1 Analyzing the Components

Table 13.6 summarizes the properties of the non-linear components of our decomposition. While it is not hard to find 4-bit permutations with a differential uniformity of 4, we see that none of the components chosen do except for the inverse function. We can thus discard the idea that the strength of $\pi$ against differential and linear attacks relies on the individual resilience of each of its components.

| | 1-to-1 | Best differentials and their probabilities | Best linear approximations and their probabilities |
|---|---|---|---|
| $\phi$ | No | 1 $\rightsquigarrow$ d (8/16) | 3 $\rightsquigarrow$ 8 (2/16), 7 $\rightsquigarrow$ d (2/16) |
| $\sigma$ | Yes | f $\rightsquigarrow$ b (6/16) | 1 $\rightsquigarrow$ f (14/16) |
| $\nu_0$ | Yes | 6 $\rightsquigarrow$ c (6/16), e $\rightsquigarrow$ e (6/16) | 30 approximations $(8 \pm 4)/16$ |
| $\nu_1$ | Yes | 9 $\rightsquigarrow$ 2 (16/16) | 8 approximations $(8 \pm 6)/16$ |

Table 13.6: Linear and differential properties of the components of $\pi$.

As can be seen in Table 13.6, there is a probability 1 differential in $\nu_1$: 9 $\rightsquigarrow$ 2. Furthermore, a difference equal to 2 on the left branch corresponds to a 1 bit difference on bit 5 of the input of $\omega$, a bit which is left unchanged by $\omega$.

The structure itself also implies the existence of a truncated differential with high probability. Indeed, if the value on the left branch is equal to 0 for two different inputs, then the output difference on the left branch will remain equal to 0 with probability 1. This explains why the probability that a difference in $\Delta_{\text{in}} = \{\alpha^{-1}(\ell||0), \ell \in \mathbb{F}_2^4, \ell \neq 0\}$ is mapped to a difference in $\Delta_{\text{out}} = \{\omega(\ell||0), \ell \in \mathbb{F}_2^4, \ell \neq 0\}$ is higher than the expected $2^{-4}$:

$$\frac{1}{2^4 - 1} \sum_{\delta \in \Delta_{\text{in}}} P[\pi(x \oplus \delta) \oplus \pi(x) \in \Delta_{\text{out}}] = \frac{450}{(2^4 - 1) \times 2^8} \approx 2^{-3}.$$

#### 13.2.3.2 Comments on the Structure Used

We define $\hat{\pi}$ as $\omega^{-1} \circ \pi \circ \alpha^{-1}$, i.e. $\pi$ minus its whitening linear layers.

The structure of $\hat{\pi}$ is similar to a 2-round combination of a Misty-like and Feistel structure where the xors have been replaced by finite field multiplications. To the best of our knowledge, this is the first time such a structure has been used in cryptography. There are sophisticated lightweight decompositions of the S-Box of the Aes which rely on finite field multiplications in $\mathbb{F}_{2^4}$, for instance in [Can05]. However, the high level structure used in this case is quite different. If $\pi$ corresponds to such a decomposition then we could not find what it corresponds to. Recall in particular that $\pi$ cannot be affine-equivalent to a monomial.

The use of finite field multiplications in such a structure yields a problem: if the output of the "Feistel function" is equal to 0 then the structure is not invertible. This issue is solved in a different way in each round. During the first round, a different data-path is used in the case which should correspond to a multiplication by zero. In the second round, the "Feistel function" is not bijective and, in particular, has no pre-image for 0.

Our decomposition also explains the pattern in the LAT[3] of $\pi$ and $\pi'$ that we used in Section 13.2.1 (p. 248) to partially recover the linear layers permutations $\alpha$ and $\omega$. This pattern is made of two parts: the white square appearing at the top-left of $\mathcal{L}'_\pi$ and the "stripe" covering the 16 left-most columns of this table (see Figure 13.3). While the white-square is caused by an integral pattern, the reason behind the stripe is more complicated.

**On the Stripe** The "stripe" is explained by the following lemma which gives a closed formula expressing those biases as a function of the biases in the LAT of $v_0$ and $v_1$.

**Lemma 13.2.1.** *Biases in the stripe correspond to approximations $(a_L||a_R \rightsquigarrow b_L||0)$ in $\hat{\pi}$, where $b_L > 0$. The expression of $\mathcal{L}[a_L||a_R, b_L||0]$ is*

$$\mathcal{L}[a_L||a_R, b_L||0] = \mathcal{L}_0[a_L, b_L] + 8 \times \left((-1)^{b_L \cdot y_0} - \hat{\delta}(b_L)\right),$$

*where $\mathcal{L}_0$ is the LAT of $v_0$, $y_0$ depends on $a_R$, $a_L$ and the LAT of $v_1$, and $\hat{\delta}(b_L)$ is equal to 1 if $b_L = 0$ and to 0 otherwise.*

*Proof.* Biases in the stripe correspond to approximations $(a_L||a_R \rightsquigarrow b_L||0)$ in the linear layer-less version of $\pi$, which is denoted $\hat{\pi}$. These approximations are equal to:

$$2 \times \mathcal{L}[a_L||a_R, b_L||0] = \sum_{x \in \mathbb{F}_2^n} (-1)^{(a_L||a_R)\cdot x \,\oplus\, (b_L||0)\cdot \hat{\pi}(x)},$$

which we decompose by splitting $x \in \mathbb{F}_2^8$ into $(\ell, r) \in (\mathbb{F}_2^4)^2$ to obtain

$$\sum_{r \in \mathbb{F}_2^4, r \neq 0} \sum_{\ell \in \mathbb{F}_2^4} (-1)^{a_L \cdot \ell \,\oplus\, a_R \cdot r \,\oplus\, b_L \cdot v_1(\ell \odot \mathcal{I}(r))} + \sum_{\ell \in \mathbb{F}_2^4} (-1)^{a_L \cdot \ell \,\oplus\, b_L \cdot v_0(\ell)}. \tag{13.5}$$

The second term in this sum is equal to $2 \times \mathcal{L}_0[a_L, b_L]$ where $\mathcal{L}_0$ is the LAT of $v_0$. The first term can be simplified using the change of variable $u = v_1(\ell \odot \mathcal{I}(r))$, i.e. $\ell = v_1^{-1}(u) \odot r$:

$$\sum_{\ell \in \mathbb{F}_2^4} (-1)^{a_L \cdot \ell \,\oplus\, b_L \cdot v_1(\ell \odot \mathcal{I}(R))} = \sum_{u \in \mathbb{F}_2^4} (-1)^{b_L \cdot u \,\oplus\, a_L \cdot (v_1^{-1}(u) \odot r)}.$$

As a consequence, the first term of Equation (13.5) can be re-written:

$$\sum_{r \in \mathbb{F}_2^4, r \neq 0} \sum_{u \in \mathbb{F}_2^4} (-1)^{a_R \cdot r \,\oplus\, b_L \cdot u \,\oplus\, a_L \cdot (v_1^{-1}(u) \odot r)}$$

$$= \sum_{u \in \mathbb{F}_2^4} (-1)^{b_L \cdot u} \left( \sum_{r \in \mathbb{F}_2^4} (-1)^{a_R \cdot r \,\oplus\, a_L \cdot (v_1^{-1}(u) \odot r)} - 1 \right).$$

[3] Note that the LAT of $\hat{\pi}$ is not exactly the same as $\mathcal{L}'_\pi$ which is given in Figure 13.3 because of a nibble swap.

First, we note that $\sum_{u=0}^{16}(-1)^{b_L \cdot u}$ is equal to 0 if $b_L \neq 0$ and 16 otherwise. Then, we remark that $\sum_{r \in \mathbb{F}_2^4}(-1)^{a_R \cdot r \, \oplus \, a_L \cdot (v_1^{-1}(u) \odot r)}$ is equal to $2 \times \mathcal{L}^m_{v_1^{-1}(u)}[a_R, a_L]$, where $\mathcal{L}^m_k$ is the LAT of the Boolean linear permutation $r \mapsto r \odot k$. If we further replace $u$ by $y = v_1^{-1}(u)$ then Equation (13.5) can be re-written

$$2 \times \sum_{y \in \mathbb{F}_2^4} (-1)^{b_L \cdot v_1(y)} \mathcal{L}^m_y[a_R, a_L] - 16 \times \hat{\delta}(b_L).$$

where $\hat{\delta}(b_L) = 1$ if and only if $b_L = 0$. Besides, since $r \mapsto r \odot k$ is a linear function for all $k \neq 0$ it holds that for every pair $(a_R, a_L)$ with $a_R > 0$ and $a_L > 0$, there is exactly one value $y_0$ such that $\mathcal{L}^m_{y_0}[a_R, a_L] = 8$ and $\mathcal{L}^m_y[a_R, a_L] = 0$ for $y \neq y_0$.

We deduce that the expression of $\mathcal{L}[a_L || a_R, b_L || 0]$ is indeed

$$\mathcal{L}[a_L || a_R, b_L || 0] = \mathcal{L}_0[a_L, b_L] + 8 \times \left( (-1)^{b_L \cdot y_0} - \hat{\delta}(b_L) \right).$$

□

We deduce from Lemma 13.2.1 that, very roughly, $v_1$ is responsible for the sign of the biases in the stripe and $v_0$ for their values. Since the minimum and maximum biases in $\mathcal{L}_0$ are $-4$ and $+4$, the absolute value of $\mathcal{L}[a_L || a_R, b_L || 0]$ is indeed in $\{4, 6, 8, 10, 12\}$. As we deduce from our computation of these biases, the stripe is caused by the conjunction of three elements:

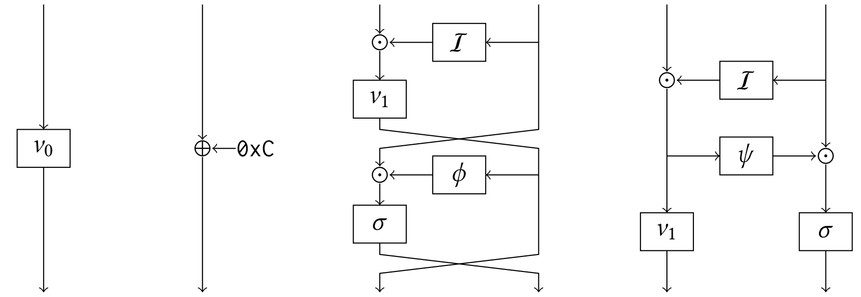
- the use of a multiplexer,
- the use of finite field inversion, and
- the fact that $v_0$ has good non-linearity.

Ironically, the only "unsurprising" sub-component of $\pi$, namely the inverse function, is one of the reasons why we were able to reverse-engineer this S-Box in the first place. Had $\mathcal{I}$ been replaced by a different (and possibly weaker!) S-Box, there would not have been any observable lines in the LAT, the very pattern which got our reverse-engineering started.

However, detecting linear subspaces of zeroes in the LAT of $\pi$ would have worked because the white-square does not depend on the use of the inverse function. Furthermore, while not visible with the naked eye, the lower number of values in the stripe would still imply a drop in the variance of the absolute value of the coefficients in it. Thus, the technique previously described in Section 9.1.2.4 (p. 163) would still work as well.

**Alternative Representation** Because of the multiplexer, we can deduce an alternative representation of $\hat{\pi}$. If the right nibble of the input is not equal to 0 then $\hat{\pi}$ can be represented using a Feistel-like structure as shown in Figure 13.11b. Otherwise, it is essentially equivalent to one call to the 4-bit S-Box $v_0$, as shown in Figure 13.11a. We also have some freedom in the placement of the branch bearing $\phi$. Indeed, as shown in Figure 13.11c, we can move it before the call to $v_1$ provided we replace $\phi$ by $\psi = \phi \circ v_1$.

Moreover, the decomposition we found is not unique. In fact, we can create many equivalent decompositions by e.g. adding multiplication and division by constants around the two finite field multiplications. We can also change the finite field in which the operations are made at the cost of appropriate linear isomorphisms modifying the 4-bit S-Boxes and the whitening linear layers. The presented decomposition is the most structured that we have found.



(a) $\hat{\pi}$ when $r = 0$. (b) $\hat{\pi}$ when $r \neq 0$. (c) $\hat{\pi}$ when $r \neq 0$.

Figure 13.11: Alternative representations of $\hat{\pi}$ where $\pi = \omega \circ \hat{\pi} \circ \alpha$.

#### 13.2.3.3 Hardware Implementation

It is not uncommon for cryptographers to build an S-Box from smaller ones, typically an 8-bit S-Box from several 4-bit S-Boxes, as described in Section 8.3.2 (p. 152). A common goal is to allow an efficient implementation of the S-Box in hardware or using a bit-sliced approach. Another possible reason behind such a choice is given by the designers e.g. of Clefia: it is to prevent attacks based on the algebraic properties of the S-Box, especially if it is based on the inverse in $\mathbb{F}_{2^8}$ like in the Aes. This would be coherent with the information gathered by Saarinen.

As stated before, hardware optimization was supposed to be one of the design criteria used by the designers of $\pi$. Thus, it is reasonable to assume that one of the aims of the decomposition we found was to decrease the hardware footprint of the S-Box.

To test this hypothesis, we simulated the implementation of $\pi$ in hardware.[4] We used four different definitions of $\pi$: the look up table given by the designers, our decomposition, a tweaked decomposition where the multiplexer is moved lower[5] and, finally, the alternative decomposition presented in Figure 13.11c. Table 13.7 contains both the area taken by our implementations and the delay, i.e. the time taken to compute the output of the S-Box. For both quantities, the lower is the better. As we can see, the area is divided by up to 2.5 and the delay by 8, meaning that an implementer knowing the decomposition has a significant advantage over one that does not.

## 13.3 Exponential Decompositions

In the S-Box of BelT, the low-variance rows in the LAT identified in Section 9.1.2.4 (p. 163) are related to its being a finite field pseudo-exponentiation. Given that the

[4] We used Synopsys `design_compiler` (version J-2014.09-SP2) along with digital library `SAED_EDK90_CORE` (version 1.11). I thank my colleague Yann Le Corre for performing those experiments

[5] More precisely, the multiplexer is moved after the left side is input to $\phi$. This does not change the output: when the output of $\nu_0$ is selected, the right branch is equal to 0 and the input of $\sigma$ is thus 0 regardless of the left side.

| Structure | Area ($\mu m^2$) | Delay (ns) |
|---|---|---|
| Naïve implementation | 3889.6 | 362.52 |
| Feistel-like (similar to Fig. 13.11b) | 1534.7 | 61.53 |
| Multiplications-first (similar to Fig. 13.11c) | 1530.3 | 54.01 |
| Feistel-like (with tweaked MUX) | 1530.1 | 46.11 |

Table 13.7: Results on the hardware implementation of $\pi$.

lines observed in the Pollock representation of the LAT of $\pi$ correspond in fact to low variance columns, it is tempting to investigate a possible relation between $\pi$ and pseudo-exponential functions. Another element links $\pi$ to finite field exponentiation: the 4-bit S-Box $\nu_0$ from the decomposition obtained in Section 13.2.2.3 (p. 254) is affine-equivalent to a finite field logarithm in $\mathbb{F}_{2^4}$.

In this section, we explore this relation further. We first formally identify patterns linking $\pi$ to finite field exponentiations in Section 13.3.1 (p. 260). We deduce two different decompositions, one based on an exponentiation in Section 13.3.1 (p. 260) and one based on a pseudo-exponentiation in Section 13.3.2 (p. 263). Finally, we discuss the consequences of these decompositions in Section 14.2 (p. 274).

### 13.3.1 Finding exponential patterns in $\pi^{-1}$

For any exponential function $x \mapsto g^x$, it holds that $g^{x \boxplus c} = g^x \odot g^c$ where $\boxplus$ denotes addition modulo 255. This property can be used to check if a function is based on an exponentiatial. Unfortunately, if the function is composed with some whitening linear layers then this property does not hold anymore. We can still use it though: $x \boxplus 2^i = x \oplus 2^i$ with probability $1/2$ and "$\oplus$" is linear in the same field as the whitening layer, meaning that values $2^i$ are simply mapped by the whitening layer to other fixed values.

We made an exhaustive search and found four high probability relations: for $c \in \{12, 26, 24, 30\}$, one of the following relations holds for all but 16 values of $x$:

$$\pi^{-1}(x \oplus c_i) = \pi^{-1}(x) \odot w^{2^i}, \text{ or}$$
$$\pi^{-1}(x \oplus c_i) = \pi^{-1}(x)/w^{2^i},$$

where $i$ runs over $\{0,1,2,3\}$, multiplication and division are done in the finite field $\mathbb{F}_2[X]/(X^8+X^4+X^3+X^2+1)$ and $w$ is the primitive element defined by $X$. For other polynomials, the relations hold with much lower probabilities.

Interestingly, the linear layer of the block cipher Kuznyechik uses multiplications in a finite field defined by another irreducible polynomial. It is therefore hard to see whether there is a special relation or interaction between the S-Box and the linear layer of the block cipher. The linear layer of the inner block cipher of the hash function Streebog has also been reverse-engineered [KK13] and, much like in Kuznyechik, uses finite field arithmetic over $\mathbb{F}_2^8$ defined by the irreducible polynomial $X^8+X^6+X^5+X^4+1$. To simplify further analysis, we consider the permutation $\tau = \log_{w,0} \circ \pi^{-1}$.

If $\pi$ is indeed based on an exponentiation, the constants $c_i$ should be mapped by the linear whitening layer to some powers of 2. We therefore assume that an unknown linear layer maps $c_i$ to $2^i$ for $i \in \{0,1,2,3\}$.

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

Table 13.8: The look-up table of $\tau^{-1} \circ \alpha^{-1}$.

We complete the mapping by setting preimages for powers $2^j$ for $j \in \{4,5,6,7\}$. First, we complete it randomly and get some linear mapping $\alpha'$ (see Equation 13.6). We then observe that the look-up table of $\tau \circ \alpha'^{-1}$ given in Table 13.8 is very highly structured. As with the mini-block cipher $T$ decomposed in Section 13.2.2.1 (p. 251), we can XOR a value which depends on the column index to the right branch before the permutation in such a way that the entries in each rows are sorted in increasing order — except in the row `E.` where 0 is misplaced. This function is linear so it can be included into the linear layer. The matrix of the resulting linear layer $\beta$ is given in Equation 13.6 and the look-up table of $\tau \circ \beta^{-1}$ is given in Table 13.9.

$$\alpha' = \begin{bmatrix} 1&0&0&0&0&0&0&0\\ 0&1&0&0&0&0&0&0\\ 0&0&0&0&1&1&1&0\\ 0&0&0&0&1&0&0&1\\ 0&0&1&0&0&0&0&0\\ 0&0&0&0&0&1&1&0\\ 0&0&0&0&0&0&1&1\\ 0&0&0&1&0&0&0&0 \end{bmatrix}^{-1}, \quad \beta = \begin{bmatrix} 1&0&0&0&0&0&0&0\\ 0&1&0&0&0&0&0&0\\ 0&0&0&0&1&1&1&0\\ 1&0&0&0&1&0&0&1\\ 0&0&1&0&0&0&0&0\\ 0&1&0&0&0&1&1&0\\ 1&0&1&0&0&0&1&1\\ 0&0&0&1&0&0&0&0 \end{bmatrix}^{-1}. \tag{13.6}$$

Let

$$q = [12, 2, 9, 10, 13, 6, 3, 5, 11, 4, 8, 15, 14, 7, 0, 1],$$

and let $q_L(x||y) = q(x) \mid\mid y$ be a permutation of $\mathbb{F}_2^4 \times \mathbb{F}_2^4$. We reorder the rows by composing the function with $q_L$, that is, by applying a 4-bit S-Box on the left input branch of $\tau \circ \beta^{-1}$. The look-up table of $\tau \circ \beta^{-1} \circ q_L^{-1}$ is given in Table 13.10. As we can see, this permutation is almost the identity. We then deduce a complete decomposition of the inverse of $\pi$ which is presented in Algorithm 13.2.

Oddly, $q$ is affine-equivalent to one of the S-Boxes used in the GOST R 34.11-94 hash function, namely the S-Box "pi[1]" specified in RFC 5831 [Dol10b]. This hash function is the predecessor of Streebog; it was designed by the Russian Federal Agency of Government Communications and Information, an institution which was later incorporated into the FSB. However, there are only 302 affine-equivalence classes of 4-bit S-Boxes (see Table 5 of [BDBP03]), meaning that a mere coincidence cannot be ruled out.

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

Table 13.9: The look-up table of $\log_{\lambda,0} \circ \pi^{-1} \circ \beta^{-1}$.

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

Table 13.10: The look-up table of $\log_{\lambda,0} \circ \pi^{-1} \circ \beta^{-1} \circ q_L^{-1}$.

**Algorithm 13.2** $\pi^{-1}$ evaluation;
**Input:** 8-bit block $x$ ;
**Output:** 8-bit block $y$

```
$\ell \,||\, r \leftarrow \beta(x)$
$\ell \leftarrow q(\ell)$
if $\ell = 0$ then
    $z \leftarrow 17 \times ((r + 1) \mod 16)$
else
    $z \leftarrow 17 \times \ell + r - 16$
end if
$y \leftarrow \exp_{w,0}(z)$
return $y$
```

### 13.3.2 Another Decomposition of $\pi$

The somewhat heavy arithmetic performed in the middle of the computation of $\pi$ using our new decomposition could point towards the idea that $\pi$ is built using a pseudo-exponential rather than an exponential substitution. Intuitively, these operations could correspond to a "correction" of the offset.

To explore this idea, we brute-forced all possible preimages $z$ for 0 and all possible bases $v$ for the exponentiation. For each such pseudo-exponential substitution, we ran the following steps.

1. Compute[6] $s = \pi \circ \exp_{v,z}$.
2. Compute the differential uniformity and the maximum LAT coefficient of $s$.
3. If these quantities are high enough then look for vector spaces $\mathcal{V}_0$ and $\mathcal{V}_1$ such that, for all $(a, b)$ in $\mathcal{V}_0 \times \mathcal{V}_1$, $\text{LAT}[a, b] = 0$.

We restrict ourselves to cases where the maximum LAT coefficient or differential uniformity of $s$ is high. Indeed, this restriction ensures that the functions we study are "weak", which would be the case if it consisted only of few simple operations. In fact, we found that for appropriate choices of $v$ and $z$, the differential uniformity and the maximum LAT coefficient are respectively above 120 and 80, while typical values for a random permutation are expected to be in the vicinity of 12 and 32 respectively as was seen in Table 9.1 (p. 161).

The idea behind the search for vector spaces of zeroes is of course to identify possible linear layers to apply before and/or after $s$ in such a way that the resulting permutation has a TU-decomposition. To do so, we used a variant of the algorithm described in [BPU16] to recover part of the linear layer used to whiten a 4-round Feistel Network by reconstructing the pattern described by Lemma 12.2.2 (p. 233).

Among the pairs $(z, v)$ such that $\mathcal{V}_0$ and $\mathcal{V}_1$ were large enough, some were such that

$$\begin{cases} \mathcal{V}_0 = \{\texttt{00},\texttt{01},\texttt{02},\texttt{03},\texttt{04},\texttt{05},\texttt{06},\texttt{07},\texttt{08},\texttt{09},\texttt{0a},\texttt{0b},\texttt{0c},\texttt{0d},\texttt{0e},\texttt{0f}\} \\ \mathcal{V}_1 = \{\texttt{00},\texttt{1a},\texttt{20},\texttt{3a},\texttt{44},\texttt{5e},\texttt{64},\texttt{7e},\texttt{8a},\texttt{90},\texttt{aa},\texttt{b0},\texttt{ce},\texttt{d4},\texttt{ee},\texttt{f4}\}. \end{cases}$$

Interestingly, $\mathcal{V}_1 = \mathcal{V}$, the vector space corresponding to the indices of the columns identified in the Jackson Pollock representation of the LAT of $\pi$ in Section 13.2.1 (p. 248). The inverse of the final linear layer of the decomposition of Section 13.2 (p. 248), which was denoted $\omega$, can therefore be composed with permutations $s$ obtained in this way to derive TU-decompositions of these.

Furthermore, it is possible to find a $4 \times 4$ linear layer $L_u$ such that $x \mapsto (L_u \circ U_k)(x) = x \boxplus q'(k)$ for some 4-bit function $q'$. However, the only pair $(v, z)$ for which $q'$ is a permutation is $(v = w, z = 16)$, in which case $q'$ has the following look-up table:

$$q' = [4, 14, 5, 12, 3, 10, 2, 9, 8, 1, 7, 6, 15, 0, 13, 11].$$

Therefore, if we compose $s$ with the inverse of $L_u$ on the right side and the inverse of $q'$ on the left one, we obtain a new permutation whose TU-decomposition is such

[6] We did consider that $\pi$ might consist of a pseudo-logarithm *preceded* by another permutation, unlike here where we consider that it is *followed* by one. This line of investigation did not lead to any decomposition.

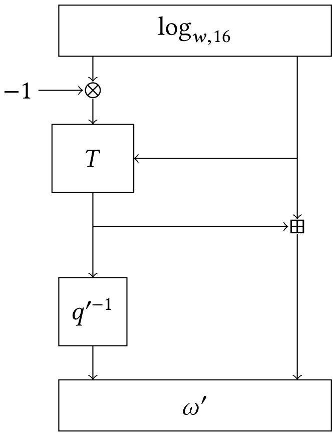
that $U_k(x) = x \boxplus k$, which is quite simple. Unfortunately, $T$ remains a strange object:

$$T_k((-1) \times x) = \begin{cases} 15, \text{ if } \overline{x} + \overline{k} = 16, \\ x \boxminus 1, \text{ if } \overline{x} + \overline{k} > 16, \\ x, \text{ otherwise,} \end{cases}$$

where + denotes addition without a modulo and where $x \times y = \overline{x} \times \overline{y} \mod 16$. The full codebook of $T$ is provided in Figure 13.12b. As we can see, each permutation $T_k$ is obtained from the identity by inserting 15 at position $17 - \overline{k}$ (except for $T_0$), much like 0 is inserted at an arbitrary position in a pseudo-exponential substitution. The complete decomposition is summarized in Algorithm 13.3 and Figure 13.12a. The arithmetic layer of Algorithm 13.3 is quite sparse: $U_k$ is merely a modular addition and $T_k$ is simply derived from the identity. In fact, this arithmetic layer has a differential with probability 1/2.

**Algorithm 13.3** Evaluation of $\pi$;
**Input:** 8-bit block $x$;
**Output:** 8-bit block $y$.

```
$(l||r) \leftarrow \log_{w,16}(x)$
$l \leftarrow (-l) \mod 16$
if $l + r = 16$ then
    $l \leftarrow 15$
else if $l + r > 16$ then
    $l \leftarrow (l - 1) \mod 16$
end if
$r \leftarrow (l + r) \mod 16$
$l \leftarrow q'^{-1}(l)$
$y \leftarrow \omega'(l||r)$
return $y$
```



(a) High level view.

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | a | b | c | d | e | f |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $T_0$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | a | b | c | d | e | f |
| $T_1$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | a | b | c | d | e | f |
| $T_2$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | a | b | c | d | f | e |
| $T_3$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | a | b | c | f | d | e |
| $T_4$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | a | b | f | c | d | e |
| $T_5$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | a | f | b | c | d | e |
| $T_6$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | f | a | b | c | d | e |
| $T_7$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | f | 9 | a | b | c | d | e |
| $T_8$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | f | 8 | 9 | a | b | c | d | e |
| $T_9$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | f | 7 | 8 | 9 | a | b | c | d | e |
| $T_a$ | 0 | 1 | 2 | 3 | 4 | 5 | f | 6 | 7 | 8 | 9 | a | b | c | d | e |
| $T_b$ | 0 | 1 | 2 | 3 | 4 | f | 5 | 6 | 7 | 8 | 9 | a | b | c | d | e |
| $T_c$ | 0 | 1 | 2 | 3 | f | 4 | 5 | 6 | 7 | 8 | 9 | a | b | c | d | e |
| $T_d$ | 0 | 1 | 2 | f | 3 | 4 | 5 | 6 | 7 | 8 | 9 | a | b | c | d | e |
| $T_e$ | 0 | 1 | f | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | a | b | c | d | e |
| $T_f$ | 0 | f | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | a | b | c | d | e |

(b) The code-book of the block-cipher $T$.

Figure 13.12: Another decomposition of $\pi$.

## 13.4 A Discussion of Our Decompositions

The exponential-based decompositions have less “information-heavy” elements than the Feistel-based one — one 4-bit S-Box and one linear layer instead of four and two respectively.

The last decompositions contain a central layer consisting of several arithmetic operations whose purpose is not clear. Oddly enough, the finite field logarithms or exponentials do not have TU-decompositions, but one appears when the arithmetic layer is added. It is therefore possible that these were added so as to build S-Boxes with better hardware implementations, as discussed in Section 13.2.3.3 (p. 259).

The relationship between our decompositions is anything but obvious. While the final linear layers are related, the remainder is utterly different: two 4-bit finite field multiplications, 5 random-looking 4-bit S-Boxes and a multiplexer turn out to be functionally equivalent to a pseudo logarithm followed by modular arithmetic and a 4-bit S-Box.

The relation between the two representations would have been clearer if the modular arithmetic layer had been placed *before* the pseudo-logarithm. In this case the finite field multiplications would have been performed with logarithm tables. It is not the case: removing the final linear layer exposes directly a modular addition in the new decompositions.

We also note that the function $\pi^{-1} \circ \log_{w,16}$ is extremely weak from a cryptographic perspective. Its coordinates have an algebraic degree as low as 3, it is differentially 128-uniform and its highest linear bias is $96/128$. We deduce that, in some sense, $\pi$ is “close” to $\log_{w,16}$. A permutation picked uniformly should yield a random looking S-Box when composed with $\log_{w,16}$, not a differentially 128-uniform. The probability that a random 8-bit permutation is at least differentially 128-uniform can be estimated[7] using Theorem 9.1.1 (p. 160). Under the conditions of Theorem 9.1.1, probability that a random permutation has a differential uniformity strictly less than 128 is given by:

$$\Pr[\Delta_s < 128] \;=\; \left(1 - \sum_{k=128}^{256} \frac{\exp(-1)}{k!}\right)^{255\times 255} \;\approx\; 1 - \frac{(255^2)}{e} \sum_{k=128}^{256} \frac{1}{k!} \;\approx\; 1 - 2^{-346}\,.$$

In other words, the probability that a random permutation composed with $\log_{w,16}$ is differentially 128-uniform is extremely low. The only way such an event can occur is if $\pi$ indeed has a structure somewhat related to $\log_{w,16}$.

These results show that the algebraic structure, whose presence was known thanks to the first decomposition, is stronger and more mathematical in nature.

The permutation $\pi$ may have been built using one of the decompositions presented in this section. However, we think it more likely that each of these decompositions is a consequence of another still secret algebraic structure, one related to a finite field exponential. Still, this “master decomposition” remains elusive. Unfortunately, unless the Russian secret service release their design strategy,[8] their exact

[7] Theorem 9.1.1 (p. 160) relies on the assumption that the DDT coefficients are independent. This approximation is likely to interfere with our computation: since the sum of the coefficients in each column is equal to 256, having one greater than or equal to 128 will skew the distribution of all the other coefficients in the same row and column. Still, this estimation of $\Pr[\Delta_s < 128]$ shows that this probability is negligible.

[8] Alas, such a release seem unlikely. The designers of these algorithms did release the design criteria for the round constants of Streebog [Rud15] after it was shown that those could be chosen malevolently [AY15b]. However, when asked by a cryptographer about their S-Box design in light of the work

process is likely to remain a mystery, if nothing else because of the existence of alternative decompositions. Which exists by design and which is a mere side-effect of this design?

presented in this chapter, they insisted that they did generate $\pi$ pseudo-randomly and that our first decomposition constitutes a “discovery” which is welcome as it simplifies the hardware implementation of the S-Box. This claim is coherent with Saarinen’s initial recollection of his conversations with the designers. I remain nevertheless extremely skeptical about the groundedness of their claims.

Chapter 14

# Decomposing the Only Known APN Permutation on $\mathbb{F}_{2^{2n}}$

The techniques intended for S-Box reverse-engineering and other structural attacks described in the previous chapters can be applied to other aims. Indeed, as long as the full look-up table of a function is known, those same methods can be applied to try and decompose this function.

We have applied this idea with great success to the only known solution to the APN problem. The coefficients in the DDT of an S-Box are always even and, in order to ensure resilience against differential attacks, the lower their maximum the better. An optimal function in this context is one which only has 0 and 2 in its DDT — except when the input difference is equal to 0. Such a function is called Almost Perfect Nonlinear (APN). It is easy to find APN permutations in $\mathbb{F}_{2^n}$ when $n$ is odd but the very existence of such objects for $n$ even is an open problem. The most recent advance on this topic is a paper by a team of mathematicians from the NSA [BDMW10] who exhibited the first APN permutation in $\mathbb{F}_{2^6}$, $S_0$. Unfortunately, they could not generalize their construction to higher dimensions and stated the still open "big APN problem".

**Open Problem 14.0.1** (Big APN Problem)**.** *Is it possible to find an APN permutation of $\mathbb{F}_{2^n}$ for $n$ even and $n \geq 8$?*

In this chapter, I first present the decomposition of $S_0$ Aleksei Udovenko and I derived. It is a particular case of the *open butterfly* construction which is recovered in Section 14.1 (p. 268). A first analysis of this structure is presented in Section 14.2 (p. 274). Our decomposition implies the existence of efficient hardware and bit-sliced implementations of $S_0$ which are discussed in Section 14.3 (p. 277). Using these results, Anne Canteaut, Sébastien Duval and I developed a more advanced generalization of the butterfly structure. The properties of such mappings are explored in Section 14.4 (p. 280). Unfortunately, we show that Dillon's permutation is the only APN generalized butterfly, which therefore cannot be solutions of the big APN problem.

In what follows, the field trace of an element $x$ of $\mathbb{F}_{2^n}$ is:

$$\mathrm{Tr}(x) = \sum_{i=0}^{n-1} x^{2^i} .$$

## 14.1 A Decomposition of the 6-bit APN Permutation

In this section, we identify a decomposition of the Dillon APN permutation. We denote this permutation $S_0 : \mathbb{F}_2^6 \to \mathbb{F}_2^6$ and give its look-up table in Table 14.1. As we are interested only in its being an APN permutation, we allow ourselves to compose it with affine permutations as such transformations preserve this property. We will omit the respective inverse permutations to simplify our description.

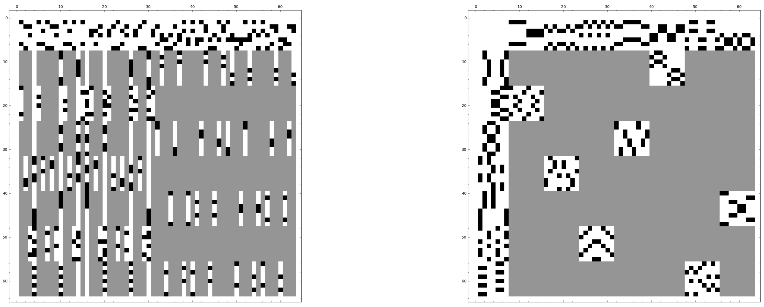
| | .0 | .1 | .2 | .3 | .4 | .5 | .6 | .7 | .8 | .9 | .a | .b | .c | .d | .e | .f |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0. | 00 | 36 | 30 | 0d | 0f | 12 | 35 | 23 | 19 | 3f | 2d | 34 | 03 | 14 | 29 | 21 |
| 1. | 3b | 24 | 02 | 22 | 0a | 08 | 39 | 25 | 3c | 13 | 2a | 0e | 32 | 1a | 3a | 18 |
| 2. | 27 | 1b | 15 | 11 | 10 | 1d | 01 | 3e | 2f | 28 | 33 | 38 | 07 | 2b | 2c | 26 |
| 3. | 1f | 0b | 04 | 1c | 3d | 2e | 05 | 31 | 09 | 06 | 17 | 20 | 1e | 0c | 37 | 16 |

Table 14.1: The Dillon permutation $S_0$ in hexadecimal (e.g. $S_0$(0x10) = 0x3b).

As in Chapter 13, our strategy relies on the TU-decomposition of Theorem 1. Section 14.1.1 presents this first decomposition. Then, we decompose the corresponding mini-block ciphers in Section 14.1.2. Finally, we provide the complete decomposition of an S-Box affine-equivalent to $S_0$ in Section 14.1.3.

### 14.1.1 High-Level TU-Decomposition

The "Jackson Pollock" representation of the absolute value of the LAT of $S_0$ is given in Figure 14.1a. We can see some patterns, namely columns and aligned short vertical segments of black and white colors within a grey rectangle, where white is 0, grey is 4 and black is 8. The black-and-white columns also have the 8 topmost coefficients equal to zero. Moreover, their horizontal coordinates form a linear subspace of $\mathbb{F}_2^6$.



(a) LAT of $S_0$.

(b) LAT of $\eta^t \circ S_0$.

Figure 14.1: The Jackson Pollock representation of the LAT of two permutations (absolute value).

We compose the S-Box with a linear permutation mapping these special columns to the leftmost columns of the picture. The black-and-white columns have coordinates {0, 4, a, e, 10, 14, 1a, 1e}. Those form a linear subspace of $\mathbb{F}_2^6$ of dimension 3 spanned by the binary expansions of 4, a and 10.

We therefore build the linear permutation $\eta \mathbb{F}_2^6 \to \mathbb{F}_2^6$ so that

$$\eta(1) = 4,\ \eta(2) = \mathtt{a},\ \eta(4) = 10$$

and, in order to fully define it, we complete it by setting

$$\eta(8) = 1,\ \eta(10) = 2,\ \eta(20) = 20\ .$$

Using Lemma 8.2.4, the composition $\eta^t \circ S_0$ has an LAT which groups the black-and-white columns to its left, as intended. It is indeed the case: this pattern is visible in the Pollock representation of $\eta^t \circ S_0$ in Figure 14.1b.

The "grey-less" columns have indeed been grouped, as was our goal, but additional patterns have emerged. The short black-and-white segments became small squares, thus making the whole picture more structured. There is also a "white-square", typical of the structures described by Lemma 12.3.1 in the top-left corner.

By applying this Lemma, we deduce a TU-decomposition of the original permutation based on mini-block ciphers $T$ and $U$ as well as the linear layer $\eta^t$. The corresponding mini-block ciphers $T$ and $U$ are given in Table 14.2.

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

(a) $T$.

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

(b) $U$.

Table 14.2: The keyed permutations $T$ and $U$. $T_k$ and $U_k$ denote the permutations corresponding to the key $k$.

Using the algorithm by Biryukov *et al.* from [BDBP03], we found that the permutations $T_k^{-1}$ and $U_k$ are linearly equivalent. This linear equivalence is given by:

$$\mathcal{U}(x) = M'_U \circ \mathcal{T}^{-1} \circ M_U(x),$$

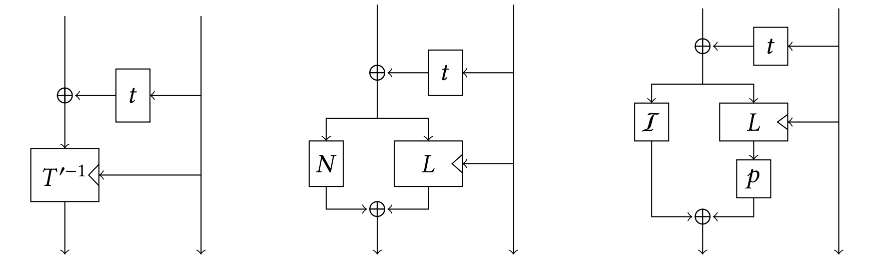
where $\mathcal{T} : x||y \mapsto x||T(x,y)$ and $\mathcal{U} : x||y \mapsto x||U(x||y)$ are 6-bit permutations and the linear permutations $M_U$ and $M'_U$ correspond to the following binary matrices:

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

### 14.1.2 Decomposing $T$

Since $T$ and $U$ are related, we first decompose $T$ and then will deduce a decomposition of $U$ from it.

As was done for the GOST S-Box in Section 13.2.2.1 (p. 251), we compose $T_k$ with a Feistel round to ensure that 0 is mapped to itself for all keys. If we apply such an appropriate Feistel round before or after $T^{-1}$, the corresponding Feistel function is always a permutation. This Feistel function is linear when used between $T$ and $U$ but non-linear when applied before. We thus consider the former case. We define



(a) Detaching a linear Feistel round.

(b) Splitting $T'^{-1}$ into $N$ and $L$.

(c) Simplifying $N$ into $\mathcal{I}$ and linear functions.

Figure 14.2: Simplifying the keyed permutation $T'^{-1}$.

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

Table 14.3: The values and polynomial interpolation of each $T_k'^{-1}$.

$t(k) = T_k(0)$ and $T_k'(x) = T_k(x) \oplus t(k)$ as illustrated in Figure 14.2a, so that $T_k'(0) = T_k'^{-1}(0) = 0$ for all $k$. The linear permutation $t$ is given by $t(x) = [0,7,4,3,1,6,5,2]$.

We then look for algebraic structures in $T'$. The irreducible polynomial $X^3+X+1$ is used to perform computations in $\mathbb{F}_{2^3}$. Let $w$ be a generator of the multiplicative subgroup of $\mathbb{F}_{2^3}$. We represent an element $x = x_0 \oplus x_1 w \oplus x_2 w^2$ of this field using an italic rendition of the integer $x_0 + 2x_1 + 4x_2$. For example, $w$ is represented as *2*. This representation is motivated by concision and convenience for working with Sage [Dev16] where field elements can be generated from integers in this fashion using the `F.fetch_int()` method of a finite field instance `F`. We only use this notation for this decomposition process.

The Lagrange interpolation of $T_k'^{-1}$ for all $k$ as polynomials over $\mathbb{F}_{2^3}$ is given in Table 14.3. Interestingly, the coefficients of the non-linear terms $x^6, x^5$ and $x^3$ are key-independent. We thus decompose $T'^{-1}$ as a sum of its non-linear part $N$ and its key-dependent linear part $L_k$ so that $T_k'^{-1}(x) = N(x) + L_k(x)$, where $N(x) = 3x^6 + 2x^5 + 5x^3$ and $L_k(x)$ is linear for any $k$. This process is illustrated in Figure 14.2b.

We now simplify $N$ by applying a linear permutation of our choice after $T'^{-1}$ (see Figure 14.2c). The output of $T'^{-1}$ corresponds to the input of the higher level structure, meaning that applying this linear function will result in an overall construction affine-equivalent to the original, and thus one which is also apn. Choosing this side also avoids a corresponding modification of $U$. We choose the linear permutation $p(x) = 4x^4 + x^2 + x$ because $(p \circ N) : x \mapsto x^6$ is the multiplicative inverse function in $\mathbb{F}_{2^3}$. We denote this inverse $\mathcal{I}$.

| Function | Polynomial |
|---|---|
| $p \circ L_0$ | $7x^4 + 3x^2$ |
| $p \circ L_1$ | $2x^4 + 4x^2$ |
| $p \circ L_2$ | $0x^4 + 0x^2$ |
| $p \circ L_3$ | $5x^4 + 7x^2$ |

| Function | Polynomial |
|---|---|
| $p \circ L_4$ | $4x^4 + 6x^2$ |
| $p \circ L_5$ | $1x^4 + 1x^2$ |
| $p \circ L_6$ | $3x^4 + 5x^2$ |
| $p \circ L_7$ | $6x^4 + 2x^2$ |

Table 14.4: The interpolation polynomials of each $p \circ L_k$.

The composition of $p$ and $L_k$, denoted, $(p \circ L_k)$, is also simpler than $L_k$. Its Lagrange interpolation only has nonzero coefficients for the terms in $x^2$ and in $x^4$, as can be see seen in Table 14.4. The composition of $p$ with $L_2$ is the constant function $(p \circ L_2) : x \mapsto 0$ so we add $2$ to $k$ to obtain these linear layers:

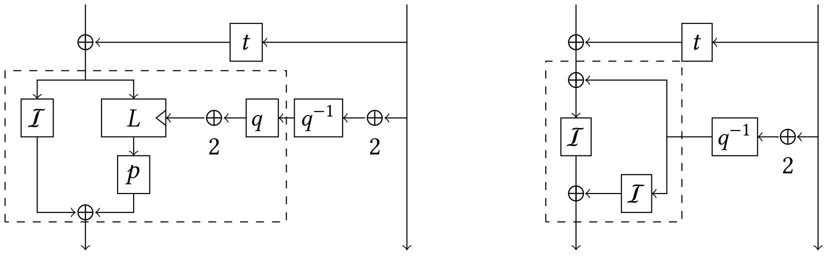
$$(p \circ L_k)(x) = l_2(k+2)x^2 + l_4(k+2)x^4,$$

where $l_2(x) = 2x^4 + 4x^2 + x$ and $l_4(x) = x^4 + 6x^2 + 2x$ are deduced from the Lagrange interpolations of $p \circ L_k$ from Table 14.4.

We can simplify this structure further. To this end, we looked for a linear permutation $q$ such that $l_2 \circ q$ and $l_4 \circ q$ have simpler forms. It turns out that if $q(x) = 3x^4 + 7x^2 + 3x$ then $(l_2 \circ q)(x) = x^4$ and $(l_4 \circ q)(x) = x^2$. Thus, we deduce that $(p \circ L_k)(x) = k'^4x^2 + k'^2x^4$, where $k' = q^{-1}(k+2)$.

In the end, we have obtained a representation of of $p \circ T'^{-1}$ which depends only on linear functions and the inverse function. It is described in Equation (14.1) and in Figure 14.3:

$$\begin{aligned}(p \circ T_k'^{-1})(x) &= x^6 + x^2k'^4 + x^4k'^2 \\ &= (x + k')^6 + k'^6, \text{ with } k' = q^{-1}(k+2)\ .\end{aligned} \tag{14.1}$$



(a) Using $k' = q(k) \oplus 2$.

(b) Using Equation (14.1).

Figure 14.3: Simplifying $p \circ L$ and thus $T'^{-1}$. The dashed area corresponds to the equivalence given by Equation 14.1.

Then, we replace the application of $x \mapsto q^{-1}(x+2)$ on the horizontal branch in Figure 14.3b by its application on the right vertical branch followed by its inverse (see Figure 14.4a; note that $q^{-1}(2) = 5$). By then discarding the affine permutation applied on the top of the right branch, we obtain the structure shown in Figure 14.4b. Finally, we merge the two linear Feistel functions $t$ and $q$ into $z : x \mapsto t(q(x)) \oplus x$ to obtain our final decomposition of $\mathcal{T}^{-1}$:

$$\mathcal{T}^{-1}(\ell||r) \ = \ \mathcal{I}\left(\ell + z\left(q^{-1}(r)\right) + 5\right) + \mathcal{I}\left(q^{-1}(r) + 5\right) \ || \ (q^{-1}(r) + 5),$$

which is also described in Figure 14.4c. In what follows, we use this decomposition of $T$ to express a complete permutation affine-equivalent to $S_0$.

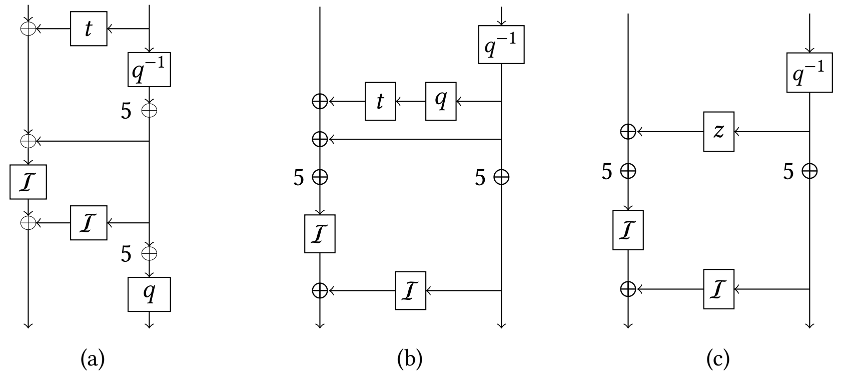


Figure 14.4: Finishing the decomposition of $T^{-1}$: moving $q$, $q^{-1}$ and $x \mapsto x+2$ around, removing the outer affine layer and merging the Feistel linear rounds.

### 14.1.3 Joining the decompositions of $T$ and $U$.

Let us now join the decomposition of $T$ and $U$ together, that of $U$ being obtained using that $U$ its affine-equivalence with $T$. The affine transformations applied on the top of $T'^{-1}$ make the relation between $T^{-1}$ and $U$ affine instead of linear on one side. This side corresponds to the output of the S-Box and we omit this transformation. The other linear mapping $M_U$ connecting $T^{-1}$ and $U$ merges with the linear part of $T^{-1}$ and its symmetric copy from $U$ into the linear mapping $M$, as illustrated in Figures 14.5a and 14.5b). The linear permutation $M$ has the following matrix representation:

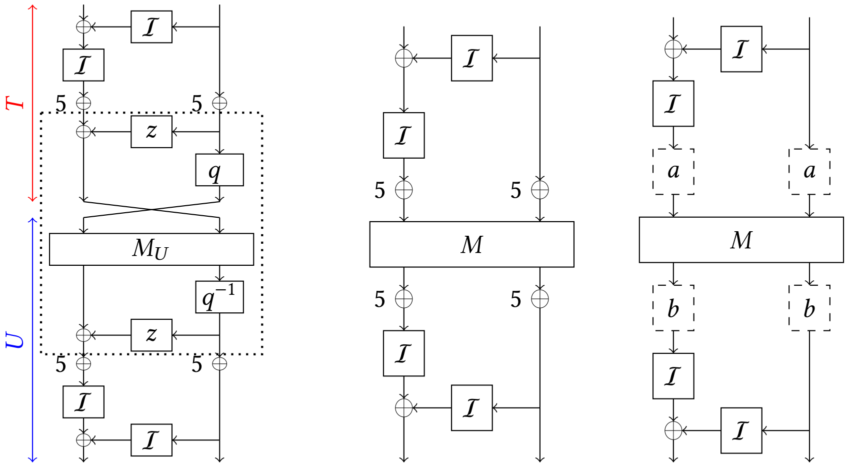
$$M = \begin{bmatrix} 1 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 \end{bmatrix} .$$

In order to further improve our decomposition, we studied how each component of this structure could be modified so as to preserve the APN property of the permutation. We investigated both the replacement of the linear and non-linear permutations used and described our findings in [PUB16]. In particular, we found that we could modify the central affine layer in the following fashions while still keeping the APN property of the permutation (see Theorem 2 of [PUB16]):

- changing the XOR constants to any value, and, in particular, setting them to 0;
- inserting two arbitrary 3-bit linear permutations $a$ and $b$ as shown in Figure 14.5c.

Thus, we remove the XORs from the structure and exhaustively check all linear permutations $a, b$ such that the resulting linear layer from Figure 14.5c has the simplest form. We found that for

$$\begin{cases} a(x) = 2x^4 + 2x^2 + 4x \\ b(x) = 2x^4 + 3x^2 + 2x \end{cases} ,$$



(a) Joining the decompositions of $T$ and $U$.

(b) Merging linear layers.

(c) Allowed transformations.

Figure 14.5: Simplifying the middle affine layer. The linear mappings in the dotted area in Figure 14.5a form the linear layer $M$.

the resulting linear permutation can be represented by the matrix $M'$ over $(\mathbb{F}_{2^3} \times \mathbb{F}_{2^3})$ defined as

$$M' = \begin{bmatrix} \mathit{2} & \mathit{5} \\ \mathit{1} & \mathit{2} \end{bmatrix} .$$

Because $M'$ is an involution and because of the symmetry of our decomposition, the whole S-Box is involutive too! The matrix $M'$ can further be decomposed into a 2-round Feistel Network with finite field multiplications by $\mathit{2}$ as Feistel functions. We deduce our final decomposition from this final observation and describe it in the following theorem.

**Main Theorem 2.** *There exist linear bijections $A$ and $B$ such that the* apn *6-bit permutation of Dillon et al. is equal to*

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

*where $S_{\mathcal{I}}(\ell||r)$ is the concatenation of two bivariate polynomials of $\mathbb{F}_{2^3}$ denoted $S^L_{\mathcal{I}}(\ell, r)$ and $S^R_{\mathcal{I}}(\ell, r)$ and which are equal to*

$$\begin{cases} S^R_{\mathcal{I}}(\ell||r) = (r^6 + \ell)^6 + \mathit{2} \odot r, \\ S^L_{\mathcal{I}}(\ell||r) = \left(r + \mathit{2} \odot S^R_{\mathcal{I}}(\ell, r)\right)^6 + \left(S^R_{\mathcal{I}}(\ell, r)\right)^6 . \end{cases}$$

*A picture representing a circuit computing $S_{\mathcal{I}}$ is given in Figure 14.6.*

For the sake of completeness, we give the matrices of the linear permutations $A$ and $B$:

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

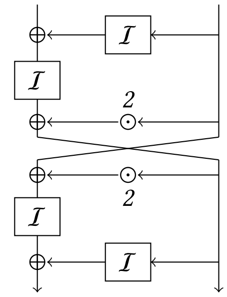


Figure 14.6: The APN involution $S_{\mathcal{I}}$, where $\mathcal{I}$ denotes the inverse in the finite field $\mathbb{F}_{2^3}$ i.e. the monomial $x \mapsto x^6$.

## 14.2 Analyzing Our Decomposition

In this section, we study the structure of the 6-bit APN permutation we derived from the Dillon permutation in the previous section. We start with a description of its cryptographic properties in Section 14.2.1. Then, we generalize this structure into the *Butterfly structure* (see Section 14.2.2). We also discover some new relations between the APN permutation, the Kim function and the cube mapping over $\mathbb{F}_{2^6}$ in Section 14.2.3.

### 14.2.1 Cryptographic Properties

The first consequence of our decomposition is the surprising observation that the 6-bit APN permutation is affine-equivalent to an involution. To the best of our knowledge, this was not known.

The permutation $S_{\mathcal{I}}$ is obviously APN due to its affine-equivalence with the permutation of Dillon *et al.*, so that the highest differential probability is equal to $2/64 = 2^{-5}$. The Pollock representation of the DDT of $\mathtt{Swap} \circ S_{\mathcal{I}} \circ \mathtt{Swap}$, where $\mathtt{Swap}$ denotes a swap of two 3-bit branches, is provided in Figure 14.7a. The LAT of $S_{\mathcal{I}}$ contains[1], in absolute value, only 3 different coefficients: 945 occurrences of 0, 2688 occurrences of 4 and 336 occurrences of 8. Its Pollock representation is in Figure 14.7b. The maximum linear bias is $8/32 = 2^{-2}$. The left half of its output bits have algebraic degree 4 and those on the right half have algebraic degree 3.

### 14.2.2 The Butterfly Structure

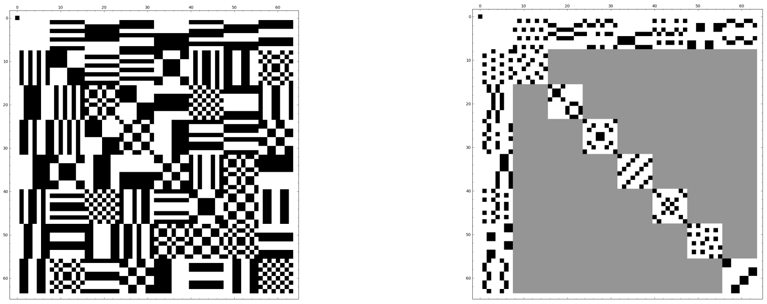
As described above, the output of our 6-bit APN permutation $S_{\mathcal{I}}$ is the concatenation of two bivariate polynomials of $\mathbb{F}_{2^3}$. We define the keyed permutation $R_k$ of $\mathbb{F}_{2^3}$ with a key in $\mathbb{F}_{2^3}$ as

$$R_k(x) = (x + 2 \odot k)^6 + k^6,$$

where $R_k$ is indeed a permutation affine equivalent to the inverse function $x \mapsto x^6$. In fact, its inverse $R_k^{-1}$ such that $R_k^{-1}(R_k(x)) = x$ is equal to $R_k^{-1} = (x + k^6)^6 + 2 \odot k$. Using this keyed permutation and its inverse, it is easy to express $S_{\mathcal{I}}$ :

$$S_{\mathcal{I}}(\ell || r) = R_{R_r^{-1}(\ell)}(r) \mid\mid R_r^{-1}(\ell) \ .$$

[1] As $S_{\mathcal{I}}$ is a permutation, we ignore the first line and the first column of its LAT.



(a) DDT of Swap $\circ$ $S_\mathcal{I}$ $\circ$ Swap (white: 0, black: 2).

(b) LAT of $S_\mathcal{I}$ (white: 0, grey: 4, black: 8).

Figure 14.7: The Pollock representation of the DDT and LAT of $S_\mathcal{I}$.

This structure is described in Figure 14.8a.

Using this representation, we show that $S_\mathcal{I}$ is CCZ-equivalent to a quadratic function with a very similar structure. First, we recall the definition of CCZ-equivalence, where CCZ stands for Carlet-Charpin-Zinoviev [CCZ98] as it is defined e.g. in [BN15].

**Definition 14.2.1** (CCZ-equivalence)**.** *Let $f$ and $g$ be two functions mapping $\mathbb{F}_{2^n}$ to itself. They are said to be* CCZ-equivalent *if the sets $\{(x, f(x)) \mid x \in \mathbb{F}_{2^n}\}$ and $\{(x, g(x)) \mid x \in \mathbb{F}_{2^n}\}$ are affine equivalent. In other words, they are CCZ-equivalent if and only if there exists a linear permutation $L$ of $(\mathbb{F}_{2^n})^2$ such that*

$$\left\{(x, f(x)), \forall x \in \mathbb{F}_{2^n}\right\} = \left\{L\left(x, g(x)\right), \forall x \in \mathbb{F}_{2^n}\right\}.$$

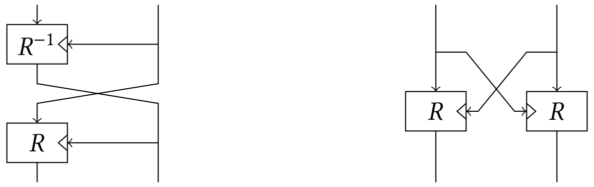
For example, a permutation is CCZ-equivalent to its inverse because the function $L : (x, y) \mapsto (y, x)$ is linear.

A key property of CCZ-equivalence is that it preserves both the differential uniformity and the Walsh spectrum (i.e. the distribution of the coefficients in the LAT).

**Lemma 14.2.1.** *The permutation $S_\mathcal{I}$ is CCZ-equivalent to the quadratic function $Q_\mathcal{I} : \mathbb{F}_{2^6} \to \mathbb{F}_{2^6}$ obtained by concatenating two bivariate polynomials of $\mathbb{F}_{2^3}$:*

$$Q_\mathcal{I}(\ell || r) = R_r(\ell) \; || \; R_\ell(r).$$

*A representation of $Q_\mathcal{I}$ is given Figure 14.8b.*



(a) The permutation $S_\mathcal{I}$.

(b) The function $Q_\mathcal{I}$.

Figure 14.8: Two CCZ-equivalent APN functions of $\mathbb{F}_2^6$.

*Proof.* The functional graph of the function $Q_\mathcal{I}$ is the following set:

$$\left\{\left(x||y,\ R_y(x)||R_x(y)\right),\ \forall x||y \in \mathbb{F}_{2^6}\right\},$$

in which we can replace the variable $x$ by $z = R_y(x)$ so that $x = R_y^{-1}(z)$ — recall that $R_k$ is invertible for all $k$. We obtain a new description of the same set:

$$\left\{\left(R_y^{-1}(z)||y,\ z||R_{R_y^{-1}(z)}(y)\right),\ \forall z||y \in \mathbb{F}_{2^6}\right\}.$$

As the function $\mu : (\mathbb{F}_{2^6})^2 \rightarrow (\mathbb{F}_{2^6})^2$ with $\mu(x||y,a||b) = (a||y,b||x)$ is linear, this graph is linearly equivalent to the following one:

$$\left\{\left(z||y,\ R_{R_y^{-1}(z)}(y)||R_y^{-1}(z)\right),\ \forall z||y \in \mathbb{F}_{2^6}\right\},$$

which is the functional graph of $S_\mathcal{I}$. We conclude that the two functions are ccz-equivalent. □

**Definition 14.2.2** (Butterfly Structure)**.** *Let $\alpha$ be in $\mathbb{F}_{2^n}$, $e$ be an integer such that $x \mapsto x^e$ is a permutation of $\mathbb{F}_{2^n}$ and $R_k[e,\alpha]$ be the keyed permutation*

$$R_k[e,\alpha](x) = (x + \alpha k)^e + k^e.$$

*We call* Butterfly Structures *the functions of $(\mathbb{F}_{2^n})^2$ defined as follows:*

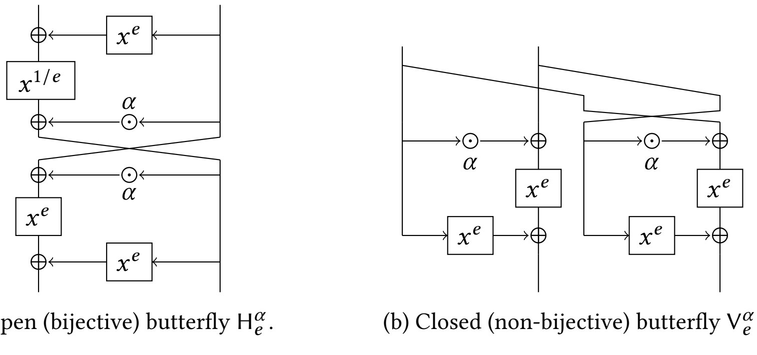
- *the* Open Butterfly *with branch size $n$, exponent $e$ and coefficient $\alpha$ is the permutation denoted $\mathsf{H}_e^\alpha$ defined by:*

$$\mathsf{H}_e^\alpha(x,y) = \left(R^{-1}_{R_y[e,\alpha](x)}(y),\ R_y[e,\alpha](x)\right),$$

- *the* Closed Butterfly *with branch size $n$, exponent $e$ and coefficient $\alpha$ is the function denoted $\mathsf{V}_e^\alpha$ defined by:*

$$\mathsf{V}_e^\alpha(x,y) = \left(R_y[e,\alpha](x),\ R_x[e,\alpha](y)\right).$$

*Furthermore, the permutation $\mathsf{H}_e^\alpha$ and the function $\mathsf{V}_e^\alpha$ are ccz-equivalent.*



(a) Open (bijective) butterfly $\mathsf{H}_e^\alpha$.

(b) Closed (non-bijective) butterfly $\mathsf{V}_e^\alpha$.

Figure 14.9: The two types of butterfly structure with coefficient $\alpha$ and exponent $e$.

Pictures representing such functions are given in Figure 14.9. Our decomposition of the 6-bit apn permutation and its ccz-equivalent function have butterfly structures: $S_\mathcal{I} = \mathsf{H}_6^2$ and $Q_\mathcal{I} = \mathsf{V}_6^2$. In fact, the proof of the ccz-equivalence of open and closed butterfly is identical to that of Lemma 14.2.1. The properties of a generalization of such structures are studied in Section 14.4, in particular in Theorem 14.4.2. In this section, we focus on the case $n = 3$.

### 14.2.3 Relations with the Kim and the Cube functions

Biryukov *et al.* suggest in [BDBP03] counting the number of pairs of affine permutations $A, B$ such that $S_I = B \circ S_I \circ A$ as a measure of the symmetries inside $S_\mathcal{I}$. An algorithm performing this task is also provided. Using it, we have found that there are only 7 such pairs including the pair of identity mappings. This property is preserved by affine transformations, meaning that our decomposition is not necessary to find this number. However, for the S-Box $S_\mathcal{I}$, these 7 pairs of transformations have a natural description. Let "⊗" be such that $(a,b) \otimes (c,d) = (ac, bd)$. Then for all $\lambda$ in $\mathbb{F}_{2^3}^*$, it holds that

$$S_\mathcal{I}(\lambda x, \lambda^{-1} y) = (\lambda, \lambda^{-1}) \otimes S_\mathcal{I}(x,y) \ . \qquad (14.2)$$

In other words, multiplying the inputs by $\lambda$ and $\lambda^{-1}$ is equivalent to multiplying the outputs by the same values.

This property is reminiscent of one of those of the "Kim mapping". This function is a non-bijective quadratic APN function from which Dillon *et al.* [BDMW10] obtained the APN permutation by applying CCZ-equivalence preserving transformations. The Kim mapping is defined over $\mathbb{F}_{2^6}$ as

$$\kappa(x) = x^3 + x^{10} + w \odot x^{24} \ ,$$

where $w$ is some primitive element of $\mathbb{F}_{2^6}$. Dillon *et al.* noticed [BDMW10] that it has the following property for all $\lambda \in \mathbb{F}_{2^3}$:

$$\kappa(\lambda x) = \lambda^3 \kappa(x) \ . \qquad (14.3)$$

We found experimentally that the Kim mapping is actually affine-equivalent to all closed butterflies $\mathsf{V}_e^\alpha$ with $n = 3, e \in \{3,5,6\}, Tr(\alpha) = 0$ and $\alpha \neq 0$. In particular, it is affine-equivalent to the function $Q_\mathcal{I} = \mathsf{V}_6^2$ described before.

The property that $k(\lambda x) = \lambda^3 k(x)$ for all $\lambda \in \mathbb{F}_{2^3}$ can be nicely translated to $\mathsf{V}_e^\alpha$ structures when $\alpha \neq 0$. Indeed, it is easy to see that the following holds for all $\lambda$ in $\mathbb{F}_{2^3}$:

$$\mathsf{V}_e^\alpha(\lambda x, \lambda y) = (\lambda^e, \lambda^e) \otimes \mathsf{V}_e^\alpha(x,y) \ .$$

In particular, setting $e = 3$ and $\alpha$ such that $\mathsf{V}_e^\alpha$ is affine-equivalent to the Kim mapping leads to a branch-wise variant of the property from Equation 14.3.

Similarly, the Open Butterflies $\mathsf{H}_e^\alpha$ exhibit the following property:

$$\mathsf{H}_e^\alpha(\lambda^e x, \lambda y) = (\lambda^e, \lambda) \otimes \mathsf{H}_e^\alpha(x,y) \text{ for all } \lambda \in \mathbb{F}_{2^3}. \qquad (14.4)$$

## 14.3 Implementing 6-bit APN Permutations

We can use the open butterfly structure to efficiently implement 6-bit APN permutations in both a bit-sliced fashion for use in software and in hardware. In this section, we explore this idea and provide an S-Box $A_o$ which is affine equivalent to $\mathsf{H}_3^2$ and has an efficient bit-sliced implementation.

### 14.3.1 Efficient Bit-Sliced Implementations

Starting from the algebraic normal forms of the operations used to compute $\mathsf{H}_3^2$ (given in Table 14.5), it is easy to write a first naïve bitsliced implementation given in Algorithm 14.1.

| function | algebraic normal form |
|---|---|
| $y = x^3$ | $y_0 = x_0x_1 \oplus x_0x_2 \oplus x_1 = x_0(x_1 \oplus x_2) \oplus x_1$ |
| | $y_1 = x_0x_2 \oplus x_1 \oplus x_2 = x_0(x_2 \oplus 1) \oplus x_1$ |
| | $y_2 = x_1x_2 \oplus x_0 \oplus x_1 \oplus x_2 = (x_0 \oplus 1) \oplus (x_1 \oplus 1)(x_2 \oplus 1)$ |
| $y = 2 \odot x$ | $y_0 = x_1$ |
| | $y_1 = x_0 \oplus x_2$ |
| | $y_2 = x_2$ |
| $y = x^5$ | $y_0 = x_0x_1 \oplus x_2$ |
| | $y_1 = x_0x_1 \oplus x_0x_2 \oplus x_1 = x_0(x_1 \oplus x_2) \oplus x_1$ |
| | $y_2 = x_1x_2 \oplus x_0 \oplus x_1 \oplus x_2 = (x_0 \oplus 1) \oplus (x_1 \oplus 1)(x_2 \oplus 1)$ |

Table 14.5: The algebraic normal form of the sub-functions used to compute $\mathsf{H}_3^2$.

**Algorithm 14.1** A naïve bitsliced implementation of the open butterfly $\mathsf{H}_3^2$.
**Inputs:** words $X_0, \ldots, X_5$;
**Outputs:** updated words $X_0, \ldots, X_5$.

```
▷ $R \oplus= L^3$
$X_0 \oplus= (X_5 \wedge (X_4 \oplus X_3)) \oplus X_4$
$X_1 \oplus= (\neg X_5 \wedge X_3) \oplus X_4$
$X_2 \oplus= (X_4 \wedge \neg X_3) \oplus (\neg X_5)$
▷ $R = R^5$
$t = (\neg X_1) \wedge (\neg X_0)$
$X_0 \oplus= X_2 \wedge X_1$
$X_1 \oplus= X_2 \wedge X_0$
$X_2 \oplus= \neg t$
▷ $R \oplus= \alpha \cdot L$
$X_0 \oplus= X_4$
$X_1 \oplus= X_5 \oplus X_3$
$X_2 \oplus= X_3$
▷ $L \oplus= \alpha \cdot R$
$X_3 \oplus= X_1$
$X_4 \oplus= X_2 \oplus X_0$
$X_5 \oplus= X_0$
▷ $L = L^3$
$u = X_3$
$t = X_4$
$X_4 \oplus= ((\neg X_5) \wedge X_3)$
$X_3 = (X_5 \wedge (u \oplus t)) \oplus t$
$X_5 \oplus= \neg(\neg t \wedge \neg u)$
▷ $L \oplus= R^3$
$X_3 \oplus= (X_2 \wedge (X_1 \oplus X_0)) \oplus X_1$
$X_4 \oplus= (\neg X_2 \wedge X_0) \oplus X_1$
$X_5 \oplus= (\neg X_1 \wedge \neg X_0) \oplus (\neg X_2)$
```

This implementation can be optimized using Boolean algebra and removing the linear component of $x \mapsto x^3$ in the first and last steps. Doing this is equivalent to applying an affine permutation before and after $\mathsf{H}_3^2$ to obtain a new permutation $A_o$. This operation preserves the differential and linear properties of the permutation while also keeping the property that $A_o^{-1} = \mathtt{Swap}_6 \circ A_o \circ \mathtt{Swap}_6$, where $\mathtt{Swap}_6$ simply swaps the two 3-bit branches. The bitsliced implementation of this simplified S-Box is given in Algorithm 14.2 and its look-up table in Table 14.6.

### 14.3.2 Hardware Implementation

Our decompositions also facilitates the hardware implementation of these S-Boxes. To illustrate this, we simulated the circuit computing these functions in three differ-

| | .0 | .1 | .2 | .3 | .4 | .5 | .6 | .7 | .8 | .9 | .a | .b | .c | .d | .e | .f |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0. | 0 | 1d | 6 | 3f | 3c | 3b | 31 | 12 | 22 | 35 | 17 | 2c | 16 | 33 | 30 | 39 |
| 1. | 2d | a | 38 | 2b | 1 | 4 | 2f | 1e | 3 | 34 | 2e | 25 | 27 | 1a | 29 | 28 |
| 2. | 2a | 7 | 14 | 3d | 36 | 19 | b | 20 | 3e | d | 37 | 8 | 1b | 2 | 9 | 1c |
| 3. | 10 | 1f | 21 | 3a | 26 | 13 | 24 | 5 | c | f | 11 | e | 23 | 32 | 15 | 18 |

Table 14.6: The look-up table of $A_o$ in hexadecimal, e.g. $A_o$(0x32) = 0x21.

**Algorithm 14.2** An optimized bitsliced implementation of $A_o$, an S-Box affine-equivalent to the open butterfly with $\alpha = 2$, $e = 3$.
**Inputs:** words $X_0, \ldots, X_5$;
**Outputs:** updated words $X_0, \ldots, X_5$;

```
$t = (X_5 \wedge X_3)$
$X_0 \oplus= t \oplus (X_5 \wedge X_4)$
$X_1 \oplus= t$
$X_2 \oplus= (X_4 \vee X_3)$
$t = (X_1 \vee X_0)$
$X_0 \oplus= (X_2 \wedge X_1) \oplus X_4$
$X_1 \oplus= (X_2 \wedge X_0) \oplus X_5 \oplus X_3$
$X_2 \oplus= t \oplus X_3$
$X_3 \oplus= X_1$
$X_4 \oplus= X_2 \oplus X_0$
$X_5 \oplus= X_0$
$u = X_3$
$t = X_4$
$X_3 \oplus= t$
$X_3 = X_3 \wedge X_5 \oplus t$
$X_4 \oplus= ((\neg X_5) \wedge u)$
$X_5 \oplus= (t \vee u)$
$t = (X_2 \wedge X_0)$
$X_3 \oplus= t \oplus (X_2 \wedge X_1)$
$X_4 \oplus= t$
$X_5 \oplus= (X_1 \vee X_0)$
```

ent ways. First, we simply gave the look-up table to the software[2] and let it find the best implementation it could ("no decomposition" case). Then, we fed it our decomposition of the different structures ("decomposed" case). We also tried implementing the cube function using finite field arithmetic but this approach did not improve our results.

The software optimizes two competing quantities. The first is the area, which simply corresponds to the physical space needed to implement the circuit using the logical gates available. The second is the propagation time, i.e. the delay necessary for the electronic signal to go through the circuit implementing the S-Box and to stabilize itself to the output value.

For each function, we repeated the experiment several times using different periods for the clock cycles. Priority is given to area optimization when the period is maximum. But as the period decreases, the priority shifts towards propagation time. The results are given in Table 14.7.

As we can see, the knowledge of the decompositions always allows a more efficient implementation: regardless of what the main optimization criteria is, both the area and the delay are decreased.

[2] We used the digital cell library SAED90n-1P9M in the "normal $V_t$, high temperature, nominal voltage" corner.

| | | Base | | | Decomposed | | |
|---|---|---|---|---|---|---|---|
| S-Box | Period (ns) | $a$ | $d$ | $a{\times}d$ | $a$ | $d$ | $a{\times}d$ |
| $H_3^2$ | 100 | 799 | 56.4 | 45079.6 | 414 | 39.3 | 16274.3 |
| | 20 | 827 | 19.75 | 16333.3 | 404 | 18.7 | 7554.8 |
| | 10 | 928 | 9.81 | 9103.7 | 431 | 9.76 | 4206.6 |
| | 5 | 1062 | 4.81 | 5108.2 | 569 | 4.81 | 2736.9 |
| $A_0$ | 100 | 774 | 53.1 | 41122.6 | 384 | 42.0 | 16131.8 |
| | 20 | 812 | 19.3 | 15671.6 | 384 | 15.4 | 5925.1 |
| | 10 | 869 | 9.63 | 8368.5 | 382 | 9.8 | 3732.1 |
| | 6 | 1041 | 5.8 | 6037.8 | 464 | 5.8 | 2691.2 |

Table 14.7: Results on the hardware implementation of our S-Boxes. The area $a$ is in $(\mu m)^2$, the delay $d$ is in $ns$ and $a \times d$ is their product.

## 14.4 Generalizing the Butterfly Construction

In this section, we will show that the butterfly structure can be generalized by using an additionnal parameter $\beta$ and by considering arbitrary odd values for its branch size $n$. These functions are all differentially 4-uniform, except those affine-equivalent to the Dillon permutation. But first, let us recall some general results about Boolean functions and their differential uniformity.

An up to date overview of known APN functions can be found in [BN15]. As APN functions operating on an even number of bits are still to be found for even block sizes larger than 6, differentially 4-uniform permutations have received a lot of attention from researchers. An obvious example is the inverse function $x \mapsto x^{2^n-2}$ of $\mathbb{F}_{2^n}$ studied in the seminal work of Nyberg [Nyb94].

However, security against differential cryptanalysis is not sufficient and linear attacks need to be taken into account too. The search can thus be focused on differentially 4-uniform permutations of $2n$ bits with non-linearity $2^{2n-1} - 2^n$ which is, as far as we know, the best that can be achieved. Whether there are functions improving this bound is an open problem (Open Problem 2 in [BL10]). The same paper also states Open Problem 1: we must find other highly non-linear differentially 4-uniform functions operating on fields of even degree. Several papers have then presented constructions for such permutations, for example using binomials [BTT12] or an APN permutation on $\mathbb{F}_{2^{n+1}}$ for even $n$ [LW14a].

The generalized butterflies and their properties are introduced in Section 14.4.1 (p. 280). The proof for their differential properties is given in Section 14.4.2 (p. 283) and the one for their algebraic degree is given in Section 14.4.3 (p. 295).

### 14.4.1 Generalized Butterflies

#### 14.4.1.1 Definition

Generalized butterflies are built using the same structure as the butterflies from Definition 14.2.2. Here, we restrict ourselves to the case where $R(x,y)$ has *univariate degree 3* as the butterflies obtained in the previous sections of this chapter have such a structure. However, we consider more sophisticated functions of univariate degree 3.

The following lemma describes all polynomials $R$ satisfying this degree condition which define a keyed permutation, as demanded by the butterfly definition.

**Lemma 14.4.1** (Degree Restriction)**.** *Let $R$ be a bivariate polynomial of $\mathbb{F}_{2^n}$ such that $R_y : x \mapsto R(x,y)$ is a permutation for any $y$ and such that all terms in $R$ are non-linear terms with degree at most 3. Then $R$ can be described using two elements of $\mathbb{F}_{2^n}$ denoted $\alpha$ and $\beta$ as*

$$R(x,y) = (x + \alpha y)^3 + \beta y^3 \ .$$

The proof of this lemma relies on the following theorem.

**Theorem 14.4.1** (Cor. 2.9 from [MS87a])**.** *Let $\mathbb{F}_q$ have characteristic different from 3. Then $f(x) = ax^3 + bx^2 + cx + d = 0$ ($a \neq 0$) permutes $\mathbb{F}_q$ if and only if $b^2 = 3ac$ and $q \equiv 2 \mod 3$.*

*Proof of Lemma 14.4.1.* Let

$$R(x,y) = Ax^3 + Bx^2y + Cxy^2 + Dy^3 + Exy.$$

Since $x \mapsto R(x,0) = Ax^3$ must be a permutation, we deduce that $A \neq 0$. A multiplication by a non-zero constant changes neither the degree nor the integral property. Therefore, we consider a normalised case where $A = 1$. We need that $x \mapsto R(x,y)$ is a permutation for any $y$. We are in characteristic 2 and, using the notation of Theorem 14.4.1, we always have that $q \equiv 2 \mod 3$ because $n$ is odd. Thus, in order to fulfill the integral condition, Theorem 14.4.1 imposes that $(By)^2 = Cy^2 + Ey$ for all $y$. This implies that $E = 0$ and $B^2 = C$. The polynomial can thus be written $R(x,y) = x^3 + Bx^2y + B^2xy^2 + Dy^3$ which we factor into $R(x,y) = (x+By)^3 + (B^3+D)y^3$. Simply setting $\alpha = B$ and $\beta = B^3 + D$ gives us the lemma. □

We are now ready to formally define the generalized butterflies considered in this section.

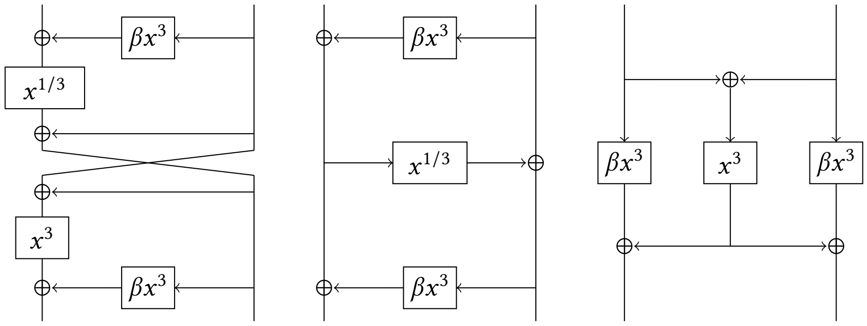
**Definition 14.4.1.** *We call the butterflies based on polynomials $R : (x,y) \mapsto (x + \alpha y)^3 + \beta y^3$* generalized butterflies. *They are denoted $\mathsf{V}_{\alpha,\beta}$ and $\mathsf{H}_{\alpha,\beta}$ for closed and open butterflies respectively.*

In this context, the results from the previous sections of this chapter can be interpreted as handling the particular case $\beta = 1$. If $\alpha = 1$, the open butterflies and closed butterflies are functionally equivalent to the functions presented in Figure 14.10. In particular, open butterflies are functionally equivalent to 3-round Feistel networks when $\alpha = 1$.

Lemma 14.4.1 excludes terms in $x^2$, $y^2$, $x$ and $y$ from $R$. Since such terms have algebraic degree 1, they could be added without changing the linearity and the differential properties of the closed butterfly, which is why we ignore them.

#### 14.4.1.2 Equivalence Relations

The proof of Lemma 14.2.1 is easily updated to show that an open and a closed generalized butterfly with identical parameters are CCZ-equivalent. There are other relations linking butterflies to each other.



(a) Open butterfly $\mathsf{H}_{1,\beta}$. (b) $\mathsf{F}^{\beta}$ (note $\mathsf{F}^{\beta} = \mathsf{H}_{1,\beta}$). (c) Closed butterfly $\mathsf{V}_{1,\beta}$.

Figure 14.10: The equivalence between $\mathsf{H}_{1,\beta}$ and $\mathsf{F}^{\beta}$.

**Equivalent Exponents.** If the exponent is equal to $e = 3 \times 2^t$, the corresponding closed butterfly is affine-equivalent to the closed butterfly with the same $\alpha, \beta$. Therefore, all results presented in this section also hold when

$$R(x,y) = (x + \alpha y)^{3\times 2^t} + \beta y^{3\times 2^t}$$

for some $t$.

**Equivalent Coefficients (Squaring).** The closed butterflies $\mathsf{V}_{\alpha,\beta}$ and $\mathsf{V}_{\alpha^2,\beta^2}$ are affine-equivalent as $\mathsf{V}_{\alpha^2,\beta^2}(x^2,y^2) = \left(\mathsf{V}_{\alpha,\beta}(x,y)\right)^2$.

**Equivalent Coefficients (Multiplication).** For any $\alpha \neq 1$, the closed butterflies $\mathsf{V}_{\alpha,\beta}$ and $\mathsf{V}_{\alpha,\beta'}$ with $\beta' = \beta^{-1}(1+\alpha)^6$ are affine-equivalent. This equivalence is obtained by composing $\mathsf{V}_{\alpha,\beta}$ with the inverse of the linear permutation

$$L : (x,y) \mapsto (z_1, z_2) = (\alpha x + y, x + \alpha y) \ .$$

Indeed, as we can see, $\mathsf{V}_{\alpha,\beta} \circ L^{-1}(x,y)$ is affine-equivalent to $\mathsf{V}_{\alpha,\beta}$:

$$\begin{aligned}\mathsf{V}_{\alpha,\beta} \circ L^{-1}(x,y) &= \left(z_2^3 + \beta\left[(1+\alpha)^{-2}(z_1 + \alpha z_2)\right]^3, z_1^3 + \beta\left[(1+\alpha)^{-2}(z_2 + \alpha z_1)\right]^3\right) \\ &= \left((1+\alpha)^{-6}\left[(z_1 + \alpha z_2)^3 + \beta' z_2^3\right], (1+\alpha)^{-6}\left[(z_2 + \alpha z_1)^3 + \beta' z_1^3\right]\right) \ .\end{aligned}$$

**Multiplicative Stability.** As in Section 14.2.3 (p. 277), let $\otimes$ be such that $(a,b) \otimes (c,d) = (ac, bd)$ for any pairs $(a,b)$ and $(c,d)$ of $(\mathbb{F}_{2^n})^2$. Then the generalized butterflies exhibit the following properties:

$$\mathsf{V}_{\alpha,\beta}\left((\lambda,\lambda) \otimes (x,y)\right) = (\lambda^3, \lambda^3) \otimes \mathsf{V}_{\alpha,\beta}(x,y),$$

and

$$\mathsf{H}_{\alpha,\beta}\left((\lambda^3,\lambda) \otimes (x,y)\right) = (\lambda^3, \lambda) \otimes \mathsf{H}_{\alpha,\beta}(x,y).$$

Note that these multiplicative properties correspond to the so-called *subspace property* introduced in [BDMW10] and investigated in [Göl15].

#### 14.4.1.3 Cryptographic Properties

The following theorem summarizes all the cryptographic properties of the generalized butterflies.

**Theorem 14.4.2.** *The cryptographic properties of the generalized butterflies* $\mathsf{V}_{\alpha,\beta}$ *and* $\mathsf{H}_{\alpha,\beta}$*, which are based on functions* $R : (x, y) \mapsto (x + \alpha y)^3 + \beta y^3$ *with* $\alpha, \beta \neq 0$ *are as follows:*

- *the algebraic degree of* $\mathsf{V}_{\alpha,\beta}$ *is always equal to 2,*
- *if* $n = 3$, $\alpha \neq 0$, $\mathrm{Tr}(\alpha) = 0$ *and* $\beta \in \{\alpha^3 + \alpha, \alpha^3 + 1/\alpha\}$ *then the butterflies are* APN*, have a linearity equal to* $2^{n+1}$ *and the algebraic degree of* $\mathsf{H}_{\alpha,\beta}$ *is equal to* $n + 1$*;*
- *if* $\beta = (1 + \alpha)^3$ *then the differential uniformity is equal to* $2^{n+1}$*, the linearity is equal to* $2^{(3n+1)/2}$ *and the algebraic degree of* $\mathsf{H}_{\alpha,\beta}$ *is equal to* $n$*;*
- *otherwise, the differential uniformity is equal to 4, the linearity is equal to* $2^{n+1}$ *and the algebraic degree of* $\mathsf{H}_{\alpha,\beta}$ *is either* $n$ *or* $n + 1$*. It is equal to* $n$ *if and only if*

$$1 + \alpha\beta + \alpha^4 = (\beta + \alpha + \alpha^3)^2.$$

*In particular, there are no* APN *butterflies operating on more than 6 bits.*

Open generalized butterflies with $\beta \neq (1 + \alpha)^3$ form a family of permutations operating on $2n$ bits with a linearity and a differential uniformity equal to the best known to be possible. Furthermore, the only known APN permutation on fields of even dimension is, up to affine-equivalence, a generalized butterfly as well.

The proof of this theorem is divided into several parts. Section 14.4.2 (p. 283) proves their differential uniformity and Section 14.4.3 (p. 295) proves their algebraic degree. The proof of their linearity, found by Anne Canteaut, is available in our joint paper [CDP17].

### 14.4.2 Differential Properties of Generalized Butterflies

In this section, we describe the differential properties of generalized butterflies. First, Section 14.4.2.1 is dedicated to Theorem 14.4.3, which shows that generalized butterflies $\mathsf{V}_{\alpha,\beta}$ and $\mathsf{H}_{\alpha,\beta}$ have differential uniformity at most 4, unless $\beta = (1 + \alpha)^3$. Then, Theorem 14.4.4 and its immediate consequence, Corollary 14.4.1, give a necessary and sufficient condition on $\alpha$ and $\beta$ for a generalized butterfly to be APN. This corollary is then used to show Proposition 14.4.1 which states that there are no APN butterflies if $n > 3$. These results are presented and proved in Section 14.4.2.2. Section 14.4.2.3 focuses on the special case $\alpha = \beta = 1$, for which the generalized butterflies are equivalent to a 3-round Feistel network. In this case, we recover, with a different proof, a result from [LW14b], which states that the difference distribution tables of the corresponding butterflies do not contain the value 2.

But first, we give a lemma playing a crucial role in these proofs. It lets us easily derive the exact number of solutions of some particular degree-4 equations that appear several times in our proof. Since most proofs in this section rely on the number of solutions of univariate equations over $\mathbb{F}_{2^n}$, the concept of “degree” always refers to the univariate degree in what follows.

**Lemma 14.4.2.** *Let $U, V$ be elements of $\mathbb{F}_{2^n}$ with $n$ odd and let $Uz^4+Vz^2+(U+V)z = C$ be some linearized degree-4 equation in $z$. It has:*

- *0 or $2^n$ solutions if $U = V = 0$,*
- *0 or 4 solutions if $U \neq 0$, $U \neq V$ and $\mathrm{Tr}(V/U) = 1$,*
- *0 or 2 solutions otherwise, that is if one of the following is true:*

$$\Big(U = 0, V \neq 0\Big) \text{ or } \Big(U \neq 0, V = U\Big) \text{ or } \Big(U \neq 0, \mathrm{Tr}(V/U) = 0\Big) .$$

*Proof.* First of all, for any value of the constant $C$, the number of solutions of the equation is either zero or equal to the number of solutions of the linearized equation $Uz^4 + Vz^2 + (U + V)z = 0$. We then only need to study the case $C = 0$. Obviously, the number of solutions is always even as if $z$ is a solution then $z + 1$ is too.

If $U = V = 0$ then the linearized equation does not involve $z$, meaning that all values of $z$ satisfy it. We now suppose that either $U \neq 0$ or $V \neq 0$.

If $U = 0$ then the equation corresponds to $Vz(z + 1) = 0$, implying that it has 2 solutions. Let us now suppose that $U \neq 0$. In this case, we can rewrite the equation as

$$Uz(z + 1)\Big(1 + V/U + z(z + 1)\Big) = 0.$$

Both $z = 0$ and $z = 1$ are obviously solutions. In fact, they are the only ones if $V = U$. Let us suppose that $V \neq U$. The term $(z^2 + z + 1 + V/U)$ can be equal to 0 if and only if $\mathrm{Tr}(V/U) = 1$, meaning that the linearized equation has 2 solutions if $\mathrm{Tr}(V/U) = 0$ and 4 otherwise. □

#### 14.4.2.1 The Non-APN Cases

**Theorem 14.4.3** (Differential uniformity). *Let $n > 1$ be an odd integer and $(\alpha, \beta)$ be a pair of nonzero elements in $\mathbb{F}_{2^n}$. If $\beta \neq (1 + \alpha)^3$ then the generalized butterfly with parameters $\alpha$ and $\beta$ has differential uniformity at most* 4. *Moreover, it has differential uniformity* exactly 4 *unless $\beta \in \{(\alpha + \alpha^3), (\alpha^{-1} + \alpha^3)\}$.*

*If $\beta = (1 + \alpha)^3$, the generalized butterfly with parameters $\alpha$ and $\beta$ has differential uniformity $2^{n+1}$.*

*Proof.* In order to bound the differential uniformity of $\mathrm{V}_{\alpha,\beta}$, we must bound the number of solutions $(x, y)$ of the following system:

$$\begin{cases} R(x, y) + R(x + a, y + b) = c \\ R(y, x) + R(y + b, x + a) = d \end{cases}$$

for any tuple $(a, b, c, d)$ of $\mathbb{F}_{2^n}$ with $(a, b) \neq (0, 0)$. It holds that

$$\begin{aligned} & R(x, y) + R(x + a, y + b) = \\ & (ax^2 + a^2x) + \alpha(bx^2 + a^2y) + \alpha^2(b^2x + ay^2) + (\alpha^3 + \beta)(by^2 + b^2y) + R(a, b) . \end{aligned}$$

Let $u = a + \alpha b$. Then this expression is simplified into

$$\begin{aligned} & R(x, y) + R(x + a, y + b) = \\ & ux^2 + u^2x + (\alpha^2u + b\beta)y^2 + (\alpha u^2 + b^2\beta)y + R(a, b) . \end{aligned}$$

Similarly, for $v = \alpha a + b$, we have

$$\begin{aligned} &R(y,x) + R(y+b, x+a) \;= \\ &(\alpha^2 v + a\beta)x^2 + (\alpha v^2 + a^2\beta)x + vy^2 + v^2 y + R(b,a) \ , \end{aligned}$$

implying that we search for the solutions of

$$\begin{cases} ux^2 + u^2 x + (\alpha^2 u + b\beta)y^2 + (\alpha u^2 + b^2\beta)y = c' \\ (\alpha^2 v + a\beta)x^2 + (\alpha v^2 + a^2\beta)x + vy^2 + v^2 y = d' \ . \end{cases} \tag{14.5}$$

**Special cases.** We first focus on three special cases, namely $b = \alpha^{-1}a, b = \alpha a$ and $b = 0$. The rest of the proof will be dedicated to the general case, when $b$ differs from these three values. We also consider a fourth special case corresponding to $\beta = (1+\alpha)^3$ and $b = a$ in which the result is different.

- $\boldsymbol{b = \alpha^{-1}a}$, or equivalently $u = 0$. Note that neither $a$ nor $b$ vanishes, since it would imply $a = b = 0$, which has been excluded. In this case, Equation (14.5) can be written as

  $$\begin{cases} (b\beta)y^2 + (b^2\beta)y = c' \\ (\alpha^2 v + a\beta)x^2 + (\alpha v^2 + a^2\beta)x + vy^2 + v^2 y = d' \ . \end{cases}$$

  Since $\beta \neq 0$ and $b \neq 0$, we deduce that the first equation has at most two solutions $y_0$ and $y_1$. For each of these two solutions, the second equation has at most two solutions because the coefficients of $x^2$ and of $x$ cannot simultaneously vanish. Indeed

  $$(\alpha^2 v + a\beta) = (\alpha v^2 + a^2\beta) = 0 \tag{14.6}$$

  implies that

  $$a^2\beta = \alpha v^2 = \alpha^2 a v \ ,$$

  leading to

  $$\alpha v(v + a\alpha) = \alpha v b = 0 \ ,$$

  which is impossible since $v = 0$ together with Equation (14.6) would imply that $a = 0$. Therefore, Equation (14.5) has at most four solutions when $u = 0$.

- $\boldsymbol{b = \alpha a}$, or equivalently $v = 0$. This case is similar to the previous one. Indeed, Equation (14.5) now corresponds to

  $$\begin{cases} ux^2 + u^2 x + (\alpha^2 u + b\beta)y^2 + (\alpha u^2 + b^2\beta)y = c' \\ a\beta x^2 + a^2\beta x = d' \end{cases}$$

  Since $a\beta \neq 0$, the second equation has at most two solutions $x_0$ and $x_1$. For each of these solutions, the first equation has at most two solutions for $y$ since the coefficients of $y^2$ and $y$ cannot simultaneously vanish. Otherwise, we would have

  $$b^2\beta = \alpha u^2 = \alpha^2 b u \ ,$$

  implying $\alpha u a = 0$.

- ***b* = 0**. In this case, System (14.5) corresponds to

$$\begin{cases} ax^2 + a^2x + \alpha^2ay^2 + \alpha a^2y = c' \\ (\alpha^3a + a\beta)x^2 + (\alpha^3a^2 + a^2\beta)x + \alpha ay^2 + \alpha^2a^2y = d' \ . \end{cases}$$

By summing the first equation and the second one multiplied by $\alpha$, we get that

$$y\alpha a^2(1+\alpha^2) = (a + a\alpha^4 + a\alpha\beta)(x^2 + ax) + g$$

for some constant $g$. Let us first consider the case when $\alpha = 1$. Then, we get

$$a\beta(x^2 + ax) = g \ .$$

Since $a\beta \neq 0$, this equation has at most two solutions $x_0$ and $x_1$. Then, for each $x_i$, the first equation in the system provides at most two solutions for $y$, leading to at most four solutions $(x, y)$ for the whole system.

Let us now assume that $\alpha \neq 1$. Then, we replace $y$ by its value, i.e. $y = \mu(x^2 + ax) + g'$ in the first equation of the system, and we get

$$\alpha^2a\mu^2x^4 + \left[a + \alpha^2a^3\mu^2 + \alpha a^2\mu\right]x^2 + \left[a^2 + \alpha a^3\mu\right]x = c' \ ,$$

where

$$\mu = \frac{(1+\alpha^4+\alpha\beta)}{\alpha a(1+\alpha^2)} \ .$$

By replacing $x = ax'$, we deduce that

$$Ux'^4 + Vx'^2 + (U+V)x' = c' \tag{14.7}$$

with

$$U = \alpha^2a^5\mu^2 \text{ and } V = a^3 + \alpha^2a^5\mu^2 + \alpha a^4\mu \ .$$

This equation has at most four solutions $x_i$, and each $x_i$ leads to a single $y$, implying that the whole system has at most four solutions.

We now show that the whole system has at most two solutions for any $a \neq 0$ for two values of $\beta$ only. The case $\alpha = 1$ can be excluded: $\mathrm{V}_{1,\beta}$ cannot be APN because any three-round Feistel network has differential uniformity at least 4 [LW14b].

If $\mathrm{V}_{\alpha,\beta}$ is APN, then the previous Equation (14.7) has at most two solutions for any $a \neq 0$ and any $c'$. We derive from Lemma 14.4.2 that this happens if and only if, for all $a \neq 0$,

$$\Big(U = 0, V \neq 0\Big) \text{ or } \Big(U \neq 0, V = U\Big) \text{ or } \Big(U \neq 0, \mathrm{Tr}\,(V/U) = 0\Big) \ .$$

We first observe that $V \neq 0$, otherwise

$$\alpha^2a^2\mu^2 + \alpha a\mu + 1 = 0$$

which would mean that $(\alpha a\mu)$ is a root of $X^2 + X + 1$ while this polynomial is irreducible over $\mathbb{F}_{2^n}$, $n$ odd. Then, the first condition means that

$$\mu = \frac{(1+\alpha^4+\alpha\beta)}{\alpha a(1+\alpha^2)} = 0$$

or equivalently

$$\beta = \alpha^{-1} + \alpha^3 \ .$$

The second condition corresponds to

$$\alpha a \mu = 1 \Leftrightarrow 1 + \alpha^4 + \alpha\beta = 1 + \alpha^2 \ ,$$

which is equivalent to

$$\beta = \alpha + \alpha^3 \ .$$

The last condition corresponds to

$$\begin{aligned} \mathrm{Tr}(V/U) &= \mathrm{Tr}(1) + \mathrm{Tr}\left(\frac{1+\alpha a \mu}{a^2\alpha^2\mu^2}\right) \\ &= 1 + \mathrm{Tr}\left(\frac{1}{a^2\alpha^2\mu^2}\right) + \mathrm{Tr}\left(\frac{1}{\alpha a \mu}\right) = 1 = 0 \ , \end{aligned}$$

which is impossible, Therefore, the only values of $\beta$ for which $\mathrm{V}_{\alpha,\beta}$ *can* be APN are $\beta = \alpha^{-1} + \alpha^3$ and $\beta = \alpha + \alpha^3$.

- $\boldsymbol{b = a}$ **and** $\boldsymbol{\beta = (1+\alpha)^3}$. Note that $\beta = (1+\alpha)^3 \neq 0$ implies that $\alpha \neq 1$. In this case, Equation (14.5) is equal to

  $$\begin{cases} b(1+\alpha)x^2 + b^2(1+\alpha)^2x + b(1+\alpha)y^2 + b^2(1+\alpha)^2y = c' \\ b(1+\alpha)x^2 + b^2(1+\alpha)^2x + b(1+\alpha)y^2 + b^2(1+\alpha)^2y = d' \ . \end{cases}$$

  Thus, it has no solution if $c' \neq d'$. If $c' = d'$, it is equivalent to the single equation

  $$(x+y)^2 + b(1+\alpha)(x+y) = c'b^{-1}(1+\alpha)^{-1} \ ,$$

  since $\alpha \neq 1$ and $b \neq 0$. Thus, either System (14.5) has no solution or its solutions are of the form $x + y = \varepsilon$ for two values of $\varepsilon$, depending on $(b, c')$. In particular, System (14.5) has exactly $2^{n+1}$ solutions when $c' = d' = 0$.

**General case.** Let us now assume that $u$, $v$ and $b$ are all nonzero. We also suppose that $a = b$ and $\beta = (1+\alpha)^3$ do not hold simultaneously. Let $\ell_1$ and $\ell_2$ respectively denote the two equations in (14.5). Then the following expression must be constant:

$$\begin{aligned} v\ell_1 + u\ell_2 &= \left(uv(\alpha^2+1) + au\beta\right)x^2 + \left(uv(u+\alpha v) + a^2u\beta\right)x \\ &\quad + \left(uv(\alpha^2+1) + bv\beta\right)y^2 + \left(uv(\alpha u + v) + b^2v\beta\right)y \\ &= \left(uv(\alpha^2+1) + au\beta\right)(x^2+ax) + \left(uv(\alpha^2+1) + bv\beta\right)(y^2+by) \ , \end{aligned}$$

where the last equality comes from the fact that $(u + \alpha v) = a(\alpha^2 + 1)$ and $(\alpha u + v) = b(\alpha^2 + 1)$. We have obtained a relation of the form

$$\lambda_0(x^2+ax) + \lambda_1(y^2+by) = \varepsilon \tag{14.8}$$

for some constant $\varepsilon$. We first prove that $\lambda_0$ and $\lambda_1$ cannot simultaneously vanish. Let us first consider the case $\alpha = 1$. Then it holds that

$$\lambda_0 = (a+b)a\beta \text{ and } \lambda_1 = (a+b)b\beta \ .$$

Since $u = a + b \neq 0$, $b \neq 0$ and $\beta \neq 0$, $\lambda_1$ does not vanish. Let us now assume that $\alpha \neq 1$. Then we can write

$$\beta = (1 + \alpha^2)(\alpha + \beta')\ .$$

It follows that

$$\begin{aligned} \lambda_0 &= uv(\alpha^2 + 1) + au\beta = u(\alpha^2 + 1)(b + a\beta') \\ \lambda_1 &= uv(\alpha^2 + 1) + bv\beta = v(\alpha^2 + 1)(a + b\beta')\ . \end{aligned}$$

Then Equation (14.8) holds with

$$\lambda_0 = u(b + a\beta') \text{ and } \lambda_1 = v(a + b\beta')\ .$$

These two coefficients cannot simultaneously vanish, otherwise, it would lead to

$$a\beta' = b \text{ and } b\beta' = a$$

implying that

$$ab\beta' = a^2 = b^2 \text{ and } \beta' = 1\ ,$$

which has been excluded since it implies that $\beta = (1 + \alpha)^3$ and $a = b$.

We now combine Equation (14.8) with one of the equations in System (14.5). We need to consider two different cases:

- If $\lambda_0 = 0$, then Equation (14.8), which can be written as

  $$y^2 + by = \varepsilon'\ ,$$

  has at most two solutions $y_0$ and $y_1$. By replacing $y$ by these two values in the first equation in (14.5), we get at most two solutions for $x$ for each $y_i$ since $u \neq 0$.

- If $\lambda_0 \neq 0$, then Equation (14.8) can be written as

  $$x^2 = ax + \lambda_0^{-1}\lambda_1(y^2 + by) + \varepsilon'\ .$$

  We replace $x^2$ by this expression in the first equation in Equation (14.5), and we get

  $$(ua + u^2)x + (u\lambda_0^{-1}\lambda_1 + \alpha^2 u + b\beta)y^2 + (u\lambda_0^{-1}\lambda_1 b + \alpha u^2 + b^2\beta)y = c'\ .$$

  The coefficient of $x$ does not vanish since $u = a$ is equivalent to $b = 0$. Then $x$ can be written as a degree-2 polynomial in $y$, i.e.

  $$x = \mu_2 y^2 + \mu_1 y + \mu_0\ . \tag{14.9}$$

  By replacing $x$ by its value in Equation (14.8), we derive that

  $$\lambda_1(y^2 + by) = \lambda_0\left(\mu_2^2 y^4 + \mu_1^2 y^2 + a\mu_2 y^2 + a\mu_1 y\right) + \varepsilon''$$

  leading to

  $$\lambda_0\mu_2^2 y^4 + \left(\lambda_1 + \lambda_0\mu_1^2 + \lambda_0 a\mu_2\right) y^2 + \left(\lambda_1 b + \lambda_0 a\mu_1\right) y + \varepsilon''\ . \tag{14.10}$$

  Let us first assume that the three coefficients of $y^4$, $y^2$ and $y$ do not simultaneously vanish. Then Equation (14.10) has at most four solutions, $y_i$ for $0 \leq i < 4$.

Moreover, we know from Equation (14.9) that $x$ is entirely determined by $y$. It follows that System (14.5) has at most four solutions $(x, y)$.

Let us now suppose that all coefficients of Equation (14.10) vanish. Then we have $\mu_2 = 0$ and

$$\lambda_1 + \lambda_0 \mu_1^2 = 0 \text{ and } \lambda_1 b + \lambda_0 a \mu_1 = 0 \ .$$

This may occur in one of the following two situations:

- $\mu_1 = 0$ and $\lambda_1 = 0$. Using that $\lambda_1$ cannot vanish only when $\alpha \neq 1$, the definitions of $\lambda_1$ and $\mu_1$ imply that

  $$\alpha u^2 + b^2 \beta = 0 \text{ and } u(\alpha^2 + 1) + b\beta = 0 \ ,$$

  leading to

  $$\alpha u^2 = u(\alpha^2 + 1)b \ ,$$

  that is,

  $$\alpha a + b = v = 0 \ ,$$

  which has been excluded.

- $b\mu_1 = a$ and $b^2\lambda_1 = a^2\lambda_0$. From the definition of $\mu_2$, we deduce that $\mu_2 = 0$ together with this last relation implies that

  $$\begin{array}{rcl} 0 & = & u\lambda_0^{-1}\lambda_1 + \alpha^2 u + b\beta \\ & = & b^{-2}\left(ua^2 + u\alpha^2 b^2 + b^3\beta\right) \\ & = & b^{-2}(u^3 + b^3\beta) \end{array}$$

  i.e.,

  $$\beta = (ub^{-1})^3 \ . \qquad (14.11)$$

  Since $\mu_2 = 0$ and $\mu_1 = ab^{-1}$, Equation (14.9) can be written

  $$ay = bx + \mu_0' \ .$$

  By replacing $ay$ by its value in the second equation of System (14.5) multiplied by $a^2$, we get

  $$\begin{array}{rcl} x^2\left[vb^2 + \alpha^2 a^2 v + a^3\beta\right] + x\left[v^2 ab + \alpha a^2 v^2 + a^4\beta\right] & = & d'' \\ \Leftrightarrow (v^3 + a^3\beta)(x^2 + ax) & = & d'' \ . \end{array} \qquad (14.12)$$

  The coefficients of this equation do not vanish. Otherwise, combined with Equation (14.11), it would yield

  $$v^3 + a^3\beta = (\alpha a + b)^3 + (a^2 b^{-1} + \alpha a)^3 = 0$$

  which implies

  $$b = a^2 b^{-1}$$

  i.e., $a = b$ and $\beta = (\alpha + 1)^3$ which has been excluded.

  It follows that Equation(14.12) has at most two solutions $x_0$ and $x_1$. Since $y$ is entirely determined by $x$ (or constant), it follows that System (14.5) has at most two solutions in this case.

□

#### 14.4.2.2 On Apn Butterflies

We first derive a necessary and sufficient condition for a generalized butterfly to be APN in Theorem 14.4.4. Then, we simplify these conditions in Corollary 14.4.1. Finally, we show in Proposition 14.4.1 that this condition can only be satisfied if $n = 3$.

**Theorem 14.4.4** (Apn Condition)**.** *Let $\alpha \neq 0, 1$. A generalized butterfly with parameters $\alpha$ and $\beta$ is APN if and only if:*

$$\beta \in \{\alpha + \alpha^3, \alpha^{-1} + \alpha^3\} \text{ and } \mathrm{Tr}\left(\mathcal{A}_\alpha(e)\right) = 1, \ \forall\, e \notin \{0, \alpha, 1/\alpha\} ,$$

*where*

$$\mathcal{A}_\alpha(e) = \frac{e\alpha(1+\alpha)^2}{(1+\alpha e)(\alpha+e)^2} .$$

*Proof.* Since we have proved in Section 14.4.1.2 that generalized butterflies with parameters $(\alpha, \beta_0)$ and $(\alpha, \beta_1)$ where $\beta_1 = \beta_0^{-1}(1+\alpha)^6$ are affine-equivalent, we only need to prove the result for $\beta = \alpha + \alpha^3$. As before, we need to count the number of solutions of

$$\begin{cases} R(x,y) + R(x+a, y+b) = c \\ R(y,x) + R(y+b, x+a) = d \end{cases} \tag{14.13}$$

for any tuple $(a, b, c, d)$ of $\mathbb{F}_{2^n}$ with $(a, b) \neq (0, 0)$. This system is equivalent to

$$\begin{cases} ax^2 + a^2x + \alpha(bx^2 + a^2y) + \alpha^2(b^2x + ay^2) + (\alpha^3 + \beta)(by^2 + b^2y) = c_0 \\ by^2 + b^2y + \alpha(ay^2 + b^2x) + \alpha^2(a^2y + bx^2) + (\alpha^3 + \beta)(ax^2 + a^2x) = d_0 . \end{cases}$$

As $\alpha \neq 1$, we can replace the lines $\ell_1$ and $\ell_2$ of this system by $\ell_1 + \alpha\ell_2$ and $\alpha\ell_1 + \ell_2$ to obtain a system with the exact same number of solutions. We obtain

$$\begin{cases} (ax^2 + a^2x)(1 + \alpha\beta + \alpha^4) + (\alpha + \alpha^3)(bx^2 + a^2y) + (\alpha^3 + \alpha + \beta)(by^2 + b^2y) = c_0 \\ (by^2 + b^2y)(1 + \alpha\beta + \alpha^4) + (\alpha + \alpha^3)(ay^2 + b^2x) + (\alpha^3 + \alpha + \beta)(ax^2 + a^2x) = d_0 . \end{cases}$$

For $\beta = \alpha + \alpha^3$, the system is further simplified using that $1 + \alpha\beta + \alpha^4 = (1 + \alpha^2)$ and $\alpha + \alpha^3 + \beta = 0$:

$$\begin{cases} (ax^2 + a^2x) + \alpha(bx^2 + a^2y) = c_1 \\ (by^2 + b^2y) + \alpha(ay^2 + b^2x) = d_1 . \end{cases} \tag{14.14}$$

We first consider the cases $a = 0$ and $b = 0$. Recall that $a = b = 0$ is excluded. If $a = 0$, then the first line of the system is equivalent to

$$x = \left(\frac{c_1}{\alpha b}\right)^{2^{n-1}} .$$

Replacing $x$ by this value in the second line of System (14.14) yields a degree-2 equation in $y$ with nonzero coefficients since $b \neq 0$, implying that (14.14) has at most two solutions $(x, y)$. The case $b = 0$ is similar.

We now suppose $a \neq 0$ and $b \neq 0$, which allows us to set $x = ax'$ and $y = by'$. In this context, System (14.14) has as many solutions as

$$\begin{cases} a^3(x'^2 + x') + \alpha a^2 b(x'^2 + y') = c_1 \\ b^3(y'^2 + y') + \alpha a b^2(y'^2 + x') = d_1 , \end{cases}$$

which we rewrite using $e = a/b$ as

$$\begin{cases} e(x'^2 + x') + \alpha(x'^2 + y') = c_2 \\ e^{-1}(y'^2 + y') + \alpha(y'^2 + x') = d_2 \ . \end{cases} \tag{14.15}$$

Summing its lines yields

$$(x'^2 + x')(e + \alpha) + (y'^2 + y')(e^{-1} + \alpha) \ = \ c_2 + d_2 \ .$$

If $e = \alpha$, then $y'$ is fixed to either $y'_0$ or $y'_1$ with $y'_0 + y'_1 = 1$. The first line of the system implies in this case that $x' = y'_i + c_2/\alpha$ as the terms in $x^2$ cancel each other, meaning that the system has at most two solutions. The case $e = \alpha^{-1}$ is similar. We now suppose $e \neq \alpha, \alpha^{-1}$.

The first line of System (14.15) allows us to express $y'$ as a function of $x'$:

$$y' = x'^2 \left( \frac{e}{\alpha} + 1 \right) + x' \frac{e}{\alpha} + \frac{c_2}{\alpha} \ .$$

We replace $y'$ by this expression in the second line of System (14.15) and obtain

$$\begin{aligned} d_3 &= y'^2 \left( e^{-1} + \alpha \right) + y' e^{-1} + \alpha x' \\ &= \left( x'^2 \left( \frac{e}{\alpha} + 1 \right) + x' \frac{e}{\alpha} \right)^2 \left( e^{-1} + \alpha \right) + \left( x'^2 \left( \frac{e}{\alpha} + 1 \right) + x' \frac{e}{\alpha} \right) e^{-1} + \alpha x' \\ &= x'^4 \left( 1 + \frac{e}{\alpha} \right)^2 (\alpha + e^{-1}) + x'^2 \left( \frac{e^2}{\alpha^2} (e^{-1} + \alpha) + \left( \frac{e}{\alpha} + 1 \right) e^{-1} \right) + x' \left( \frac{1}{\alpha} + \alpha \right) \end{aligned}$$

for some constant $d_3$. If we let $U = (1 + e/\alpha)^2(\alpha + 1/e)$ and $V = U + 1/\alpha + \alpha$, then the number of solutions of this equation can be computed using Lemma 14.4.2. First, $U \neq 0$ and $U + V \neq 0$ as $\alpha \neq 1$. Therefore, the possible number of solutions is at most equal to 4 and is given by the trace of $V/U$: if $\mathrm{Tr}\,(V/U) = 0$ then the equation has at most 2 solutions, otherwise it has 0 or 4 solutions. It holds that

$$\begin{aligned} \frac{V}{U} &= 1 + \frac{\alpha^{-1} + \alpha}{(e^{-1} + \alpha)(1 + e\alpha^{-1})^2} \\ &= 1 + \frac{e\alpha(1 + \alpha)^2}{(1 + \alpha e)(\alpha + e)^2} \end{aligned}$$

so the function is APN if and only if

$$\mathrm{Tr}\,(\mathcal{A}_\alpha(e)) = 1, \ \forall e \notin \{0, \alpha, 1/\alpha\}, \text{ with } \mathcal{A}_\alpha(e) = \frac{e\alpha(1 + \alpha)^2}{(1 + \alpha e)(\alpha + e)^2} \ .$$

□

The condition provided by Theorem 14.4.4 is sufficient to describe all APN generalized butterflies but it can be greatly simplified. This is stated in the following corollary.

**Corollary 14.4.1.** *Let $\alpha \neq 1$, $\beta_0 = \alpha^3 + \alpha$ and $\beta_1 = \alpha^3 + 1/\alpha$. A generalized butterfly with parameters $\alpha$ and $\beta$ is APN if and only if $\beta = \beta_0$ or $\beta_1$ and*

$$\mathrm{Tr}\,(C_\alpha(v)) = 1, \ \forall \, u \notin \left\{ 0, 1, 1/(1 + \alpha^{-2}) \right\} \ ,$$

*with*

$$C_\alpha(v) = \left( \frac{1}{1 + \alpha^{-1}} \right)^4 \frac{1}{u + u^3} .$$

*Proof of Corollary 14.4.1.* We know from Theorem 14.4.4 that a generalized butterfly with parameters $\alpha$ and $\beta$ is APN if and only if $\beta \in \{\beta_0, \beta_1\}$ and $\mathrm{Tr}\,(\mathcal{A}_\alpha(e)) = 1$ for all $e$ not in $\{0, \alpha, 1/\alpha\}$. Suppose that $\alpha \neq 1$ and let $\ell = (e+\alpha)(1+\alpha)^2$. Then we can rewrite some of the expressions involved in $\mathcal{A}_\alpha(e)$ as follows:

$$e(1+\alpha)^2 = \ell + \alpha + \alpha^3 \text{ and } (1+\alpha e)(1+\alpha)^2 = \alpha\left(\ell + \frac{(1+\alpha)^4}{\alpha}\right) .$$

Recall that $\beta_0 = \alpha + \alpha^3$ and $\beta_1 = (\alpha+1)^4/\alpha$, so we can write:

$$\begin{aligned}
\mathcal{A}_\alpha(e) &= \frac{e\alpha(1+\alpha)^2}{(1+\alpha e)(\alpha+e)^2} \\
&= \frac{\alpha\, e(1+\alpha^2)}{(1+\alpha e)(1+\alpha^2)\frac{((\alpha+e)(1+\alpha)^2)^2}{(1+\alpha)^6}} \\
&= (1+\alpha)^6 \frac{\alpha(\ell+\beta_0)}{\alpha(\ell+\beta_1)\ell^2} \\
&= \frac{\beta_0\beta_1}{\ell^2}\frac{\ell+\beta_0}{\ell+\beta_1} .
\end{aligned}$$

Let $\mathcal{B}_\alpha(v) = v^2(v+1)/(v+\beta_0/\beta_1)$. Then the following equality holds:

$$\mathcal{B}_\alpha\left(\frac{\beta_0}{\ell}\right) = \frac{\beta_0\beta_1}{\ell^2}\frac{\ell+\beta_0}{\ell+\beta_1} = \mathcal{A}_\alpha(e) .$$

It is therefore sufficient to study the trace of $\mathcal{B}_\alpha$. The condition $e \notin \{0, \alpha, \alpha^{-1}\}$ becomes $\ell \notin \{\beta_0, 0, \beta_1\}$ respectively and, equivalently, $\beta_0/\ell \notin \{0, 1, \beta_0/\beta_1\}$. As a consequence, the generalized butterfly with parameters $\alpha, \beta$ is APN if and only if $\beta = \beta_0$ or $\beta_1$ and

$$\mathrm{Tr}\,(\mathcal{B}_\alpha(v)) = 1,\ \forall v \notin \left\{0, 1, \frac{\alpha^2}{1+\alpha^2}\right\}$$

as $\beta_0/\beta_1 = \alpha^2/(1+\alpha^2)$. Finally, we note that the trace of $\mathcal{B}_\alpha(v)$ can be simplified:

$$\begin{aligned}
\mathrm{Tr}\,(\mathcal{B}_\alpha(v)) &= \mathrm{Tr}\left(v^2\frac{1+v}{v+\frac{\alpha^2}{1+\alpha^2}}\right) \\
&= \mathrm{Tr}\left(v^2 + \frac{v^2}{(1+\alpha^2)v+\alpha^2}\right) \\
&= \mathrm{Tr}\left(v + \frac{v^2}{(1+\alpha^2)v+\alpha^2}\right) \\
&= \mathrm{Tr}\left(\frac{(1+\alpha^2)v^2+\alpha^2 v+v^2}{(1+\alpha^2)v+\alpha^2}\right) \\
&= \mathrm{Tr}\left(\frac{v^2+v}{\gamma v+1}\right) ,
\end{aligned}$$

where $\gamma = 1 + \alpha^{-2}$. We deduce the following:

$$\begin{aligned}
\mathrm{Tr}\left(\mathcal{B}_\alpha\left(u^{-1}\gamma^{-1}\right)\right) &= \mathrm{Tr}\left(\frac{\left(u^{-1}\gamma^{-1}\right)^2 + u^{-1}\gamma^{-1}}{u^{-1}+1}\right)\\
&= \mathrm{Tr}\left(\frac{\left(u^{-1}\gamma^{-1}\right)^2}{u^{-1}+1} + \frac{\left(u^{-1}\gamma^{-1}\right)^2}{u^{-2}+1}\right)\\
&= \mathrm{Tr}\left(\frac{(u^{-3}+u^{-2})\gamma^{-2} + u^{-2}\gamma^{-2}}{u^{-2}+1}\right)\\
&= \mathrm{Tr}\left(\frac{\gamma^{-2}}{u+u^3}\right) .
\end{aligned}$$

The condition $u^{-1}/\gamma \notin \{0, 1, \gamma^{-1}\}$ is equivalent to $u \notin \{0, \gamma^{-1}, 1\}$, the same set as before. This proves the corollary. □

We now show that the last condition in Corollary 14.4.1 can hold only if $n = 3$. In other words, APN generalized butterflies exist for $n = 3$ only. The proof relies on the following lemma.

**Lemma 14.4.3.** *[BRS67] The cubic equation $x^3+ax+b = 0$, where $a \in \mathbb{F}_{2^n}$ and $b \in \mathbb{F}_{2^n}^*$ has a unique solution in $\mathbb{F}_{2^n}$ if and only if $\mathrm{Tr}(a^3/b^2) \neq \mathrm{Tr}(1)$.*

**Proposition 14.4.1.** *Let $n > 1$ be an odd integer, and $\lambda \in \mathbb{F}_{2^n}^*$. If*

$$\mathrm{Tr}\left(\frac{\lambda^2}{x+x^3}\right) = 1,\ \forall\, x \notin \{0, 1, \lambda\} \ , \tag{14.16}$$

*then $n = 3$.*

*Proof.* Consider $z$ in $\mathbb{F}_{2^n}^*$ with $\mathrm{Tr}(z) = 0$. Then, there exists a unique $x \in \mathbb{F}_{2^n} \setminus \mathbb{F}_2$ such that

$$\frac{1}{x^3+x} = z \ .$$

Indeed, since $z \neq 0$, this equivalently means that

$$x^3 + x + \frac{1}{z} = 0 \ .$$

We know from Lemma 14.4.3 that this equation has a unique solution when $\mathrm{Tr}(z^2) = \mathrm{Tr}(z) = 0$. Let us define $z_\lambda$ as

$$z_\lambda = \frac{1}{\lambda^3+\lambda} \ ,$$

and

$$\mathcal{Z} = \{z \in \mathbb{F}_{2^n}^* \setminus \{z_\lambda\} : \mathrm{Tr}(z) = 0\} \ .$$

Obviously, $\mathcal{Z}$ is either a hyperplane without 0 or a hyperplane without 0 and $z_\lambda$ (depending on the value of $\mathrm{Tr}(z_\lambda)$). Then, Condition (14.16) implies that, for any $z \in \mathcal{Z}$,

$$\mathrm{Tr}(\lambda^2 z) = 1 \ .$$

Suppose that $n \geq 5$. Then, $\mathcal{Z}$ contains at least $(2^{n-1} - 2) \geq 14$ elements and there exists at least two distinct elements $z_0$ and $z_1$ in $\mathcal{Z}$ such that $z_0 + z_1 \in \mathcal{Z}$. Therefore, these two elements must satisfy

$$\mathrm{Tr}(\lambda^2 z_0) = \mathrm{Tr}(\lambda^2 z_1) = \mathrm{Tr}(\lambda^2 (z_0 + z_1)) = 1$$

which is impossible since

$$\mathrm{Tr}(\lambda^2 (z_0 + z_1)) = \mathrm{Tr}(\lambda^2 z_0) + \mathrm{Tr}(\lambda^2 z_1)$$

When $n = 3$, the situation is different since the condition may be satisfied when $\mathcal{Z}$ contains 2 elements only, i.e. when $\mathrm{Tr}(z_\lambda) = 0$. □

#### 14.4.2.3 On the Feistel Case

In the case when $\alpha = \beta = 1$, the generalized butterfly is equivalent to a 3-round Feistel network with round functions $x \mapsto x^3$, $x \mapsto x^{1/3}$ and $x \mapsto x^3$. Theorem 4 in [LW14b] shows that, in this special case, the difference distribution table of the corresponding butterflies does not contain any 2. In other words, the number of solutions $(x, y)$ of

$$\begin{cases} R(x,y) + R(x+a, y+b) = c \\ R(y,x) + R(y+b, x+a) = d \end{cases}$$

for any tuple $(a, b, c, d)$ of $\mathbb{F}_{2^n}$ with $(a, b) \neq (0, 0)$ is either 0 or 4. We now give an alternative proof of this result.

**Proposition 14.4.2.** *[LW14b, Theorem 4] For $\alpha = \beta = 1$, the difference distribution tables of the butterflies $\mathsf{V}_{1,1}$ and $\mathsf{H}_{1,1}$ contain the values 0 and 4 only.*

*Proof.* As in the proof of Theorem 14.4.3, we have to count the number of solutions of System (14.5), which simplifies to

$$\begin{cases} (a+b)x^2 + (a+b)^2 x + ay^2 + a^2 y = c' \\ bx^2 + b^2 x + (a+b)y^2 + (a+b)^2 y = d' \ . \end{cases} \tag{14.17}$$

- If $a = 0$, the first line of the system equals $b(x^2 + bx) = c'$ which has either 0 or 2 solutions, $x_0$ and $x_1$ (recall that $a$ and $b$ cannot simultaneously vanish). The second line of the system can be rewritten as

  $$b\left((x+y)^2 + b(x+y)\right) = d'$$

  which has either 0 or 2 solutions, implying $y \in \{x + z_0, x + z_1\}$. Therefore, if the first line has two solutions, the second one has either 0 or 4 solutions. The case $b = 0$ is similar.

- If $a = b$, the system is composed of two independent degree-2 equations, one in $x$ and the second one in $y$. If one of these equations has no solution, then the whole system does not have any solution. Otherwise, each equation has two solutions, and the system has 4 solutions.

- If $ab(a+b) \neq 0$. Then, the first line $\ell_1$ of System (14.17) can be replaced by $b\ell_1 + (a+b)\ell_2$, leading to

$$\begin{cases} ab(a+b)x + (ab+a^2+b^2)y^2 + (a^3+b^3+ab^2)y = \varepsilon \\ bx^2 + b^2x + (a+b)y^2 + (a+b)^2y = d' \ . \end{cases} \tag{14.18}$$

We now multiply the second line by $a^2b(a+b)^2$ and replace $ab(a+b)x$ by the value given by the first line and get

$$\begin{aligned} y^4(ab+a^2+b^2)^2 &+ y^2\left(a^6+b^6+ab^5+a^5b+a^3b^3\right) \\ &+ \ y\left(a^6b+a^5b^2+a^4b^3+a^3b^4+a^2b^5+ab^6\right) \ = \ \varepsilon' \ . \end{aligned}$$

Replacing $y' = by$, we equivalently obtain

$$Uy'^4 + Vy'^2 + Wy' = b^{-8}\varepsilon' \tag{14.19}$$

where the coefficients $U$, $V$ and $W$ depend on $e = a/b$:

$$\begin{aligned} U &= e^4+e^2+1 \ = \ (e^2+e+1)^2 \\ V &= e^6+e^5+e^3+e+1 \ = \ (e^2+e+1)^3 \\ W &= e^6+e^5+e^4+e^3+e^2+e \ = \ U+V \ . \end{aligned}$$

Lemma 14.4.2 then applies. Clearly $U \neq 0$ since the polynomial $X^2+X+1$ has no root in $\mathbb{F}_{2^n}$ when $n$ is odd. Also, $U \neq V$, otherwise $e^2+e+1 = 1$ which is not possible since the cases $e \in \{0,1\}$ (i.e., $a = 0$ or $a = b$) have been excluded. Then, Equation (14.19) has two solutions only if $\mathrm{Tr}(V/U) = 0$. But,

$$\mathrm{Tr}\left(\frac{V}{U}\right) = \mathrm{Tr}(e^2+e+1) = \mathrm{Tr}(1) = 1 \ ,$$

implying that Equation (14.19) has 0 or 4 solutions $y_i$, and each $y_i$ leads to a unique value of $x$. Therefore, the whole system has either 0 or 4 solutions.

□

### 14.4.3 Algebraic Degree of Generalized Butterflies

**Theorem 14.4.5.** *Let $\alpha$ and $\beta$ be two nonzero elements in $\mathbb{F}_{2^n}$. The generalized open butterfly $\mathsf{H}_{\alpha,\beta}$ has an algebraic degree equal to $n$ or $n+1$. It is equal to $n$ if and only if*

$$(1+\alpha\beta+\alpha^4)^3 \ = \ \beta(\beta+\alpha+\alpha^3)^3.$$

*The closed butterfly $\mathsf{V}_{\alpha,\beta}$ has algebraic degree 2.*

The condition $(1+\alpha\beta+\alpha^4)^3 = \beta(\beta+\alpha+\alpha^3)^3$ can alternatively be written $Z(\alpha,\beta) = 0$, where:

$$Z(\alpha,\beta) \ = \ \beta^4+\alpha\beta^3+\alpha(\alpha+1)^6\beta+(1+\alpha)^{12}.$$

Furthermore, $Z$ can be factorized as follows:

$$\begin{aligned} Z(\alpha,\beta) &= \beta^4+\alpha\beta^3+\alpha(\alpha+1)^6\beta+(1+\alpha)^{12} \\ &= \left(\beta^2+(1+\alpha)^6\right)\left(\beta^2+\alpha\beta+(1+\alpha)^6\right) \\ &= \left(\beta^2+(1+\alpha)^6\right)\left(1+\alpha\beta+\alpha^4+(\beta+\alpha+\alpha^3)^2\right) . \end{aligned}$$

Hence, if $\beta \neq (1+\alpha)^3$ then $Z(\alpha,\beta) = 0$ if and only if $1 + \alpha\beta + \alpha^4 = (\beta + \alpha + \alpha^3)^2$. It follows that $Z(\alpha,\beta)$ is equal to 0 when $\beta = (1+\alpha)^3$ and, if $\mathrm{Tr}(\alpha^{-1}) = 1$, for two additional values of $\beta$. This includes the Feistel case, when $\alpha = \beta = 1$.

*Proof of Theorem 14.4.5.* Obviously, $\mathsf{V}_\alpha(x,y)$ has algebraic degree 2. We then focus on the generalized open butterfly $\mathsf{H}_{\alpha,\beta}$. The right side of the output of such an open butterfly is equal to $(x+\beta y^3)^{1/3} + \alpha y$, where $(x,y)$ is the input. We deduce from Theorem 1 of [KS12] (or equivalently from Proposition 5 of [Nyb94]) that the inverse of 3 modulo $(2^n - 1)$ for odd $n$ is

$$1/3 \equiv \sum_{i=0}^{(n-1)/2} 2^{2i} \mod (2^n - 1),$$

which implies in particular that the algebraic degree of $x \mapsto x^{1/3}$ is equal to $(n+1)/2$. We deduce from this expression that the function $t(x,y) = (x + \beta y^3)^{1/3}$ is equal to $\prod_{i=0}^{(n-1)/2} (x+\beta y^3)^{2^{2k}}$. This sum can be developed as follows:

$$t(x,y) = (x+\beta y^3)^{1/3} = \sum_{J \subseteq [0,(n-1)/2]} \underbrace{\prod_{j \in J} \beta^{2^{2j}} y^{3 \times 2^{2j}}}_{\deg < 2|J|} \underbrace{\prod_{j \in \overline{J}} x^{2^{2j}}}_{\deg = (n+1)/2 - |J|} ,$$

where $\overline{J}$ is the complement of $J$ in $[0,(n-1)/2]$, i.e. $J \cap \overline{J} = \emptyset$ and $J \cup \overline{J} = [0,(n-1)/2]$. The algebraic degree of each term in this sum is at most equal to $|J| + (n+1)/2$. Thus, if $|J| < (n-1)/2$, then the degree of the corresponding term is smaller than $n$. If $\overline{J} = \emptyset$ then the corresponding term is equal to $\beta^{1/3} y$ and has degree 1. If $\overline{J} = \{j\}$ for some $j$, then the term is equal to

$$x^{2^{2j}} \times \beta^{1/3} y \times (\beta y^3)^{2^n - 1 - 2^{2j}} = \beta^{1/3 - 2^{2j}} \times x^{2^{2j}} \times y^{(2^n-1)-(2^{2j+1}+2^{2j}-1)} .$$

If $j \neq (n-1)/2$, then its algebraic degree is

$$1 + n - wt(2^{2j+1} + 2^{2j} - 1) = n - 2j .$$

If $j = (n-1)/2$, then the term (omitting the constant factor) is equal to

$$x^{2^{n-1}} \times y \times y^{2^n - 1 - (2^n - 2^{n-1})} = x^{2^{n-1}} y^{2^{n-1}-1} .$$

and has degree $n$. Therefore, $t(x,y)$ has two terms of degree $n$, corresponding to $j = 0$ and $j = (n-1)/2$ namely

$$m_0(x,y) = \beta^{-2/3} x y^{2^n - 3} \text{ and } m_1(x,y) = \beta^{(2^{n-1}-1)/3} x^{2^{n-1}} y^{2^{n-1}-1} .$$

Thus, the right side of the output has an algebraic degree equal to $n$.

The left side is equal to

$$L(x,y) = \Big(y + \alpha\big((x+\beta y^3)^{1/3} + \alpha y\big)\Big)^3 + \beta\big((x+\beta y^3)^{1/3} + \alpha y\big)^3,$$

which we can re-write using the function $t(x,y) = (x+\beta y^3)^{1/3}$ as

$$L(x,y) = \big((\alpha^2+1)y + \alpha t(x,y)\big)^3 + \beta\big(t(x,y) + \alpha y\big)^3,$$

which we expand into

$$\begin{aligned} L(x,y) &= t(x,y)^3(\alpha^3+\beta) \ + \ y^3\Big((\alpha^2+1)^3+\alpha^3\Big) \\ &\quad + \ yt(x,y)^2\Big((\alpha^2+1)\alpha^2+\beta\alpha\Big) \ + \ y^2t(x,y)\Big((\alpha^2+1)^2\alpha+\beta\alpha^2\Big). \end{aligned}$$

The terms on the first line have degree at most 3. Let us focus on those of the second line and denote their sum $L'(x,y)$. First, we can simplify this expression as follows:

$$\frac{L'(x,y)}{\alpha} \ = \ C_0yt(x,y)^2 \ + \ C_1y^2t(x,y)$$

where $C_0 = (\beta+\alpha+\alpha^3)$ and $C_1 = (1+\alpha\beta+\alpha^4)$.

Since $t(x,y)$ has algebraic degree $n$, we deduce that $L'(x,y)$ — and thus the left side of the output of $\mathsf{H}_{\alpha,\beta}$ — has algebraic degree at most $(n+1)$, while the whole function has degree at least $n$ because of the right side. Moreover, this upper bound is reached if and only if the terms of degree $(n+1)$ in $L'(x,y)$ do not cancel each other. The only terms in $L'(x,y)$ which may have degree $(n+1)$ correspond to terms of degree $n$ in $t(x,y)$, namely (omitting the constant factors):

$$\begin{aligned} y^2m_0(x,y) &= xy^{2^n-1}, & y^2m_1(x,y) &= x^{2^{n-1}}y^{2^{n-1}+1}, \\ ym_0(x,y)^2 &= x^2y^{(2^n-1)-3}, & ym_1(x,y)^2 &= xy^{2^n-1} \ . \end{aligned}$$

Only the first and the last terms actually have degree $(n+1)$. Therefore, the term of degree $(n+1)$ in $L'(x,y)$ is:

$$\begin{aligned} C_0ym_1(x,y)^2+C_1y^2m_0(x,y) &= xy^{2^n-1}\left(C_0\beta^{-1/3}+C_1\beta^{-2/3}\right) \\ &= xy^{2^n-1}\beta^{-1/3}\left(C_0+C_1\beta^{-1/3}\right) \ . \end{aligned}$$

It follows that $\mathsf{H}_{\alpha,\beta}$ has algebraic degree $(n+1)$ if and only if

$$\beta C_0^3 \neq C_1^3 \ .$$

□

Part III

# On Purposefully Hard Cryptography

# Contents of this Part

Chapter 15

# Symmetric and Asymmetric Hardness

Intuitively, a cryptographic primitive should have an implementation as efficient as possible. As discussed in the context of lightweight cryptography in Chapter 2 (p. 29), the efficiency of an algorithm is defined along three axes: time, memory and code size.

However, there are scenarios in which it is desirable to use primitives that are purposefully inefficient for one or several of these metrics. This can be done to slow down the attackers, provide different levels of service to privileged and non-privileged users, adjust cost of operation in proof-of-work schemes, etc.. The simplest illustration of functions designed to be time consuming to compute is that of key derivation functions (KDF). A KDF is typically built by iterating a one-way function (ex. a cryptographic hash function), multiple times. Such functions are intended to prevent an adversary from brute-forcing a small set of keys (corresponding to, say 12 letter strings) by making each attempt very costly.

Time, however, is not the only form of hardness for which an artificial complexity increase can be beneficial. Memory-hardness was one of the design goals of the winner of the Password Hashing Competition, Argon2 [BDK16], the aim being to prevent hardware optimization of the primitive. As another example, one research direction in white-box cryptography is nowadays focusing on designing block ciphers such that the code implementing them is very large in order to prevent duplication and distribution of their functionality [BBK14, BI15, FKKM16, BIT16, BKR16]. In this case, the aim could be to implement some form of Digital Rights Management or to prevent the exfiltration of a block cipher key by malware.

Since hardness is an inherently expensive property, there are cases where a trapdoor could be welcome. This is the case for the most recent weak white-box block ciphers [BBK14, BI15, FKKM16]: while the white-box implementation requires a significant code size, there exists a functionally equivalent implementation which is much smaller but cannot be obtained unless a secret is known. That way, two classes of users are created: those who know the secret and can evaluate the block cipher efficiently and those who do not and thus are forced to use the code-hard implementation. The different forms of hardness, their applications and instances from the literature are summarized in Table 15.1.

Regardless of the form of hardness, the aim of the designer of a hard primitive is to prevent an attacker from by-passing this complexity, even if the attacker is allowed

| | Time | Memory | Code size |
|---|---|---|---|
| Applications | KDF,<br>time-lock | Password hashing,<br>egalitarian computing | White-box crypto<br>big-key encryption |
| Symmetrically hard functions | PBKDF2 [Kal00] | Argon2 [BDK16],<br>Balloon Hashing [BCS16] | XKEY2 [BKR16],<br>WHALE (Sec. 15.4.2) |
| Asymmetrically hard functions | RSA-lock [RSW96]<br>SKIPPER (Sec. 15.4.1) | DIODON [BP17] | White-box block ciphers<br>[BBK14, BI15, FKKM16, BIT16] |

Table 15.1: Six types of hardness and their applications.

significant precomputation time. Informally, a user cannot decrease the hardness of the computation below a certain threshold. We took inspiration from the formal definitions of hardness used in white-box cryptography to build a unified framework to study and design cryptographic algorithms with all forms of hardness.

This chapter is structured as follows. First, Section 15.1 provides more details about the different forms of hardness and their current usage. Then, Section 15.2 presents our generic approach for dealing with all forms of hardness at once. Using this framework, we deduce practical modes of operation for building hard block ciphers and hard sponges which are described in Section 15.3. Our concrete proposals, called SKIPPER and WHALE, are introduced in Section 15.4.

## 15.1 Enforcing Hardness

In this section, we argue that many recent ideas in symmetric cryptography can be interpreted as particular cases of a single general concept. The aim of several a priori different research areas can be seen as imposing the use of important resources for performing basic operations or in other words, *bind an operation to a specific form of hardness*. We restrict ourselves to the basic case of a well-defined function mapping each input to a unique output. It means in particular that protocols needing several rounds of communication or randomized algorithms which may return any of the many valid solutions to a given problem such as HashCash (see below) are out of our scope.

The three main metrics for assessing the efficiency of an algorithm are its time complexity, its RAM usage and its code size. As we explain below, different lines of research in symmetric cryptography can be interpreted as investigating the design of algorithms such that one of these metrics is abnormally high and cannot be reduced while limiting the impact on the other two as much as possible.

*Time-hardness* is discussed in Section 15.1.1, *memory-hardness* in Section 15.1.2 and *code-hardness* in Section 15.1.3. Finally, in Section 15.1.4, we present the general notion of *asymmetric hardness*.

It is also worth mentioning that the three forms of hardness are not completely independent of another. For example, due to the number of memory accesses needed in order for a function to be memory-hard, it cannot be arbitrarily fast.

### 15.1.1 Time Hardness

While the time efficiency of cryptographic primitives is usually one of the main design criteria, there are cases where the opposite is needed. That is, algorithms which can be made arbitrarily slow in a controlled fashion.

One of the most simple approaches is the one used for instance by the key derivation function PBKDF2 [Kal00]. This function derives a cryptographic key from a salt and a password by iterating a hash function multiple times, the aim being to frustrate brute-force attacks. Indeed, while the password may be from a space small enough to be brute-forced, evaluating the key derivation function for each possible password is made infeasible by its time-hardness.

Somewhat similarly, proofs-of-work such as HashCash — used by Bitcoin [Nak08] — search for at least one of many solutions to a given problem. The hardness in this case comes from luck. Miners must find a value such that the hash of this value and the previous block satisfies the difficulty constraint. However, the subset of such valid values is sparse and thus miners have to try many random ones. Two different miners may find two different but equally valid values. Because of this randomness, such puzzles are out of our scope. In this chapter, we only consider functions which are equally hard to evaluate on all possible inputs, not puzzles for which finding a solution is hard *on average*.

Furthermore, in order to mitigate the impact of adversaries with vast number of processors at their disposal, we consider sequential time-hardness. Using a parallel computer should not help an attacker to evaluate the function much more quickly. Formalizing parallel time hardness the way we do it for sequential time-hardness is left as future work.

Overall, the goal of time-hardness is to prevent an adversary from computing a function in a time significantly smaller than the one intended. In other words, it must be impossible to compress the amount of time needed to evaluate the function on a random input.

### 15.1.2 Memory Hardness

Informally, a function is memory-hard if even an optimized implementation requires a significant amount of memory. For each evaluation, a large amount of information is written and queried throughout the computation. A function requiring large amounts of RAM for its computation prevents an attacker from building ASICs filled with a huge number of cores for parallel computations.

This implies that memory-hard functions make good password hashing functions and proofs-of-work. One of the first to leverage this insight was the hash function scrypt [Per09] which was recently formally proved to be memory-hard in [ACP$^+$16]. It gave rise to several other memory-hard algorithms such as the password hashing competition winner Argon2 [BDK16] as well as the more recent Balloon Hashing [BCS16] and Equihash [BK16b]. Those can be used as building blocks to create memory-hard proofs-of-work which can offset the advantage of cryptocurrency miners using dedicated ASICs.

The idea of using memory-hard functions for general purpose computations was further explored in the context of *egalitarian computing* [BK16a]. Similarly, *proofs of space* [DFKP15, ABFG14] are protocols which cannot be run by users if they are not able to both read and write a large amount of data. However, those are interactive protocols and not functions.

The recent research investigating memory-hardness has lead to several advances in our understanding of this property. For example, the difference between *amortized* and *peak* memory hardness was highlighted in [AS15]. However, for the sake of clarity, we restrict ourselves to peak memory hardness; i.e. that at least at one point in the evaluation of a function, a large amount of memory is necessary.

### 15.1.3 Code Hardness

First of all, let us clarify the distinction we make between *memory* and *code*-hardness. With code-hardness, we want to increase the space needed to store information that is needed to evaluate a function on all possible inputs. However, the information itself does not depend on said input. During evaluation of the function, it is only necessary to *read* the memory in which the code is stored. In contrast, memory-hardness deals with the case where we need to store a large amount of information which depends on the function input and which is thus different during each evaluation of the function. In this case, one must be able to both read *and write* to the memory. Furthermore, in a typical code-hard function, only a small fraction of the whole information stored in the implementation is read during each call to the function. On the other hand, if a memory-hard function uses $M$ bytes of memory, then all of those bytes will be written and read at least once.

Code-hardness is very close to one of the goals of the most recent white-box block cipher proposals such as ASASA [BBK14], Space [BI15], SPNBox [BIT16] and PuppyCipher/Hound [FKKM16] as well as the more general definition in [DLPR14]. This notion is formalized under different names in each paper: weak white-box encryption for Biryukov *et al.*, $(M,z)$-space hardness for Bogdanov *et al.* or incompressibility for both Fouque *et al.* and Delarablée *et al.*. In all cases, the aim is the same: the block cipher implementation must be such that it is impossible to write a functionally equivalent implementation with a smaller code. This stands in contrast to *strong* white-box cryptography (as defined in [BBK14]) where inverting a function given its white-box implementation should be impossible. We do not consider this case in this chapter.

As was pointed out in [FKKM16], what we call code-hardness is also the goal of so-called *big-key encryption*. The XKEY2 scheme introduced in [BKR16] achieves this goal: it uses a huge table and a nonce to derive a key of regular size (say, 128 bits) to be used in a standard encryption algorithm, e.g. a stream cipher. Bellare *et al.* show that even if an attacker manages to obtain half of the huge table, i.e. half of the code needed to implement the scheme, then they are still unable to compute the actual encryption key with non-negligible probability. Using our terminology, XKEY2 can be seen as a code-hard key derivation function. The concept of *proof of storage* can also be interpreted as a particular type of code-hard protocols. Indeed, in such algorithms, challengers must prove that they have stored a given file.

### 15.1.4 Asymmetric Hardness

In this section, we discuss the concept of *asymmetric hardness* which introduces two classes of users. *Common users* evaluate a hard function but *privileged users*, who know a secret key, can evaluate a functionally equivalent function which is *not* hard.

#### 15.1.4.1 Asymmetric Code-Hardness

The most recent White-Box block ciphers such as ASASA [BBK14], Space [BI15], the PuppyCipher [FKKM16] and SPNbox [BIT16] can be seen as providing asymmetric code-hardness. Indeed, while the first aim of these algorithms is to provide regular code-hardness, referred to as "space-hardness" for the former and "incompressibility" for the latter, they both allow the construction of far more code-efficient implementations. For both Space and the PuppyCipher, the idea is to compute a large table containing the encryptions of the first $2^t$ integers with Aes-128 for $t$ in $\{8, 16, 24, 32\}$. These tables are then used as the code-hard part of the encryption which cannot be compressed because doing so would require a break of the Aes. However, a user knowing the 128-bit Aes key can get rid of these tables and merely recompute the entries needed on the fly, thus drastically decreasing the code-hardness of the implementation.

In fact, both constructions can be seen as structures intended to turn an asymmetrically code-hard function into an asymmetrically code-hard block cipher. In both cases, the asymmetrically code-hard function consists of the evaluation of the Aes on a small input using either the secret key, in which case the implementation is not code-hard, or using only the public partial codebook which, because of its size, is code-hard.

#### 15.1.4.2 Asymmetric Time-Hardness

While the asymmetry of its hardness was not insisted upon, there is a known asymmetrically time-hard function, which we call rsa-lock. It was proposed as a time-lock in [RSW96], that is, a function whose output cannot be known before a certain date.

It is based on the Rsa cryptosystem [RSA78]. It consists of iterating squarings in a Rsa modular ring: a user who does not know the prime factor decomposition of the modulus $N$ must perform $t$ squarings while a user who knows that $N = pq$ can first compute $e = 2^t \mod (p-1)(q-1)$ and then raise the input to the power $e$. If $t$ is large enough, the second approach is much faster.

#### 15.1.4.3 Asymmetric Memory-Hardness

The only asymmetrically memory-hard function we are aware of is Diodon which we very recently[1] presented in an `eprint` submission [BP17]. It is a variant of the memory-hard function `scrypt` [Per09] which replaces some of the hash function calls used in the original design by squarings in an Rsa modular group. More precisely, given an input $x$, it first fills a large array $V$ by setting $V_0 = x$ and $V_{i+1} = V_i^{2^\eta} \mod N$ where $\eta$ is a parameter of the computation and $N = qq'$ is an Rsa modulus. It then goes through another loop querying random entries in this table. Basic users are forced to store the whole array while privileged users can recompute each $V_i$ directly by first reducing $2^{\eta \times (i-1)}$ modulo $(q-1)(q'-1)$ and then raise $x$ to the resulting exponent, as would be done in the Rsa-based time-lock presented above in Section 15.1.4.2.

However, unlike Argon2, Diodon only gives a linear penalty to users trying to reduce its memory complexity. It is possible to evaluate Diodon using $c$ times less

[1] We designed it between the defence of this thesis and the submission of its final version. Therefore, I only mention it in this manuscript.

memory at the cost of a multiplication by $c$ of the time complexity. Thus, the following problem remains open.

**Open Problem 15.1.1.** *Is it possible to design an asymmetrically memory-hard (but code-efficient) function with superlinear penalties?*

## 15.2 A Generic Framework

As we have seen, all the techniques presented in the sections above are intended to enforce some form of computational hardness. In this section, we present a unified framework for building any symmetric algorithm with some form of guaranteed hardness. We describe our aim in Section 15.2.1 and our design strategy with the generic hardness definition it is based on in Section 15.2.2. Our constructions need small functions with the intended hardness type to be bootstrapped. We provide examples of those in Section 15.2.3.

### 15.2.1 Design Strategy

Our aim is to design a general approach to build any cryptographic primitive with any form of hardness. To achieve this, we will build modes of operations allowing us to combine secure cryptographic primitives, such as the Aes [DR02] block cipher or the Keccak sponge [BDPVA09], with small functions called *plugs*. These plugs are simple functions with the desired form of hardness.

Our modes of operations, which are presented in Section 15.3, are all based on the same principle: ensuring that enough plug calls with a non-predictable input are performed so as to guarantee that, with overwhelming probability, an adversary cannot bypass all plug evaluation. This ensures that the full complexity of a plug evaluation is paid for at least once.

Indeed, regardless of the hardness form considered, the strategy of a generic adversary will always be the same. Provided that the plugs are indeed hard to evaluate, the only strategy allowing an adversary to bypass their hardness consists of storing a (feasibly) large number of plug outputs in a database and then querying them.

If $2^p$ outputs of a plug $P : \{0,1\}^t \rightarrow \{0,1\}^v$ have been stored, the plug can be evaluated successfully *without paying for its hardness* with probability $2^{p-t}$.

An alternative strategy using the same space consists of storing $2^p(v/d)$ partial outputs of length $d$. In this case, the success probability becomes $2^{p-t}(v/d) \times 2^{d-v}$: the input is partially known with a higher probability $2^{p-t}(v/d)$ but $(v-d)$ bits of the output remain to be guessed. This method is $(v/d) \times 2^{d-v}$ more efficient than the basic one but, for $1 \geq d \geq v$, this quantity is always smaller than one. The strategy consisting of storing full outputs is therefore the best.

However, if the output size of the plug is small enough, it might be efficient enough for the adversary to directly guess the whole output. The probability that an adversary merely guessing the output of the plug gets it right is $2^{-v}$.

Our aim is therefore to guard our constructions from the adversary defined below. Protecting our structure against them is sufficient to reduce their hardness to that of the plug they use.

**Definition 15.2.1** ($2^p$-adversary)**.** *Let $f$ be a time-, memory- or code-hard function. A $2^p$-adversary is an adversary trying to generate a function $f'$ which does not have the hardness of $f$ but which does not have access to more memory than needed to store $2^p$ outputs of $f$.*

A $2^p$-adversary can perform more than $2^p$ calls to the function it is trying to approximate when generating $f'$, although $f'$ itself cannot have access to more than $2^p$ of those. Still, $f'$ can perform additional computations using the information stored during its generation.

However, the computational power of this adversary is not unbounded. More precisely, in the remainder of this chapter, we consider that all $2^p$-adversaries cannot perform more than $2^{100}$ operations. This means for example that recovering a 128-bit AES key is out of their reach. We are not interested in guarding ourselves against computationally unbounded adversaries.

## 15.2.2 Theoretical Framework

### 15.2.2.1 Generic Symmetric Hardness

We are now ready to formally define hardness. We generalize the incompressibility notion from [FKKM16] to all forms of hardness.

**Definition 15.2.2** ($\mathcal{R}$-hardness)**.** *We say that a function $f : \mathcal{X} \rightarrow \mathcal{Y}$ is* $\mathcal{R}$-hard against $2^p$-adversaries *for some tuple $\mathcal{R} = (\rho, u, \epsilon(p))$ with $\rho \in \{Time, Code, \textsc{ram}\}$ if evaluating the function $f$ using less than $u$ units of the resource $\rho$ and at most $2^p$ units of storage is possible only with probability $\epsilon(p)$. More formally, the probability for a $2^p$- adversary to win the* efficient approximation game*, which is described below, must be upper-bounded by $\epsilon(p)$.*

1. *The challenger chooses a function $f$ from a predefined set of functions requiring more than $u$ units of $\rho$ to be evaluated.*
2. *The challenger sends $f$ to the adversary.*
3. *The adversary computes an approximation $f'$ of $f$ which, unlike $f$, can be computed using less than* u *units of the resource $\rho$.*
4. *The challenger picks an input $x$ of $\mathcal{X}$ uniformly at random and sends it to the adversary.*
5. *The adversary wins if $f'(x) = f(x)$.*

*This game is also represented in Figure 15.1. The approximation $f'$ computed by the adversary must be evaluated using significantly less than $u$ units of the resource $\rho$, although the precomputation may have been more expensive.*

In order for this definition to be relevant, the power of the adversary must be estimated. For example, preventing attacks from $2^{512}$-adversaries would most definitely be over engineering and, conversely, preventing attacks from $2^{20}$-adversaries would be useless since such precomputation is always feasible.

Our definition is not the strongest in the sense that it does not encompass e.g. “strong space-hardness” [BI15]. This definition of code-hardness aims at preventing the attacker from encrypting a plaintext *of their choosing*, a far stronger requirement than preventing the encryption of a *random* plaintext.

In the efficient approximation game described above, $f'$ must be less hard than $f$ along the appropriate axis. For example, if $f$ is code-hard then the code implementing $f'$ must be significantly smaller than that implementing $f$, meaning that the game corresponding to code-hardness when $f$ is an encryption algorithm is essentially the

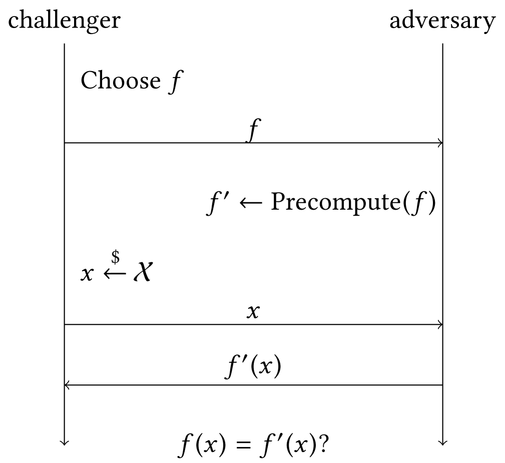

challenger
adversary
Choose $f$
$f$
$f' \leftarrow \text{Precompute}(f)$
$x \xleftarrow{\$} \mathcal{X}$
$x$
$f'(x)$
$f(x) = f'(x)?$

Figure 15.1: The game corresponding to the definition of $(\mathcal{R}, s, \epsilon)$-hardness: $\epsilon$ bounds the probability of success of the adversary and $s$ bounds the $\mathcal{R}$-hardness of $f'$.

same as the one used in the definition of encryption incompressibility [FKKM16]. Indeed, the computation of $f'$ and its use by the adversary corresponds in this case to the computation of the leakage function on the secret large table and its use by the adversary to approximate the original table.

In the case of code-hardness the maximum code size of the implementation of $f'$ must coincide with the power of the $2^p$-adversary. Indeed, the implementation of the approximation $f'$ needs at least enough space to store $2^p$ outputs of the plug.

#### 15.2.2.2 Generic Asymmetric Hardness

This generic definition is easily generalized to encompass asymmetric hardness.

**Definition 15.2.3** (Asymmetric $\mathcal{R}$-hardness)**.** *We say that a function* $f : \mathcal{X} \rightarrow \mathcal{Y}$ *is* asymmetrically $\mathcal{R}$-hard against $2^p$-adversaries *for some tuple* $\mathcal{R} = (\rho, u, \epsilon(p))$ *with* $\rho \in \{\mathit{Time}, \mathit{Code}, \textsc{ram}\}$ *if it is impossible for a* $2^p$*-adversary to win the approximation game of Definition 15.2.2 with probability higher than* $\epsilon(p)$*, unless a secret K is known.*

*If this secret is known, then it is possible to evaluate* $f_K$ *which is functionally equivalent to* $f$ *but does not have its hardness.*

An immediate consequence of this definition is that extracting the secret key $K$ from the description of $f$ must be computationally infeasible. Otherwise, the adversary could simply recover $K$ during the precomputation step, use $f_K$ as their approximation and then win the approximation game with probability 1. This observation is reminiscent of the *unbreakability* notion presented in [DLPR14]

White-box block ciphers are simple example of asymmetrically code-hard functions. This concept can also be linked to the *proof of work or knowledge* presented in [BKZZ16]. It is a proof of work where a solution can be found in a more efficient way if a secret is known.

Asymmetric hardness is a different notion from public key encryption. Indeed, in the latter case, the whole decryption functionality is secret. In our case, the functionality is public. What is secret is a method to evaluate it efficiently.

#### 15.2.2.3 A Counter-Example

The inversion of a one-way function may seem like a natural example of a time-hard function. However, as described below, it may not satisfy the requirements of our definition of $(\text{Time}, u, \epsilon(p))$-hardness.

Let $h : \{0,1\}^{50} \rightarrow \{0,1\}^{50}$ be the function mapping a 50-bit string to the first 50 bits of their SHA-256 digest and let $f$ be the inverse of $h$. In other words, $f$ returns a preimage of a given digest. This function may seem time-hard at first glance as SHA-256 is preimage resistant. More specifically, it might be expected to be about $(\text{Time}, 2^{50}, 2^{-20})$-hard against a $2^{30}$-adversary. However, as is well known, such constructions can be attacked using Hellman's tradeoff [Hel80] in the form of rainbow-tables allowing an attacker to recover a preimage in far less time at the cost of significant but practical pre-computation and storage. If $M$ is the size of this table and $T$ is the time complexity of an inversion using this table then, as explained in [BS00], it must hold that $MT^2 = N^2$ where $N = 2^{50}$ in our case. The failure of $f$ to be time-hard in the sense of Definition 15.2.2 can be seen in the approximation game.

1. The challenger chooses a secure hash function (SHA-256) and sends its description to the adversary.

2. The $2^{30}$-adversary precomputes Hellman-type rainbow tables with in total $2^{30}$ entries using about $2^{50}$ calls to $h$. This adversary chooses $M = 2^{30}$ and $T = N/\sqrt{M} = 2^{35}$.

3. The challenger chooses a random value $x \in \{0,1\}^{50}$ and sends it to the adversary.

4. With high probability, the adversary computes $f(x)$ using their precomputed table in time $T = 2^{35}$ which is $2^{15}$ times smaller than the time needed for brute-force.

Thus, such a function is not time-hard in the sense of Definition 15.2.2.

### 15.2.3 Examples of Plugs

As our modes rely on smaller $\mathcal{R}$-hard function to achieve their goal, we describe an array of such components, one for each hardness goal. A summary of all the plugs we describe, along with what we consider to be their hardness against $2^p$-adversaries, is given in Table 15.2.

While we provide an intuition on why we assume these plugs to have the hardnesses we claim, we do not prove that it is the case.

If the output it is too large to be used in a higher level construction then it is possible to truncate it to $v$ bits. If we denote $T_v$ the function discarding all but the first $v$ bits of output and if $P$ is a plug with a $t$-bit input which is $(\rho, u, \epsilon(p))$-hard against $2^p$-adversaries, then $x \mapsto T_m(P(x))$ is $(\rho, u, \max(\epsilon(p), 2^{p-t}))$-hard against $2^p$-adversaries. Overall, the probability of success of an approximation made by a $2^p$-adversary of a plug mapping $t$ to $v$ bits is lower-bounded by $\max(2^{-v}, 2^{p-t})$.

#### 15.2.3.1 Time-Hard plug

This hardness has been considered in previous works for instance in the context of key stretching and key derivation or for time-lock encryption. In fact, the construc-

| Hardness | Symmetric | Asymmetric |
|---|---|---|
| Time | $\mathsf{IterHash}^t_\eta$<br>$(\text{Time}, \eta, 2^{p-t})$ | $\mathsf{RSAlock}^t_\eta$<br>$(\text{Time}, \eta, 2^{p-t})$ |
| Memory | Argon2<br>$(\textsc{ram}, M/5, 2^{p-t})$ | Diodon<br>$(\textsc{ram}, M/10, 2^{p-t})$ |
| Code | $\mathsf{BigLUT}^t_v$<br>$(\text{Code}, 2^p, 2^{p-t})$ | $\mathsf{BcCounter}^t_v$<br>$(\text{Code}, 2^p, 2^{p-t})$ |

Table 15.2: Possible plugs, i.e. sub-components for our constructions which we assume to be $\mathcal{R}$-hard against $2^p$-adversaries.

tions proposed for each use case can be used to provide time-hardness and asymmetric time-hardness respectively.

**Symmetric Hardness.** $\mathsf{IterHash}^t_\eta$ iterates a $t$-bit hash function on a $t$-bit input block $\eta$ times where $\eta$ must be much smaller than $2^{t/2}$ to avoid issues related to the presence of cycles in the functional graph of the hash function. If we denote by $H$ the hash function used, then $\mathsf{IterHash}^t_\eta(x) = H^\eta(x)$. Evaluating this function requires at least $\eta$ hash function calls and, provided that the hash function iterated is cryptographically secure, it is impossible for an adversary to guess what the output is after $\eta$ iterations with probability higher than $2^{-t/2}$.

We consider that this function is $(\text{Time}, \eta, 2^{p-t})$-hard against $2^p$-adversaries, as long as $p \ll t/2$.

**Asymmetric Hardness.** $\mathsf{RSAlock}^t_\eta$ is a function performing $\eta$ squaring in a RSA modular ring of size $N = qq' \approx 2^t$, where $q$ and $q'$ are secret primes. Using these notations, $\mathsf{RSAlock}^t_\eta(x) = x^{2^\eta} \mod N$. The common user therefore needs to perform $\eta$ squarings in the modular ring.

However, a user who knows the prime decomposition of the RSA modulo can first compute $e = 2^\eta \mod (q-1)(q'-1)$ and thus compute $\mathsf{RSAlock}^t_\eta(x) = x^e \mod N$. Furthermore, such a user can also use the Chinese remainder theorem to further speed up the computation which increases their advantage over common users. Thus, as long as $t > n$, the privileged user has an advantage over the common. We consider that $\mathsf{RSAlock}^t_\eta$ is asymmetrically $(\text{Time}, \eta, 2^{p-t})$-hard against $2^p$-adversaries.

#### 15.2.3.2 Memory-Hard plug

Several recent functions are intended to provide memory-hardness. The main motivation was the Password Hashing Competition (PHC) which favored candidates enforcing memory-hardness to thwart the attacks of adversaries using ASICs to speed up password cracking.

**Symmetric Hardness.** The winner of the PHC competition, Argon2 [BDK16], uses $M$ bytes of memory to hash a password, where $M$ can be chosen by the user.

It was designed so that an adversary trying to use less than $M/5$ bytes of memory would have to pay a significant increase in time-hardness. Using our definition, if $t$ is the size of a digest (this quantity can also be chosen by the user) and $v$ is the size of the input, then Argon2 is about $(\textsc{ram}, M/5, 2^{p-t})$-hard against $2^p$-adversaries as long as enough passes are used to prevent ranking and sandwich attacks [AB16a, AB16b] and as long as $2^{p-t} > 2^{-v}$.

The construction of memory-hard functions is a very recent topic. Only a few such functions are known, which is why Argon2 is far more complex than the other plugs proposed in this section. It is an interesting research problem to build a simpler memory hard function with the relaxed constraint that it might be cheap to compute on a part of its domain, a flaw which would easily be factored into the $\epsilon(p)$ probability.

**Asymmetric Hardness.** Diodon is, to the best of our knowledge, the only possibility for asymmetric memory-hardness. Much like Argon2, we consider that it is $(\textsc{ram}, M/10, 2^{p-t})$-hard against $2^p$-adversaries for basic users, although the penalty for decreasing the memory complexity is only linear in the case of Diodon.

#### 15.2.3.3 Code-Hard plug

As explained in Section 15.1.3, the main goals of code-hardness are white-box and big-key encryption. The structures used for both purposes rely on the same building block, namely a large look-up table where the entries are chosen uniformly at random or as the encryption of small integers. The former, $\mathsf{BigLUT}_v^t$, is code-hard. The latter, $\mathsf{BcCounter}_v^t$, is asymmetrically code-hard. Furthermore, an identical heuristic can be applied to both of them to increase the input size of the plug while retaining a practical code size. It is described at the end of this section.

**Symmetric Hardness.** $\mathsf{BigLUT}_v^t$ uses a table $K$ consisting in $2^t$ entries, each being a $v$-bit integer picked uniformly at random. Evaluating $\mathsf{BigLUT}_v^t$ then consists simply in querying this table: $\mathsf{BigLUT}_v^t$ is the function mapping a $t$-bit integer $x$ to the $v$-bit integer $K[x]$.

This function is $(\text{Code}, 2^p, 2^{p-t})$-hard against $2^p$-adversaries. Indeed, an adversary who has access to $2^p$ outputs of the function cannot evaluate it efficiently on a random input with probability more than $2^{p-t}$. Simply guessing the output succeeds with probability $2^{-v}$ which is usually much smaller than $2^{p-t}$. Thus, we consider that $\mathsf{BigLUT}_v^t$ is $(\text{Code}, 2^p, 2^{p-t})$-hard against $2^p$-adversaries.

**Asymmetric Hardness.** $\mathsf{BcCounter}_v^t$ is the function mapping a $t$-bit integer $x$ to the $v$-bit block $E_k(0^{v-t}||x)$, where $E_k$ is a $v$-bit block cipher with a secret key $k$ of length at least $v$. A common user would be given the codebook of this function as a table of $2^t$ integers while a privileged user would use the secret key $k$ to evaluate this function.

The hardness of $\mathsf{BcCounter}_v^t$ is the same as that of $\mathsf{BigLUT}_v^t$ for a common user. The contrary would imply the existence of a distinguisher for the block cipher, which we assume does not exist. However, a privileged user with knowledge of the secret key used to build the table can bypass this complexity.

Furthermore, as the key size is at least as big as the block size in modern block ciphers, an adversary guessing the key is not more efficient than one who merely guesses the output of the cipher. Thus, we consider that $\mathsf{BigLUT}_v^t$ is asymmetrically $(\text{Code}, 2^p, 2^{p-t})$-hard.

**Increasing the input size.** Both $\mathsf{BigLUT}^t_v$ and $\mathsf{BcCounter}^t_v$ have a low input size and leave a fairly high success probability for an attacker trying to win the efficient approximation game without using a lot of resource. An easy way to work around this limitation is to use $\ell > 1$ distinct instances of a given function in parallel and XOR their outputs. For example, $x \mapsto f(x)$ where

$$f(x_0||...||x_{\ell-1}) = \oplus_{i=0}^{\ell-1} E_k(\text{byte}(i)||0^{n-t-8}||x_i)$$

and where byte$(i)$ denotes the 8-bit representation of the integer $i$ combines $\ell$ different instances of $\mathsf{BcCounter}^t_v$. We consider that this function is asymmetrically $\big(\text{Code}, 2^p, \max(2^{p-v}, (2^{p-t}/\ell)^\ell)\big)$-hard.

Indeed, an attacker could store $2^p/\ell$ entries of each of the $\ell$ distinct tables, in which case they can evaluate the whole function if and only if all the table entries they need are among those they know. This happens with probability $(2^{p-t}/\ell)^\ell$. Alternatively, they could store the output of the whole function for about $2^p$ values of the complete input. In that case, they can evaluate the function if and only if the whole input is one that was precomputed, which happens with probability $2^{p-v}$. We assume that there is not better attack for a $2^p$-adversary than the ones we just described, hence the hardness we claimed.

It is also possible to use a white-box block cipher as an asymmetrically code-hard function as this complexity is precisely the one they are designed to achieve.

## 15.3 Modes of Operations for Building Hard Primitives

As said above, our strategy is to combine hard plugs with secure cryptographic primitives in such a way that the input of the plugs are randomized and that enough such calls are performed to ensure that at least one plug evaluation was hard with a high enough probability. The method we use is nicknamed *plug-then-randomize*. It is formalized in Section 15.3.1. Then, the block cipher and the sponge mode of operation based on it are introduced respectively in Sections 15.3.2 and 15.3.3.

### 15.3.1 Plug-Then-Randomize

**Definition 15.3.1** (Plugged Function)**.** *Let $P : \{0,1\}^t \rightarrow \{0,1\}^v$ be a plug and let $F : \{0,1\}^n \rightarrow \{0,1\}^n$ be a function, where $t + v \leq n$. The* plugged function *$(F \cdot P) : \{0,1\}^n \rightarrow \{0,1\}^n$ maps $x = x_t||x_v||x'$ with $|x_t| = t$, $|x_v| = v$ and $|x'| = m - t - v$ to $y$ defined by:*

$$(F \cdot P)(x_t \mid\mid x_v \mid\mid x') = y = F\left(x_t \mid\mid x_v \oplus P(x_t) \mid\mid x'\right).$$

*This computation is summarized in Figure 15.2.*

**Lemma 15.3.1** (Plugged Function Hardness)**.** *If $P : \{0,1\}^t \rightarrow \{0,1\}^v$ is a plug $(\rho, u, \epsilon(p))$-hard against $2^p$-adversaries and if $F : \{0,1\}^n \rightarrow \{0,1\}^n$ is a public random (permutation) oracle then the plugged function $(F \cdot P)$ is $(\rho, u, \epsilon(p))$-hard.*

*Proof.* First, the adversary could try and store $2^p$ outputs of $(F \cdot P)$. However, such an approximation would work only with probability $2^{p-n} < 2^{p-v} \leq \epsilon$, so that it is less successful than an approximation based on an approximation of the plug.

Without knowledge of the full input of $F$, it is impossible to predict its output because $F$ is a random (permutation) oracle. Therefore, we simply need to show that

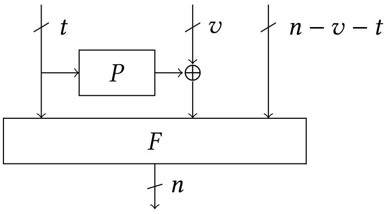


Figure 15.2: Evaluating the plugged function $(F \cdot P)$.

the function $\mathcal{F}_P$ mapping $(x,y,z)$ of $\{0,1\}^t \times \{0,1\}^v \times \{0,1\}^{n-t-v}$ to $(x, y \oplus P(x), z)$ is as hard as $P$ itself.

By contradiction, suppose that there is an adversary $\mathcal{A}$ capable of winning the approximation game for $\mathcal{F}_P$. That is, $\mathcal{A}$ can compute an approximation $\mathcal{F}_P'$ of $\mathcal{F}_P$ using less than $u$ units of the resource $\rho$ which works with probability strictly higher than $\epsilon(p)$. Then $\mathcal{A}$ can win the approximation game for $P$ itself as follows. When given $P$, $\mathcal{A}$ computes the approximation $\mathcal{F}_P'$ of the corresponding function $\mathcal{F}_P$. Then, when given a random input of $P$ of length $t$, $\mathcal{A}$ concatenates with random bitstrings $y$ and $z$ of length $v$ and $n-t-v$ respectively. The output of $P$ is then approximated as the $v$ center bits of $\mathcal{F}_P'(x||y||z) \oplus (x||y||z) = 0^t||P(x)||0^{n-t-v}$. Thus, $\mathcal{A}$ can violate the $(\rho, u, \epsilon(p))$-hardness of $P$.

We deduce that if $P$ is $(\rho, u, \epsilon(p))$-hard, then so is $\mathcal{F}_P$ and thus $(F \cdot P)$. □

Using this lemma, we can prove the following theorem which will play a key role in justifying the $\mathcal{R}$-hardness of our later constructions.

**Theorem 15.3.1** (Iterated Plugged Function Hardness)**.** *Let $F_i$, $i < r$ be a family of $r$ different random oracles (or random permutation oracles) mapping $n$ bits to $n$. Let $P : \{0,1\}^t \to \{0,1\}^v$ with $t + v \leq n$ be a plug $(\rho, u, \epsilon(p))$-hard against $2^p$-adversaries. Then the function $f : \{0,1\}^n \to \{0,1\}^n$ defined by*

$$f : x \mapsto \Big((F_{r-1} \cdot P) \circ \ldots \circ (F_0 \cdot P)\Big)(x)$$

*is $(\rho, u, \max(\epsilon(p)^r, 2^{p-n}))$-hard against $2^p$-adversaries.*

*Proof.* We denote $f_i$ the function defined by $f : x \mapsto \Big((F_{i-1} \cdot P) \circ \ldots \circ (F_0 \cdot P)\Big)(x)$, so that $f = f_r$. We proceed by induction on the number of rounds $i$, our induction hypothesis being that the theorem holds for $r \leq i$.

**Initialization.** If $i = 1$ i.e. for $f_1 = (F \cdot P)$, Lemma 15.3.1 tells us that this function is $(\rho, u, \epsilon)$-hard. As $\epsilon \geq 2^{p-v} > 2^{p-n}$, the induction holds for $i = 1$.

**Inductive Step.** Suppose that the theorem holds for $i$ rounds. The attack based on pre-querying $2^p$ outputs of $f_{i+1}$ and then approximating $f_{i+1}$ using the content of this table would still work. Thus, if $\epsilon^{i+1} \leq 2^{n-p}$ then this strategy is the optimal one. Suppose now that $\epsilon^{i+1} > 2^{n-p}$, which also implies that $\epsilon^i > 2^{n-p}$.

As $F_{i+1}$ is a random (permutation) oracle, the only way to evaluate the output of $f_{i+1}$ is to first evaluate $f_i$ and then to evaluate $(F_{i+1} \cdot P)$. The existence of another efficient computation method would violate the assumption that $F_{i+1}$ is a random oracle. The attack strategy consisting in precomputing several outputs of the random oracles is limited by the fact that we consider only $2^p$-adversaries.

Thus, the adversary needs first to evaluate $f_i$ and then $(F_{i+1} \cdot P)$. Let $f'_j$ be an approximation of the function $f_j$ computed by a $2^p$-adversary and let $g_j$ be an approximation of $(F_j \cdot F)$ computed by the same adversary. The probability of the successful evaluation of $f'_{i+1}$ is:

$$\begin{aligned}\Pr\left[f'_{i+1}(x) = f_{i+1}(x),\ x \xleftarrow{\$} \{0,1\}^n\right] & \\ = \ & \Pr\left[g_{i+1}(y) = (F_{i+1} \cdot P)(y) \mid y = f_i(x)\right] \\ & \times \Pr\left[f'_i(x) = f_i(x), x \xleftarrow{\$} \{0,1\}^n\right].\end{aligned}$$

Because of the induction hypothesis, we know that

$$\Pr\left[f'_i(x) = f_i(x), x \xleftarrow{\$} \{0,1\}^n\right] \le \epsilon^i$$

On the other hand, the first term is equal to

$$\begin{aligned}\Pr\left[g_{i+1}(y) = (F_{i+1} \cdot P)(y) \mid y = f_i(x)\right] & \\ = \ & \Pr\left[g_{i+1}(y) = (F_{i+1} \cdot P)(y), y \xleftarrow{\$} \{0,1\}^n\right]\end{aligned} \tag{15.1}$$

which, because of Lemma 15.3.1, is at most equal to $\epsilon$.

Equation (15.1) is true. Were it not the case, then $F_{i+1}$ would not be behaving like a random oracle. Indeed, $y = f_i(x)$ is the output of a sequence of random oracle calls sandwiched with simple bijections consisting in the plug calls that are independent from said oracle. Therefore, since $x$ is picked uniformly at random, $y$ must take any value with equal probability. Furthermore, the events $f_i(x) = y$ and $g_{i+1}(y) = (F_{i+1} \cdot P)(y)$ are independent: the latter depends only on the last random (permutation) oracle $F_{i+1}$ while the former depends on all other random (permutation) oracles. As a consequence, the probability that $f'_{i+1}(x) = f_i(x)$ for $x$ picked uniformly at random and for any approximation $f'_{i+1}$ obtained by a $2^p$-adversary is upper-bounded by $\epsilon^{i+1}$. □

### 15.3.2 Hard Block Cipher Mode (HBC)

Let $E_k$ be a block cipher operating on $n$-bit blocks using a key of length $\kappa \ge n$. Let $P$ be a plug $(\rho, u, \epsilon(p))$-hard against $2^p$-adversaries. The HBC mode of operation iterates these two elements to create an $n$-bit block cipher with a $\kappa$-bit secret key which is $(\rho, u, \max(\epsilon(p)^r, 2^{p-n}))$-hard against $2^p$-adversaries. This construction, when keyed by the $\kappa$-bit key $k$, is the permutation $\text{HBC}[E_k, P, r]$ which transforms an $n$-bit input $x$ as described in Algorithm 15.1. This process is also summarized in Figure 15.3. Below, we describe the hardness (Theorem 15.3.2) such a HBC instance achieves. We also reiterate that if an asymmetrically hard plug is used then the block cipher thus built is also asymmetrically hard.

Our proof is in the ideal cipher model, a rather heavy handed assumption. We leave as future work to prove the hardness of this mode of operation in simpler settings.

The role of the round counter XORed in the key is merely to make the block cipher calls independent of one another. If the block cipher had a tweak, these counter additions could be replaced by the use of the counter as a tweak with a fixed key. It is possible to use block ciphers which are not secure in the related-key setting and

**Algorithm 15.1** HBC$[E_k, P, r]$ encryption
*Inputs:* $n$-bit plaintext $x$; $\kappa$-bit key $k$
*Output:* $n$-bit ciphertext $y$

```
$y \leftarrow E_k(x)$
for all $i \in \{1, \ldots, r\}$ do
    $y_t \mid\mid y_{n-t} \leftarrow y$, where $|y_t| = t$ and $|y_{n-t}| = n - t$
    $y_{n-t} \leftarrow y_{n-t} \oplus P(y_t)$
    $y \leftarrow E_{k \oplus i}(y_t || y_{n-t})$
end for
return $y$
```

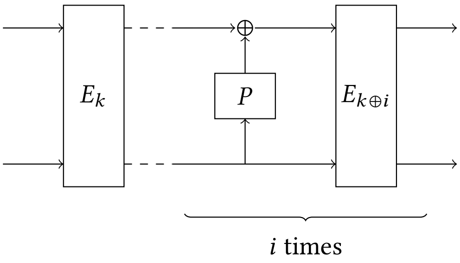


Figure 15.3: The HBC block cipher mode.

still retain the properties of HBC by replacing the keys $k \oplus i$ by the outputs of a key derivation function.

**Theorem 15.3.2** (Hardness of HBC). *If the block cipher $E_k$ used to build HBC$[E_k, P, r]$ is an ideal block cipher and if the plug $P$ is $(\rho, u, \epsilon(p))$-hard against $2^p$-adversaries, then the block cipher HBC$[E_k, P, r]$ is $(\rho, u, \max(\epsilon(p)^r, 2^{n-p}))$-hard against $2^p$-adversaries.*

*Proof.* As $E_k$ is an ideal cipher, $E_k$ and $E_{k\oplus i}$ cannot be distinguished from two independent random permutation oracles using less than $2^\kappa$ operations. As a consequence Theorem 15.3.1 immediately gives us the theorem. □

We used the HBC structure to build an asymmetrically time-hard block cipher, Skipper, which we describe in Section 15.4.1.

### 15.3.3 Hard Sponge Mode (HSp)

The sponge construction was introduced by Bertoni *et al.* as a possible method to build a hash function [BDPVA07]. They used it to design Keccak [BDPVA09] which later won the Sha-3 competition. It is a versatile structure which can be used to implement hash functions, stream ciphers, message authentication codes (mac), authenticated ciphers as described in [BDPV12], pseudo-random number generators (prng) and key derivation functions (kdf) as explained for example in [GT16]. In this section, we first provide a brief reminder on the sponge construction as this object is relatively recent compared to block ciphers and hash functions. Then, we show how plugs can be combined with secure sponge transformation to build $\mathcal{R}$-hard sponges, thus providing a $\mathcal{R}$-hard hash function, mac, etc.

#### 15.3.3.1 The Sponge Construction

A sponge construction uses an $n$-bit public permutation $g$ and is parametrized by its capacity $c$ and its rate $r$ which are such that $r + c = n$. This information is sufficient to build a hash function, as was described in Section 1.2.1.3 (p. 7). Let us briefly recall how it works. The two higher level operations provided by a sponge object parametrized by the function $g$, the rate $r$ and the capacity $c$ are listed below.

- **Absorption.** The $r$-bit block $m_i$ of the padded message $m$ is xored into the first $r$ bits of the internal state of the sponge and the function $g$ is applied.
- **Squeezing.** The first $r$ bits of the internal state are output and the function $g$ is applied on the internal state.

This process is summarized in Figure 15.4. The internal state of the sponge obviously needs to be initialized. It can be set to a fixed string to create a hash function. However, if the initial value is a secret key, we obtain a mac. Similarly, if the initial value is a secret key/initialization pair, we can generate a pseudo-random keystream by iterating the squeezing operation.

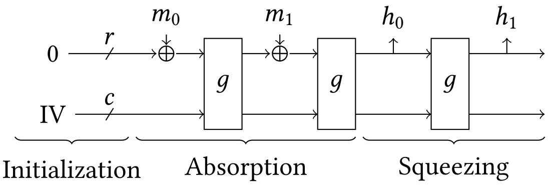


Figure 15.4: A sponge-based hash function.

As explained in [BDPV12], this structure can be modified to allow single-pass authenticated encryption. This is achieved by using the sponge object to generate a stream with the added modification that, between the generation of the $r$-bit keystream block and the application of $g$ to the internal state, the $r$-bit block of padded message is xored into the internal state, just like during the absorption phase. In this case, there is no distinction between the absorption and squeezing phase. Once the whole padded message has been absorbed and encrypted using the keystream, the sponge object is squeezed to obtain the tag.

Finally, a sponge object can also be used to build a kdf or a prng using a similar strategy in both cases, as proposed in [GT16]. The general principle is to absorb the output of the low-entropy source and follow the absorption of each block by many iterations of $x \mapsto 0^r || T_c(g(x))$ on the internal state, where $T_c(x)$ is equal to the last $c$ bits of $x$. Setting the first $r$ bits of the internal state to zero prevents the inversion of the update function.

Sponge constructions are known to be secure as long as the public update function $g$ has no *structural distinguishers* such as high probability differentials or linear approximation with a high bias.

The main advantages of the sponge structure are its simplicity and its versatility. It is simple because it only needs a public permutation and it is versatile because all symmetric primitives except block ciphers can be built from it with very little overhead. As we will show below, the fact that its internal state is larger than that of a usual block cipher also means that attacks based on precomputations are far weaker.

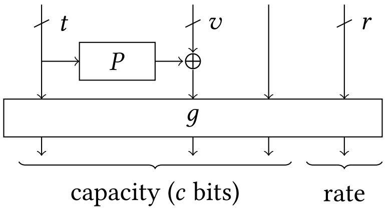


Figure 15.5: The hard sponge transformation $(g \cdot P)$.

### 15.3.4 The HSp Mode and its Hardness

Given that a sponge object is fully defined by its rate $r$, capacity $c$ and public update function $g$, we intuitively expect that building a $\mathcal{R}$-hard sponge object can be reduced to building a $\mathcal{R}$-hard update function. As stated in the theorem below, this intuition is correct provided that the family of functions $g_k : \{0,1\}^c \rightarrow \{0,1\}^c$ indexed by $k \in \{0,1\}^r$ and defined as the capacity bits of $g(x||k)$ is assumed to be a family of independent random oracles.

We call HSp the mode of operation described in this section. It is superficially similar to a round of the HBC block cipher mode.

An update function $g$ can be made $\mathcal{R}$-hard using the $\mathcal{R}$-hardness of a plug $P : \{0,1\}^t \rightarrow \{0,1\}^v$ to obtain a new update function $(g \cdot P)$ as described in Algorithm 15.2.

**Algorithm 15.2** $(g \cdot P)$ sponge transformation
*Inputs:* $n$-bit block $x$;
*Output:* $n$-bit block $y$

```
$x_t \mid\mid x_v \mid\mid x' \leftarrow x$, where $|x_t| = t, |x_v| = v, |x'| = n - t - v$
$x_v \leftarrow x_v \oplus P(x_t)$
$y \leftarrow g(x_t \mid\mid x_v \mid\mid x')$
return $y$
```

This process is summarized in Figure 15.5. In order to prevent the adversary from reaching either the input or the output of $P$, which could make some attacks possible, we impose that $t + v \leq c$ so that the whole plug input and output are located in the capacity.

**Theorem 15.3.3** (HSp absorption hardness). *Consider a sponge defined by the $n$-bit transformation $(g \cdot P)$, a rate $r$ and a capacity $c$ so that $r + c = n$ and $r > c$. Let $(g \cdot P)$ be defined as in Algorithm 15.2, where $P : \{0,1\}^t \rightarrow \{0,1\}^v$ is a plug $(\rho, u, \epsilon(p))$-hard against $2^p$-adversaries.*

*Let* Absorb $: \{0,1\}^{\ell \times r} \rightarrow \{0,1\}^c$ *be the function mapping an un-padded message $m$ of $\ell$ $r$-bit blocks to the capacity bits of the internal state of the sponge after it absorbed $m$.*

*Furthermore, suppose that the $n$-bit transformation $g$ is such that the family of functions $g_k : \{0,1\}^c \rightarrow \{0,1\}^c$ indexed by $k \in \{0,1\}^r$ and defined as $g_k(x) = T_c\left((g(x||k)\right)$ can be modeled as a family of random oracles.*

*Then* Absorb *is* $\left(\rho, u, \max(\epsilon(p)^{\ell-1}, 2^{p-c})\right)$*-hard against $2^p$-adversaries.*

This theorem deals with un-padded messages. The padding of such a message

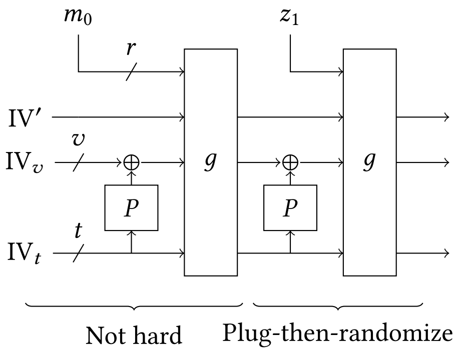


Figure 15.6: An alternative representation of the absorption procedure.

imposes the creation of a new block with a particular shape which cannot be considered to be random.

*Proof.* Let $g_k : \{0,1\}^c \rightarrow \{0,1\}^c$ be as defined in the theorem. Let the message $m$ be picked uniformly at random.

The first call to $(g \cdot P)$ is not $(\rho, u, \epsilon(p))$-hard. Indeed, the content of the message has not affected the content of the capacity yet. However, the capacity bits of the internal state after this first call to $(g \cdot P)$ are uniformly distributed as they are the image of a constant by the function indexed by $m_0$ from a family of $2^r$ different random oracles.

Let $m'_i = m_i \oplus z_i$, where $m_i$ is the message block with index $i > 1$ and where $z_i$ is the first $r$ bits of the content of the sponge after the absorption of $m_0, ..., m_{i-1}$. That is, $z_i$ is the content of the rate just before the call to $(g \cdot P)$ following the addition of the message block $m_i$. We can therefore represent the absorption function as described in Figure 15.6.

Since the message blocks $m_i$ have been picked uniformly at random, so are the values $z_i$. We can therefore apply Theorem 15.3.1, where the independent random oracles are $g_{z_i}$, the plug is $P$, the random message is $(g_{m_0} \cdot P)(0||IV)$, the block size is $c$ and the number of rounds is $\ell - 1$. □

As $c$ is typically much larger than a usual block cipher size of 128 bits, the probability of the success of a $2^p$ adversary can be made much smaller when a sponge is built rather than a block cipher.

Note that if a sponge is used to provide e.g. authenticated encryption, the same bound should be used as the message is absorbed into the state in the same fashion in this case.

The following claim describes the hardness of the squeezing operation.

**Claim 1** (HSp squeezing hardness)**.** *Consider a sponge defined by the $n$-bit transformation $(g \cdot P)$, a rate $r$ and a capacity $c$ so that $r + c = n$ and $r > c$. Let $(g \cdot P)$ be defined as in Algorithm 15.2, where $P : \{0,1\}^t \rightarrow \{0,1\}^v$ is a plug $(\rho, u, \epsilon(p))$-hard against $2^p$-adversaries.*

*Let* Squeeze$^\ell : \{0,1\}^n \rightarrow \{0,1\}^{\ell \times r}$ *be the function mapping an internal state of $n$ bits to a stream of $\ell$ $r$-bit blocks obtained by iterating $\ell$ times the* Squeeze *operation.*

*Then* Squeeze$^\ell$ *is* $\left(\rho, u, \max(\epsilon(p)^\ell, 2^{p-(c+r)})\right)$*-hard against $2^p$-adversaries.*

We cannot prove this hardness using Theorem 15.3.1 because the transformations called in each round are all identical. In particular, they cannot be independent. This situation can however be interpreted as a variant of the one in the proof of Theorem 1 where $z_i$ is not formally picked uniformly at random as there is no message absorption but can be interpreted as such because it is the output of the sponge function.

The claimed probability of success bound comes from the hardness of approximating all $\ell$ calls to the plug composed with the sponge transformation ($\epsilon(p)^{\ell}$) and the hardness of using a precomputation of the image of $2^p$ possible internal states ($2^{p-(c+r)}$)).

If the sponge is used to provide a simple stream cipher, this claimed bound should be used. Since there is no message absorption in this case, Theorem 1 cannot be used.

## 15.4 Practical Instances: Skipper and Whale

We illustrate the versatility and simplicity of the modes of operation described in the previous section by presenting an instance of each. The first is an asymmetrically time-hard block cipher called Skipper and the second is a code-hard sponge called Whale.

### 15.4.1 The Skipper Block Cipher

One possible application for *egalitarian computing* which is mentioned but not explored in the original paper [BK16a] is *obfuscation*. The idea is to modify a program in such a way that a memory-hard function must be computed in parallel to the execution of the program. Using this very general approach, any program or function could be made memory-hard, not just cryptographic ones. However, a shortcoming in this case is the fact that the compiler returning the obfuscated code must also pay the full price of running this parallel memory-hard function.

Solving this issue requires a primitive with asymmetric hardness. While we do not have an asymmetrically memory-hard plug, we do have an asymmetrically time-hard and a code-hard one. Using the HBC mode, we combine the Aes and the rsa-lock plug to create a block cipher with asymmetric time-hardness, Skipper. It could be used to create an efficient obfuscator. The obfuscator would use the fast implementation of the plug to create an obfuscated program which forces common users to evaluate its slow version to run the program. That way, the computational hardness is only paid by the users of the program and not by the compiler. While this cost might be offset through the use of dedicated hardware for the computation of rsa-lock, we note that this function cannot be parallelized.

Our proposal Skipper is HBC[$\text{AES}-128, \text{RSAlock}_{\eta}^{n_p}, 2$], that is, a 128-bit block cipher using a 128-bit secret key $k$, an $\text{RSAlock}_{\eta}^{n_p}$ instance truncated to 40 bits as a plug and 3 calls to Aes-128 sandwiching 2 calls to the plug. The plug operates modulo $N \geq 2^{n_p}$. The Skipper encryption procedure is described in Algorithm 15.3.

The Rsa-based plug we use is asymmetrically $\big(\text{Time}, \eta, \max(2^{p-88}, 2^{-40})\big)$-hard. As said before in Section 15.2.3, we assume that no adversary can evaluate $x^{2^t} \mod N$ without performing $\eta$ squarings in the modular ring. However, a $2^p$-adversary can either guess all 40 bits of the output, which succeeds with probability $2^{-40}$, or store $2^p$ out of the $2^{88}$ possible outputs, in which case a successful evaluation is possible with probability $2^{p-88}$.

**Algorithm 15.3** Skipper encryption
*Inputs:* $n$-bit plaintext $x$; $k$-bit key $k$; Rsa modulus $N$
*Output:* $n$-bit ciphertext $y$

```
$y \leftarrow \mathrm{AES}_k(x)$
for all $i \in \{1,2\}$ do
    $y_1 \mid\mid y_2 \leftarrow y$, where $|y_1| = 88$ and $|y_2| = 40$
    $y_2 \leftarrow y_2 \oplus T_{40}(y_1^{2^\eta} \mod N)$
    $y \leftarrow \mathrm{AES}_{k \oplus i}(y_1 || y_2)$
end for
return $y$
```

Merely guessing is the best strategy unless the adversary has access to at least $40 \times 2^{88-40} \approx 2^{53.3}$ bits of storage, i.e. more than a thousand terabytes. Furthermore, the cost of such a pre-computation could only be amortized if more than $2^{48}/2 = 2^{47}$ blocks are encrypted using the same plug, i.e. $2^{54}$ bits (more than a thousand Tb). We therefore consider $2^{48}$-adversaries, that is, adversaries capable of pre-computing $2^{48}$ values of $\mathrm{RSAlock}_\eta^{n_p}(x)$. Such an adversary is already quite powerful as it has significant computing power and storage in addition to knowing the secret key $k$. Providing maximum security against more powerful adversaries would probably be over-engineering. Thus, in our setting, the plug is asymmetrically $(\mathrm{Time}, \eta, 2^{-40})$-hard.

**Claim 2** (Properties of Skipper)**.** *The block cipher Skipper is asymmetrically* $(Time, \eta, 2^{-80})$*-hard and cannot be distinguished from a pseudo-random permutation using less than* $2^{128}$ *operations.*

Skipper is $\mathrm{HBC}[\mathrm{AES}-128, \mathrm{RSAlock}_\eta^{n_p}, 2]$ and its plug is asymmetrically $(\mathrm{Time}, \eta, 2^{-40})$-hard. Thus, by applying Theorem 15.3.2 we obtain that Skipper is asymmetrically $\left(\mathrm{Time}, \eta, \max\left(2^{48-128}, (2^{-40})^2\right)\right)$-hard.

As there is to the best of our knowledge no related-key attack against full-round AES-128 in the case where the related keys are linked by a simple XOR, we claim that Skipper cannot be distinguished from a random permutation using much less than $2^{128}$ operations. Should such distinguishers be found, an alternative key schedule such as the one from [Nik11] could be used.

We have implemented this block cipher using `openssl` (for AES) and the `GMP` library (for RSA). Specifically, modular exponentiation is performed with the function `mpz_powm_sec` to minimize the threat of side-channel attacks, although this function is about 20% slower than the basic `mpz_powm`.

To benchmark its speed, we have computed the average time taken to encrypt 20 blocks of data for all $n_p \in \{768, 1024, 1536, 2048, 4096\}$ and all $\eta = \mu \times n_p$ for $\mu \in \{2, 10, 100, 1000\}$, each time being the average of 10 experiments performed with a different RSA modulus. These are just examples, in fact arbitrary slowdowns for the non-privileged user are possible. We express the results as throughputs in blocks per seconds. The results are given in Table 15.3; e.g. a common user using Skipper with $n_p = 768$ and $\mu = 2$ encrypts on average 750.1 blocks of 128 bits per second on our benchmark machine.

In order to prevent attacks from $2^p$-adversaries who cannot perform more than $2^{100}$ operations, a modulus of 2048 bits is necessary according to Table 2 of [BBB+07].

Otherwise, such adversaries could simply factorize the RSA modulus. However, as we can see in Table 15.3, the performance penalty when increasing $n_p$ is significant.

| $n_p$ | $\mu$ | Tput. common | Tput. privileged | ratio |
|---|---|---|---|---|
| 768 | 2 | 750.1 | 4234.4 | 5.6 |
| | 20 | 76.5 | 4182.7 | 54.7 |
| | 500 | 3.0 | 4210.3 | 1401.8 |
| 1024 | 2 | 357.6 | 2012.0 | 5.6 |
| | 20 | 35.8 | 2062.1 | 57.6 |
| | 500 | 1.4 | 2106.8 | 1463.3 |
| 1536 | 2 | 119.4 | 739.7 | 6.2 |
| | 20 | 12.3 | 752.0 | 60.9 |
| | 500 | 0.5 | 754.9 | 1525.5 |
| 2048 | 2 | 52.4 | 346.1 | 6.6 |
| | 20 | 5.5 | 358.1 | 64.8 |
| | 500 | 0.22 | 359.4 | 1605.2 |
| 4096 | 2 | 7.6 | 54.5 | 7.1 |
| | 20 | 0.77 | 55.4 | 71.8 |
| | 500 | 0.03 | 55.4 | 1778.1 |

Table 15.3: Benchmark of the Skipper block cipher, where $n_p$ is the RSA modulus, $\mu \times n_p$ is the number of squarings performed by the common user, and the throughputs are given in block/second.

The tests were done on a regular desktop with an i7-3770 CPU clocked at 3.4 Ghz and 8 Gb of RAM. The code was written in C++.

The ratio of the efficiency of the encryption depending on whether the secret is known or not is not equal to $\mu$. This is because the common user performs exactly $\mu \times n_p$ squarings while the privileged one uses a smaller exponent. Further, privileged user uses the Chinese Remainder Theorem to implement an even faster exponentiation. Still, multiplying $\mu$ by 10 does divide the throughput by 10 for the common user.

### 15.4.2 The Whale Hash Function

Preventing the leakage of encryption keys is a necessity in order for a system to be secure. A possible method for preventing this was informally proposed by Shamir in a talk at Rsa'2013 and then formalized by Bellare *et al.* in their crypto'16 paper. As the throughput of the exfiltration method used by the attacker is limited, using a huge key would make their task all the harder. To use our terminology, an encryption algorithm with significant code-hardness would effectively be bound to the physical device storing it: since the code cannot be compressed, an attacker would have to duplicate the whole implementation to be able to decrypt the communications. Even a partial leakage would be of little use.

The proposal of Bellare *et al.*, XKEY2, is effectively a code-hard key derivation algorithm which takes as input a random initialization vector and outputs a secret key. Since it is code-hard, an attacker cannot evaluate this function without full knowledge of the source code of the function and cannot extract a smaller (and thus easier to leak) implementation.

We propose the code-hard hash function Whale as an alternative to XKEY2. It can indeed be used to derive a key by hashing a nonce, a process which cannot be approximated by an attacker unless they duplicate the entirety of the implementation of Whale. Whale is a simple sponge-based hash function which uses the xor of $\lceil 128/t \rceil$ instances of $\mathsf{BigLUT}_t^{128}$ as a plug. Different choices of $t$ lead to different levels of code-hardness. It is only parametrized by the input length of the tables $t$.

It is based on Sha-3-256: it uses the Keccak $- f[1600]$ permutation, the same padding scheme, the same rate $r = 1088$, the same capacity $c = 512$ and the same digest size of 256 bits. There are only two differences:

- the permutation Keccak $- f[1600]$ is augmented with the code-hard plug consisting of the xor of $\ell = \lceil 128/t \rceil$ distinct instances of $\mathsf{BigLUT}_t^{128}$, and
- $t$ blank calls to the transformation are performed between absorption and squeezing.

These parameters were chosen so as to prevent an adversary with access to at most half of the implementation of Whale to compute the digest of a message with probability higher than $2^{-128}$.

**Claim 3** (Code-hardness of Whale)**.** *The Whale hash function using tables with $t$-bit inputs is $(Code, 2^{t+13}/t, 2^{-128})$-hard against an adversary trying to use only half of the code-space used to implement Whale.*

Whale uses $\lceil 128/t \rceil$ tables of $2^t$ 128-bit entries. Thus, about $2^t \times 128 \times \lceil 128/t \rceil \approx 2^{t+14}/t$ bits are needed to store the implementation of its plug. An adversary trying to compress it and divide its size by 2 therefore has access to $2^{t+13}/t$ bits. Note however that, since the entries in each instance of $\mathsf{BigLUT}_t^{128}$ have been picked uniformly at random, it is impossible to actually compress them. The best an attacker can do is therefore to store as many entries as they can.

When hashing a message, at least $t$ calls to the plug are performed during the blank calls to the transformation between the absorption and the squeezing. Therefore, the adversary needs to successfully compute $t \times \lceil 128/t \rceil \geq 128$ entries of the big tables. If they only have half of them stored, then they succeed in computing the digest of a message with probability at most $2^{-128}$.

# Conclusion

# Final Words

## On Part I

The first part of this thesis discussed lightweight cryptography. I argued that this field had become too wide and that it should be split into two different areas: ultra-lightweight cryptography, dedicated to the tiniest of devices, and IoT cryptography, aimed at low-power but network-enabled micro-controllers. I also presented several attacks targeting such algorithms. They are based on a different observation for each primitive:

1. the update function of Gluon-64 looses too much information,
2. the linear layer of Prince causes the existence of families of differential pairs following the same trail, and
3. the branch permutation used by Twine has a surprisingly structured diffusion.

Apart from Gluon-64, these observations are not detrimental to the security of the full-round primitive. Nevertheless, they improve our understanding of the inner workings of these algorithms.

The last two chapters of this part dealt with our own block cipher, Sparx, which is one of the most efficient on low-power micro-controllers. More importantly, it is the first ARX-based block cipher provably secure against simple differential and linear cryptanalysis. I hope that the Long Trail Strategy it introduces will be the basis of other designs.

## On Part II

S-Boxes are some of the main components of many symmetric primitives. Understanding the construction of the S-Box used by a given algorithm is a crucial step in assessing the security offered by said algorithm. In Part II, I presented a survey of all the S-Boxes used by public algorithms. If an S-Box re-uses any of the structures currently known, it is possible to recover it. The corresponding reverse-engineering techniques were presented.

They were successfully applied to the S-Boxes of Skipjack, Kuznyechik and Cmea. However, the results obtained raised more questions than answers: what is the motivation behind the structures and statistical artifacts hidden in these?

They also lead to new results on Boolean functions. In particular, a structure was identified in the only known solution of the APN problem. Unfortunately, it was also shown that a generalization of this structure to bigger fields cannot yield an APN permutation.

## On Part III

It is possible to achieve a high time-, memory- or code-hardness using the same high level structure. This insight is at the heart of this part. It also implies a simple generalization of the concept of white-box cryptography. It can also be used to build block ciphers and sponge permutation with any of these hardnesses. This was illustrated by the design of the code-hard hash function WHALE and of the asymmetrically time-hard SKIPPER.

## Conclusion

While working on S-Box reverse-engineering and on purposefully hard cryptography, I identified several open problems. They are recalled in the next page. As the final word of this thesis, I will quote the conclusion of one of the first academic papers on symmetric cryptology which, despite being 41 years old [HMS+76], is still as relevant today as it was back then.

> *[...] it is poor security practice to trust a system whose design and certification will not be described* — Hellman *et al.*, 1976

Belval, May 16, 2017

# Open Problems

- *Open Problem 6.3.1 (p. 115).* Is it possible to design a structure similar to Lax in such a way that the maximum linear correlation after several rounds can be upper bounded?

- *Open Problem 8.5.1 (p. 158).* What is the exact structure used by the designers of the Des and the gost variants to build their S-Boxes?

- *Open Problem 9.1.1 (p. 167).* Is Conjecture 9.1.1 (p. 167) true?

- *Open Problem 12.2.1 (p. 229).* Is there an algorithm which, given a list of elements $L$ of $(\mathbb{F}_2^n)^2$, can efficiently find the largest vector spaces $\mathcal{U} \subset \mathbb{F}_2^n$ and $\mathcal{V} \subset \mathbb{F}_2^n$ such that $(u, v)$ is in $L$ for all $u \in \mathcal{U}$ and $v \in \mathcal{V}$?

- *Open Problem 12.2.2 (p. 232).* What is the cause of the checkered pattern occurring in both the ddt and the lat of a 3-round Feistel network (after composition with branch swaps)?

- *Open Problem 12.3.1 (p. 242).* Is there a decomposition of the mini-block ciphers $T$ and $U'$ used to build the S-Box of Cmea?

- *Open Problem 14.0.1 (p. 267).* Is it possible to find an apn permutation of $\mathbb{F}_{2^n}$ for $n$ even and $n \geq 8$?

- *Open Problem 15.1.1 (p. 308).* Is it possible to design an asymmetrically memory-hard (but code-efficient) function with superlinear penalties?

# Bibliography

[AA04] Sergey Agievich and Andrey Afonenko. Exponential s-boxes. Cryptology ePrint Archive, Report 2004/024, 2004. http://eprint.iacr.org/2004/024.

[AA05] S.V. Agievich and A.A. Afonenko. О свойствах экспоненциальных подстановок [On properties of the exponential S-Boxes]. In Вести НАН Беларуси *[News of National Academy of Sciences of Belarus]*, volume 1, pages 106–112. National Academy of Sciences of Belarus, 2005. Available at http://elib.bsu.by/handle/123456789/24138.

[AB05] François Arnault and Thierry P. Berger. F-FCSR: Design of a new class of stream ciphers. In Henri Gilbert and Helena Handschuh, editors, *Fast Software Encryption – FSE 2005*, volume 3557 of *Lecture Notes in Computer Science*, pages 83–97. Springer, Heidelberg, February 2005.

[AB12] Jean-Philippe Aumasson and Daniel J. Bernstein. SipHash: A fast short-input PRF. In Steven D. Galbraith and Mridul Nandi, editors, *Progress in Cryptology - INDOCRYPT 2012: 13th International Conference in Cryptology in India*, volume 7668 of *Lecture Notes in Computer Science*, pages 489–508. Springer, Heidelberg, December 2012.

[AB16a] Joël Alwen and Jeremiah Blocki. Efficiently computing data-independent memory-hard functions. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology – CRYPTO 2016, Part II*, volume 9815 of *Lecture Notes in Computer Science*, pages 241–271. Springer, Heidelberg, August 2016.

[AB16b] Joël Alwen and Jeremiah Blocki. Towards practical attacks on Argon2i and balloon hashing. Cryptology ePrint Archive, Report 2016/759, 2016. http://eprint.iacr.org/2016/759.

[ABFG14] Giuseppe Ateniese, Ilario Bonacina, Antonio Faonio, and Nicola Galesi. Proofs of space: When space is of the essence. In Michel Abdalla and Roberto De Prisco, editors, *SCN 14: 9th International Conference on Security in Communication Networks*, volume 8642 of *Lecture Notes in Computer Science*, pages 538–557. Springer, Heidelberg, September 2014.

[ABL$^+$09] François Arnault, Thierry P. Berger, Cédric Lauradoux, Marine Minier, and Benjamin Pousse. A new approach for FCSRs. In Michael J. Jacobson Jr., Vincent Rijmen, and Reihaneh Safavi-Naini, editors, *SAC 2009: 16th Annual International Workshop on Selected Areas in Cryptography*, volume 5867 of *Lecture Notes in Computer Science*, pages 433–448. Springer, Heidelberg, August 2009.

[ABM$^+$12] Wim Aerts, Eli Biham, Dieter De Moitie, Elke De Mulder, Orr Dunkelman, Sebastiaan Indesteege, Nathan Keller, Bart Preneel, Guy A. E. Vandenbosch, and Ingrid Verbauwhede. A practical attack on KeeLoq. *Journal of Cryptology*, 25(1):136–157, January 2012.

[ABP$^+$13] Nadhem J. AlFardan, Daniel J. Bernstein, Kenneth G. Paterson, Bertram Poettering, and Jacob CN Schuldt. On the security of RC4 in TLS. In *Proceedings of the 22nd USENIX Security Symposium*, volume 2013, Washington DC, USA, 2013. USENIX.

[ACP$^+$16] Joël Alwen, Binyi Chen, Krzysztof Pietrzak, Leonid Reyzin, and Stefano Tessaro. Scrypt is maximally memory-hard. Cryptology ePrint Archive, Report 2016/989, 2016. http://eprint.iacr.org/2016/989.

[Ada97] C. Adams. The CAST-128 Encryption Algorithm. RFC 2144 (Informational), May 1997.

[ADK$^+$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 Juan A. Garay and Rosario Gennaro, editors, *Advances in Cryptology – CRYPTO 2014, Part I*, volume 8616 of *Lecture Notes in Computer Science*, pages 57–76. Springer, Heidelberg, August 2014.

[AHMN10] Jean-Philippe Aumasson, Luca Henzen, Willi Meier, and María Naya-Plasencia. Quark: A lightweight hash. In Stefan Mangard and François-Xavier Standaert, editors, *Cryptographic Hardware and Embedded Systems – CHES 2010*, volume 6225 of *Lecture Notes in Computer Science*, pages 1–15. Springer, Heidelberg, August 2010.

[AHMP08] Jean-Philippe Aumasson, Luca Henzen, Willi Meier, and Raphael C-W Phan. Sha-3 proposal blake. Submission to NIST, see also https://www.131002.net/blake/, 2008.

[AIK$^+$01] Kazumaro Aoki, Tetsuya Ichikawa, Masayuki Kanda, Mitsuru Matsui, Shiho Moriai, Junko Nakajima, and Toshio Tokita. Camellia: A 128-bit block cipher suitable for multiple platforms - Design and analysis. In Douglas R. Stinson and Stafford E. Tavares, editors, *SAC 2000: 7th Annual International Workshop on Selected Areas in Cryptography*, volume 2012 of *Lecture Notes in Computer Science*, pages 39–56. Springer, Heidelberg, August 2001.

[AJN16] Jean-Philippe Aumasson, Philipp Jovanovic, and Samuel Neves. NORX v3.0. Candidate for the CAESAR Competition. See also https://norx.io, 2016.

[ALL12] Farzaneh Abed, Eik List, and Stefan Lucks. On the security of the core of PRINCE against biclique and differential cryptanalysis. Cryptology ePrint Archive, Report 2012/712, 2012. http://eprint.iacr.org/2012/712.

[AM15] Frederik Armknecht and Vasily Mikhalev. On lightweight stream ciphers with shorter internal states. In Gregor Leander, editor, *Fast Software Encryption – FSE 2015*, volume 9054 of *Lecture Notes in Computer Science*, pages 451–470. Springer, Heidelberg, March 2015.

[And94] Ross Anderson. A5 (Was: HACKING DIGITAL PHONES). uk.telecom (Usenet), https://groups.google.com/forum/?msg/uk.telecom/TkdCaytoeU4/Mroy719hdroJ#!msg/uk.telecom/TkdCaytoeU4/Mroy719hdroJ, June 1994.

[Ano95] Anonymous. This looks like it might be interesting. sci.crypt (Usenet), https://groups.google.com/forum/#!msg/sci.crypt/vLtuBDoqPfc/jm6MshFbomgJ, August 1995.

[ANWOW13] Jean-Philippe Aumasson, Samuel Neves, Zooko Wilcox-O'Hearn, and Christian Winnerlein. BLAKE2: simpler, smaller, fast as MD5. Available online: https://blake2.net/blake2.pdf, 2013.

[AS15] Joël Alwen and Vladimir Serbinenko. High parallel complexity graphs and memory-hard functions. In Rocco A. Servedio and Ronitt Rubinfeld, editors, *47th Annual ACM Symposium on Theory of Computing*, pages 595–603. ACM Press, June 2015.

[Ava17] Roberto Avanzi. The QARMA block cipher family. almost MDS matrices over rings with zero divisors, nearly symmetric Even-Mansour constructions with non-involutory central rounds, and search heuristics for low-latency S-Boxes. *IACR Transactions on Symmetric Cryptology*, 2017(1):4–44, 2017.

[AY15a] Riham AlTawy and Amr M. Youssef. A meet in the middle attack on reduced round Kuznyechik. Cryptology ePrint Archive, Report 2015/096, 2015. http://eprint.iacr.org/2015/096.

[AY15b] Riham AlTawy and Amr M Youssef. Watch your constants: Malicious streebog. *IET Information Security*, 9(6):328–333, 2015.

[BAK98] Eli Biham, Ross J. Anderson, and Lars R. Knudsen. Serpent: A new block cipher proposal. In Serge Vaudenay, editor, *Fast Software Encryption – FSE'98*, volume 1372 of *Lecture Notes in Computer Science*, pages 222–238. Springer, Heidelberg, March 1998.

[BBB+07] Elaine Barker, William Barker, William Burr, William Polk, and Miles Smid. NIST special publication 800-57. *NIST Special Publication*, 800(57):1–142, 2007.

[BBD+99] Eli Biham, Alex Biryukov, Orr Dunkelman, Eran Richardson, and Adi Shamir. Initial observations on Skipjack: Cryptanalysis of Skipjack-3XOR (invited talk). In Stafford E. Tavares and Henk Meijer, editors, *SAC 1998: 5th Annual International Workshop on Selected Areas* *in Cryptography*, volume 1556 of *Lecture Notes in Computer Science*, pages 362–376. Springer, Heidelberg, August 1999.

[BBI$^+$15] Subhadeep Banik, Andrey Bogdanov, Takanori Isobe, Kyoji Shibutani, Harunaga Hiwatari, Toru Akishita, and Francesco Regazzoni. Midori: A block cipher for low energy. In Tetsu Iwata and Jung Hee Cheon, editors, *Advances in Cryptology – ASIACRYPT 2015, Part II*, volume 9453 of *Lecture Notes in Computer Science*, pages 411–436. Springer, Heidelberg, November / December 2015.

[BBK03] Elad Barkan, Eli Biham, and Nathan Keller. Instant ciphertext-only cryptanalysis of GSM encrypted communication. In Dan Boneh, editor, *Advances in Cryptology – CRYPTO 2003*, volume 2729 of *Lecture Notes in Computer Science*, pages 600–616. Springer, Heidelberg, August 2003.

[BBK08] Elad Barkan, Eli Biham, and Nathan Keller. Instant ciphertext-only cryptanalysis of GSM encrypted communication. *Journal of Cryptology*, 21(3):392–429, July 2008.

[BBK$^+$13] Begül Bilgin, Andrey Bogdanov, Miroslav Knežević, Florian Mendel, and Qingju Wang. Fides: Lightweight authenticated cipher with side-channel resistance for constrained hardware. In Guido Bertoni and Jean-Sébastien Coron, editors, *Cryptographic Hardware and Embedded Systems – CHES 2013*, volume 8086 of *Lecture Notes in Computer Science*, pages 142–158. Springer, Heidelberg, August 2013.

[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 Palash Sarkar and Tetsu Iwata, editors, *Advances in Cryptology – ASIACRYPT 2014, Part I*, volume 8873 of *Lecture Notes in Computer Science*, pages 63–84. Springer, Heidelberg, December 2014.

[BBR16] Subhadeep Banik, Andrey Bogdanov, and Francesco Regazzoni. Atomic-AES: A compact implementation of the AES encryption/decryption core. In Orr Dunkelman and Somitra Kumar Sanadhya, editors, *Progress in Cryptology – INDOCRYPT 2016*, volume 10095 of *Lecture Notes in Computer Science*, pages 173–190, Cham, 2016. Springer International Publishing.

[BBS05] Eli Biham, Alex Biryukov, and Adi Shamir. Cryptanalysis of Skipjack reduced to 31 rounds using impossible differentials. *Journal of Cryptology*, 18(4):291–311, September 2005.

[BC11] Christina Boura and Anne Canteaut. Zero-sum distinguishers for iterated permutations and application to Keccak-f and Hamsi-256. In Alex Biryukov, Guang Gong, and Douglas R. Stinson, editors, *SAC 2010: 17th Annual International Workshop on Selected Areas in Cryptography*, volume 6544 of *Lecture Notes in Computer Science*, pages 1–17. Springer, Heidelberg, August 2011.

[BC16] Christina Boura and Anne Canteaut. Another view of the division property. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology – CRYPTO 2016, Part I*, volume 9814 of *Lecture Notes in Computer Science*, pages 654–682. Springer, Heidelberg, August 2016.

[BCC10] Céline Blondeau, Anne Canteaut, and Pascale Charpin. Differential properties of power functions. *International Journal of Information and Coding Theory*, 1(2):149–170, 2010.

[BCC11] Céline Blondeau, Anne Canteaut, and Pascale Charpin. Differential properties of $x \mapsto x^{2^t-1}$. *IEEE Transactions on Information Theory*, 57(12):8127–8137, 2011.

[BCD11] Christina Boura, Anne Canteaut, and Christophe De Cannière. Higher-order differential properties of Keccak and Luffa. In Antoine Joux, editor, *Fast Software Encryption – FSE 2011*, volume 6733 of *Lecture Notes in Computer Science*, pages 252–269. Springer, Heidelberg, February 2011.

[BCG$^+$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 Xiaoyun Wang and Kazue Sako, editors, *Advances in Cryptology – ASIACRYPT 2012*, volume 7658 of *Lecture Notes in Computer Science*, pages 208–225. Springer, Heidelberg, December 2012.

[BCS16] Dan Boneh, Henry Corrigan-Gibbs, and Stuart E. Schechter. Balloon hashing: A memory-hard function providing provable protection against sequential attacks. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *Advances in Cryptology – ASIACRYPT 2016, Part I*, volume 10031 of *Lecture Notes in Computer Science*, pages 220–248. Springer, Heidelberg, December 2016.

[BD04] M. Becker and A. Desoky. A study of the DVD content scrambling system (CSS) algorithm. In *Proceedings of the Fourth IEEE International Symposium on Signal Processing and Information Technology, 2004.*, pages 353–356, Dec 2004.

[BD07] Eli Biham and Orr Dunkelman. A framework for iterative hash functions - HAIFA. Cryptology ePrint Archive, Report 2007/278, 2007. http://eprint.iacr.org/2007/278.

[BD08] Steve Babbage and Matthew Dodd. The MICKEY stream ciphers. In Matthew Robshaw and Olivier Billet, editors, *New Stream Cipher Designs*, volume 4986 of *Lecture Notes in Computer Science*, pages 191–209. Springer Berlin Heidelberg, 2008.

[BDBP03] Alex Biryukov, Christophe De Canniére, An Braeken, and Bart Preneel. A toolbox for cryptanalysis: Linear and affine equivalence algorithms. In Eli Biham, editor, *Advances in Cryptology – EUROCRYPT 2003*, volume 2656 of *Lecture Notes in Computer Science*, pages 33–50. Springer, Heidelberg, May 2003.

[BDG16] Alex Biryukov, Daniel Dinu, and Johann Großschädl. Correlation power analysis of lightweight block ciphers: From theory to practice. In *International Conference on Applied Cryptography and Network Security – ACNS 2016*, volume 9696 of *Lecture Notes in Computer Science*, pages 537–557. Springer, 2016.

[BDK$^+$93] Ernest F. Brickell, Dorothy E. Denning, Stephen T. Kent, David P. Maher, and Walter Tuchman. SKIPJACK review: Interim report. This note is available at http://faculty.nps.edu/dedennin/publications/SkipjackReview.txt., 1993.

[BDK16] Alex Biryukov, Daniel Dinu, and Dmitry Khovratovich. Argon2: new generation of memory-hard functions for password hashing and other applications. In *2016 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 292–302. IEEE, 2016.

[BDM$^+$12] Thierry P. Berger, Joffrey D'Hayer, Kevin Marquet, Marine Minier, and Gaël Thomas. The GLUON family: A lightweight hash function family based on FCSRs. In Aikaterini Mitrokotsa and Serge Vaudenay, editors, *AFRICACRYPT 12: 5th International Conference on Cryptology in Africa*, volume 7374 of *Lecture Notes in Computer Science*, pages 306–323. Springer, Heidelberg, July 2012.

[BDMW10] K.A. Browning, J.F. Dillon, M.T. McQuistan, and A.J. Wolfe. An APN permutation in dimension six. In *Finite Fields: Theory and Applications - FQ9*, volume 518 of *Contemporary Mathematics*, pages 33–42. AMS, 2010.

[BDN$^+$10] Stéphane Badel, Nilay Dagtekin, Jorge Nakahara, Khaled Ouafi, Nicolas Reffé, Pouyan Sepehrdad, Petr Susil, and Serge Vaudenay. ARMADILLO: A multi-purpose cryptographic primitive dedicated to hardware. In Stefan Mangard and François-Xavier Standaert, editors, *Cryptographic Hardware and Embedded Systems – CHES 2010*, volume 6225 of *Lecture Notes in Computer Science*, pages 398–412. Springer, Heidelberg, August 2010.

[BDP15] Alex Biryukov, Patrick Derbez, and Léo Perrin. Differential analysis and meet-in-the-middle attack against round-reduced TWINE. In Gregor Leander, editor, *Fast Software Encryption – FSE 2015*, volume 9054 of *Lecture Notes in Computer Science*, pages 3–27. Springer, Heidelberg, March 2015.

[BDP$^+$16] Guido Bertoni, Joan Daemen, Michaël Peeters, Gilles Van Assche, and Ronny Van Keer. Caesar submission: Ketje v2. Candidate for the CAESAR Competition. See also http://ketje.noekeon.org/, 2016.

[BDPA15] Guido Bertoni, Joan Daemen, Michael Peeters, and Gilles Van Assche. keccak. Cryptology ePrint Archive, Report 2015/389, 2015. http://eprint.iacr.org/2015/389.

[BDPV08] Guido Bertoni, Joan Daemen, Michael Peeters, and Gilles Van Assche. On the indifferentiability of the sponge construction. In Nigel P. Smart, editor, *Advances in Cryptology – EUROCRYPT 2008*, volume

4965 of *Lecture Notes in Computer Science*, pages 181–197. Springer, Heidelberg, April 2008.

[BDPV12] Guido Bertoni, Joan Daemen, Michael Peeters, and Gilles Van Assche. Duplexing the sponge: Single-pass authenticated encryption and other applications. In Ali Miri and Serge Vaudenay, editors, *SAC 2011: 18th Annual International Workshop on Selected Areas in Cryptography*, volume 7118 of *Lecture Notes in Computer Science*, pages 320–337. Springer, Heidelberg, August 2012.

[BDPVA07] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Sponge functions. In *ECRYPT hash workshop*, volume 2007. Citeseer, 2007.

[BDPVA09] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Keccak specifications. Submission to NIST (Round 2), 2009.

[BDPVA13] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Keccak and the sha-3 standardization. Slides of a NIST presentation, http://csrc.nist.gov/groups/ST/hash/sha-3/documents/Keccak-slides-at-NIST.pdf, 2013.

[BDQ04] Alex Biryukov, Christophe De Cannière, and Michaël Quisquater. On multiple linear approximations. In Matthew Franklin, editor, *Advances in Cryptology – CRYPTO 2004*, volume 3152 of *Lecture Notes in Computer Science*, pages 1–22. Springer, Heidelberg, August 2004.

[Bel11] Belarusian State University, National Research Center for Applied Problems of Mathematics and Informatics. “Information technologies. Data protection. Cryptographic algorithms for encryption and integrity control.”. State Standard of Republic of Belarus (STB 34.101.31-2011), 2011. http://apmi.bsu.by/assets/files/std/belt-spec27.pdf.

[Ber08a] Daniel J. Bernstein. Chacha, a variant of Salsa20. SASC 2008 – the State of the Art in Stream Ciphers. See also https://cr.yp.to/chacha.html, 2008.

[Ber08b] Daniel J. Bernstein. The salsa20 family of stream ciphers. In Matthew Robshaw and Olivier Billet, editors, *New Stream Cipher Designs: The eSTREAM Finalists*, volume 4986 of *Lecture Notes in Computer Science*, pages 84–97, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[BFMT15] Thierry Pierre Berger, Julien Francq, Marine Minier, and Gaël Thomas. Extended generalized feistel networks using matrix representation to propose a new lightweight block cipher: Lilliput. *IEEE Transactions on Computers*, PP(99), August 2015.

[BGS$^+$05] Stephen C. Bono, Matthew Green, Adam Stubblefield, Ari Juels, Aviel D. Rubin, and Michael Szydlo. Security analysis of a cryptographically-enabled RFID device. In *Proceedings of the 14th Conference on USENIX Security Symposium - Volume 14*, SSYM’05, pages 1–1, Berkeley, CA, USA, 2005. USENIX Association.

[BI15] Andrey Bogdanov and Takanori Isobe. White-box cryptography revisited: Space-hard ciphers. In Indrajit Ray, Ninghui Li, and Christopher Kruegel:, editors, *ACM CCS 15: 22nd Conference on Computer and Communications Security*, pages 1058–1069. ACM Press, October 2015.

[Bih00] Eli Biham. Cryptanalysis of Patarin's 2-round public key system with S boxes (2R). In Bart Preneel, editor, *Advances in Cryptology – EUROCRYPT 2000*, volume 1807 of *Lecture Notes in Computer Science*, pages 408–416. Springer, Heidelberg, May 2000.

[Bir03] Alex Biryukov. Analysis of involutional ciphers: Khazad and Anubis. In Thomas Johansson, editor, *Fast Software Encryption – FSE 2003*, volume 2887 of *Lecture Notes in Computer Science*, pages 45–53. Springer, Heidelberg, February 2003.

[Bir17] Alex Biryukov. Personal communication, March 2017.

[BIT16] Andrey Bogdanov, Takanori Isobe, and Elmar Tischhauser. Towards practical whitebox cryptography: Optimizing efficiency and space hardness. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *Advances in Cryptology – ASIACRYPT 2016, Part I*, volume 10031 of *Lecture Notes in Computer Science*, pages 126–158. Springer, Heidelberg, December 2016.

[BJK$^+$16] Christof Beierle, Jérémy Jean, Stefan Kölbl, Gregor Leander, Amir Moradi, Thomas Peyrin, Yu Sasaki, Pascal Sasdrich, and Siang Meng Sim. The SKINNY family of block ciphers and its low-latency variant MANTIS. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology – CRYPTO 2016, Part II*, volume 9815 of *Lecture Notes in Computer Science*, pages 123–153. Springer, Heidelberg, August 2016.

[BJM$^+$14] Lejla Batina, Domagoj Jakobovic, Nele Mentens, Stjepan Picek, Antonio De La Piedra, and Dominik Sisejkovic. S-box pipelining using genetic algorithms for high-throughput AES implementations: How fast can we go? In Willi Meier and Debdeep Mukhopadhyay, editors, *Progress in Cryptology - INDOCRYPT 2014: 15th International Conference in Cryptology in India*, volume 8885 of *Lecture Notes in Computer Science*, pages 322–337. Springer, Heidelberg, December 2014.

[BK04] Mihir Bellare and Tadayoshi Kohno. Hash function balance and its impact on birthday attacks. In Christian Cachin and Jan Camenisch, editors, *Advances in Cryptology – EUROCRYPT 2004*, volume 3027 of *Lecture Notes in Computer Science*, pages 401–418. Springer, Heidelberg, May 2004.

[BK16a] Alex Biryukov and Dmitry Khovratovich. Egalitarian computing. In Thorsten Holz and Stefan Savage, editors, *25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016.*, pages 315–326. USENIX Association, 2016.

[BK16b] Alex Biryukov and Dmitry Khovratovich. Equihash: Asymmetric proof-of-work based on the generalized birthday problem. *Proceedings of NDSS'16, 21–24 February 2016, San Diego, CA, USA. ISBN 1-891562-41-X*, 2016.

[BKL+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 Pascal Paillier and Ingrid Verbauwhede, editors, *Cryptographic Hardware and Embedded Systems – CHES 2007*, volume 4727 of *Lecture Notes in Computer Science*, pages 450–466. Springer, Heidelberg, September 2007.

[BKL+11] Andrey Bogdanov, Miroslav Knežević, Gregor Leander, Deniz Toz, Kerem Varici, and Ingrid Verbauwhede. Spongent: A lightweight hash function. In Bart Preneel and Tsuyoshi Takagi, editors, *Cryptographic Hardware and Embedded Systems – CHES 2011*, volume 6917 of *Lecture Notes in Computer Science*, pages 312–325. Springer, Heidelberg, September / October 2011.

[BKLM09] Julia Borghoff, Lars R. Knudsen, Gregor Leander, and Krystian Matusiewicz. Cryptanalysis of C2. In Shai Halevi, editor, *Advances in Cryptology – CRYPTO 2009*, volume 5677 of *Lecture Notes in Computer Science*, pages 250–266. Springer, Heidelberg, August 2009.

[BKP17] Alex Biryukov, Dmitry Khovratovich, and Léo Perrin. Multiset-algebraic cryptanalysis of reduced Kuznyechik, Khazad, and secret SPNs. *IACR Transactions on Symmetric Cryptology*, 2016(2):226–247, 2017.

[BKR16] Mihir Bellare, Daniel Kane, and Phillip Rogaway. Big-key symmetric encryption: Resisting key exfiltration. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology – CRYPTO 2016, Part I*, volume 9814 of *Lecture Notes in Computer Science*, pages 373–402. Springer, Heidelberg, August 2016.

[BKZ11] Alex Biryukov, Ilya Kizhvatov, and Bin Zhang. Cryptanalysis of the Atmel cipher in SecureMemory, CryptoMemory and CryptoRF. In Javier Lopez and Gene Tsudik, editors, *ACNS 11: 9th International Conference on Applied Cryptography and Network Security*, volume 6715 of *Lecture Notes in Computer Science*, pages 91–109. Springer, Heidelberg, June 2011.

[BKZZ16] Foteini Baldimtsi, Aggelos Kiayias, Thomas Zacharias, and Bingsheng Zhang. Indistinguishable proofs of work or knowledge. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *Advances in Cryptology – ASIACRYPT 2016, Part II*, volume 10032 of *Lecture Notes in Computer Science*, pages 902–933. Springer, Heidelberg, December 2016.

[BL10] Carl Bracken and Gregor Leander. A highly nonlinear differentially 4 uniform power mapping that permutes fields of even degree. *Finite Fields and Their Applications*, 16(4):231–242, 2010.

[BL16] Karthikeyan Bhargavan and Gaëtan Leurent. On the practical (in-)security of 64-bit block ciphers: Collision attacks on HTTP over TLS and OpenVPN. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, pages 456–467, New York, NY, USA, 2016. ACM.

[Bla94] Matt Blaze. Protocol failure in the escrowed encryption standard. In *ACM CCS 94: 2nd Conference on Computer and Communications Security*, pages 59–67. ACM Press, 1994.

[BLN15] Daniel J. Bernstein, Tanja Lange, and Ruben Niederhagen. Dual EC: A standardized back door. Cryptology ePrint Archive, Report 2015/767, 2015. http://eprint.iacr.org/2015/767.

[BLNW12] Andrey Bogdanov, Gregor Leander, Kaisa Nyberg, and Meiqin Wang. Integral and multidimensional linear distinguishers with correlation zero. In Xiaoyun Wang and Kazue Sako, editors, *Advances in Cryptology – ASIACRYPT 2012*, volume 7658 of *Lecture Notes in Computer Science*, pages 244–261. Springer, Heidelberg, December 2012.

[BLP04] Alex Biryukov, Joseph Lano, and Bart Preneel. Cryptanalysis of the alleged SecurID hash function. In Mitsuru Matsui and Robert J. Zuccherato, editors, *SAC 2003: 10th Annual International Workshop on Selected Areas in Cryptography*, volume 3006 of *Lecture Notes in Computer Science*, pages 130–144. Springer, Heidelberg, August 2004.

[BLP16] Alex Biryukov, Gaëtan Leurent, and Léo Perrin. Cryptanalysis of Feistel networks with secret round functions. In Orr Dunkelman and Liam Keliher, editors, *Selected Areas in Cryptography – SAC 2015*, volume 9566 of *Lecture Notes in Computer Science*, pages 102–121, Cham, 2016. Springer International Publishing.

[BLR13] Alex Biryukov, Gaëtan Leurent, and Arnab Roy. Cryptanalysis of the “kindle” cipher. In Lars R. Knudsen and Huapeng Wu, editors, *SAC 2012: 19th Annual International Workshop on Selected Areas in Cryptography*, volume 7707 of *Lecture Notes in Computer Science*, pages 86–103. Springer, Heidelberg, August 2013.

[BM15] Céline Blondeau and Marine Minier. Analysis of impossible, integral and zero-correlation attacks on type-II generalized Feistel networks using the matrix method. In Gregor Leander, editor, *Fast Software Encryption – FSE 2015*, volume 9054 of *Lecture Notes in Computer Science*, pages 92–113. Springer, Heidelberg, March 2015.

[BMR+14] Andrey Bogdanov, Florian Mendel, Francesco Regazzoni, Vincent Rijmen, and Elmar Tischhauser. ALE: AES-based lightweight authenticated encryption. In Shiho Moriai, editor, *Fast Software Encryption – FSE 2013*, volume 8424 of *Lecture Notes in Computer Science*, pages 447–466. Springer, Heidelberg, March 2014.

[BN13] Céline Blondeau and Kaisa Nyberg. New links between differential and linear cryptanalysis. In Thomas Johansson and Phong Q. Nguyen, editors, *Advances in Cryptology – EUROCRYPT 2013*, volume 7881 of *Lecture Notes in Computer Science*, pages 388–404. Springer, Heidelberg, May 2013.

[BN14] Céline Blondeau and Kaisa Nyberg. Links between truncated differential and multidimensional linear properties of block ciphers and underlying attack complexities. In Phong Q. Nguyen and Elisabeth

Oswald, editors, *Advances in Cryptology – EUROCRYPT 2014*, volume 8441 of *Lecture Notes in Computer Science*, pages 165–182. Springer, Heidelberg, May 2014.

[BN15] Céline Blondeau and Kaisa Nyberg. Perfect nonlinear functions and cryptography. *Finite Fields and Their Applications*, 32:120 – 147, 2015. Special Issue : Second Decade of FFA.

[BNN$^+$10] Paulo Barreto, Ventzislav Nikov, Svetla Nikova, Vincent Rijmen, and Elmar Tischhauser. Whirlwind: a new cryptographic hash function. *Designs, Codes and Cryptography*, 56(2):141–162, 2010.

[BP14a] Alex Biryukov and Léo Perrin. State of the art in lightweight cryptography. http://cryptolux.org/index.php/Lightweight_Cryptography, 2014.

[BP14b] Céline Blondeau and Léo Perrin. More differentially 6-uniform power functions. *Designs, Codes and Cryptography*, 73(2):487–505, 2014.

[BP15] Alex Biryukov and Léo Perrin. On reverse-engineering S-boxes with hidden design criteria or structure. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *Advances in Cryptology – CRYPTO 2015, Part I*, volume 9215 of *Lecture Notes in Computer Science*, pages 116–140. Springer, Heidelberg, August 2015.

[BP17] Alex Biryukov and Léo Perrin. A generic framework and examples of symmetrically and asymmetrically hard functions. Cryptology ePrint Archive, Report 2017/414, 2017. http://eprint.iacr.org/2017/414.

[BPU16] Alex Biryukov, Léo Perrin, and Aleksei Udovenko. Reverse-engineering the S-box of streebog, kuznyechik and STRIBOBr1. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology – EUROCRYPT 2016, Part I*, volume 9665 of *Lecture Notes in Computer Science*, pages 372–402. Springer, Heidelberg, May 2016.

[BR00a] Paulo Barreto and Vincent Rijmen. The ANUBIS Block Cipher. First Open NESSIE Workshop, see also https://www.cosic.esat.kuleuven.be/nessie/workshop/submissions/anubis.zip, 2000.

[BR00b] Paulo Barreto and Vincent Rijmen. The Khazad legacy-level Block Cipher. First Open NESSIE Workshop, see also https://www.cosic.esat.kuleuven.be/nessie/workshop/submissions/khazad.zip, 2000.

[BR00c] Paulo Barreto and Vincent Rijmen. The Whirlpool hashing function. First open NESSIE Workshop, see also https://www.cosic.esat.kuleuven.be/nessie/workshop/submissions/whirlpool.zip, 2000.

[BR11] Andrey Bogdanov and Vincent Rijmen. Linear hulls with correlation zero and linear cryptanalysis of block ciphers. Cryptology ePrint Archive, Report 2011/123, 2011. http://eprint.iacr.org/2011/123.

[BR15] Elaine Barker and Allen Roginsky. NIST special publication 800-131a revision 1. *NIST Special Publication*, 800(131A):1–29, 2015.

[BRS67] E.R. Berlekamp, H. Rumsey, and G. Solomon. On the solution of algebraic equations over finite fields. *Inform. Contr.*, 12(5):553–564, October 1967.

[BS91] Eli Biham and Adi Shamir. Differential cryptanalysis of DES-like cryptosystems. *Journal of Cryptology*, 4(1):3–72, 1991.

[BS00] Alex Biryukov and Adi Shamir. Cryptanalytic time/memory/data tradeoffs for stream ciphers. In Tatsuaki Okamoto, editor, *Advances in Cryptology – ASIACRYPT 2000*, volume 1976 of *Lecture Notes in Computer Science*, pages 1–13. Springer, Heidelberg, December 2000.

[BS01] Alex Biryukov and Adi Shamir. Structural cryptanalysis of SASAS. In Birgit Pfitzmann, editor, *Advances in Cryptology – EUROCRYPT 2001*, volume 2045 of *Lecture Notes in Computer Science*, pages 394–405. Springer, Heidelberg, May 2001.

[BS10] Alex Biryukov and Adi Shamir. Structural cryptanalysis of SASAS. *Journal of Cryptology*, 23(4):505–518, October 2010.

[BS15] Adnan Baysal and Suhap Sahin. RoadRunneR: A small and fast bitslice block cipher for low cost 8-bit processors. Cryptology ePrint Archive, Report 2015/906, 2015. http://eprint.iacr.org/2015/906.

[BŞ16] Adnan Baysal and Sühap Şahin. RoadRunneR: A small and fast bitslice block cipher for low cost 8-bit processors. In Tim Güneysu, Gregor Leander, and Amir Moradi, editors, *Lightweight Cryptography for Security and Privacy — LightSec 2015*, volume 9542 of *Lecture Notes in Computer Science*, pages 58–76, Berlin, Heidelberg, 2016. Springer International Publishing.

[BSK96] F.J. Bruwer, W. Smit, and G.J. Kuhn. Microchips and remote control devices comprising same, May 1996. US Patent 5,517,187.

[BSS+13] Ray Beaulieu, Douglas Shors, Jason Smith, Stefan Treatman-Clark, Bryan Weeks, and Louis Wingers. The SIMON and SPECK families of lightweight block ciphers. Cryptology ePrint Archive, Report 2013/404, 2013. http://eprint.iacr.org/2013/404.

[BSW01] Alex Biryukov, Adi Shamir, and David Wagner. Real time cryptanalysis of A5/1 on a PC. In Bruce Schneier, editor, *Fast Software Encryption – FSE 2000*, volume 1978 of *Lecture Notes in Computer Science*, pages 1–18. Springer, Heidelberg, April 2001.

[BTCS+15] Ray Beaulieu, Stefan Treatman-Clark, Douglas Shors, Bryan Weeks, Jason Smith, and Louis Wingers. The SIMON and SPECK lightweight block ciphers. In *Design Automation Conference (DAC), 2015 52nd ACM/EDAC/IEEE*, pages 1–6. IEEE, 2015.

[BTT12] Carl Bracken, Chik How Tan, and Yin Tan. Binomial differentially 4 uniform permutations with high nonlinearity. *Finite Fields and Their Applications*, 18(3):537–546, 2012.

[BVC16] Alex Biryukov, Vesselin Velichkov, and Yann Le Corre. Automatic search for the best trails in ARX: Application to block cipher speck. In Thomas Peyrin, editor, *Fast Software Encryption – FSE 2016*, volume 9783 of *Lecture Notes in Computer Science*, pages 289–310. Springer, Heidelberg, March 2016.

[BW99] Alex Biryukov and David Wagner. Slide attacks. In Lars R. Knudsen, editor, *Fast Software Encryption – FSE'99*, volume 1636 of *Lecture Notes in Computer Science*, pages 245–259. Springer, Heidelberg, March 1999.

[BW12] Andrey Bogdanov and Meiqin Wang. Zero correlation linear cryptanalysis with reduced data complexity. In Anne Canteaut, editor, *Fast Software Encryption – FSE 2012*, volume 7549 of *Lecture Notes in Computer Science*, pages 29–48. Springer, Heidelberg, March 2012.

[Can05] D. Canright. A very compact S-box for AES. In Josyula R. Rao and Berk Sunar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2005*, volume 3659 of *Lecture Notes in Computer Science*, pages 441–455. Springer, Heidelberg, August / September 2005.

[Can06] Christophe De Cannière. Trivium: A stream cipher construction inspired by block cipher design principles. In Sokratis K. Katsikas, Javier Lopez, Michael Backes, Stefanos Gritzalis, and Bart Preneel, editors, *ISC 2006: 9th International Conference on Information Security*, volume 4176 of *Lecture Notes in Computer Science*, pages 171–186. Springer, Heidelberg, August / September 2006.

[Car11] Claude Carlet. Relating three nonlinearity parameters of vectorial functions and building apn functions from bent functions. *Des. Codes Cryptography*, 59(1-3):89–109, April 2011.

[CBW08] Nicolas Courtois, Gregory V. Bard, and David Wagner. Algebraic and slide attacks on KeeLoq. In Kaisa Nyberg, editor, *Fast Software Encryption – FSE 2008*, volume 5086 of *Lecture Notes in Computer Science*, pages 97–115. Springer, Heidelberg, February 2008.

[CCZ98] Claude Carlet, Pascale Charpin, and Victor Zinoviev. Codes, bent functions and permutations suitable for des-like cryptosystems. *Designs, Codes and Cryptography*, 15(2):125–156, 1998.

[CDK09] Christophe De Cannière, Orr Dunkelman, and Miroslav Knežević. KATAN and KTANTAN - a family of small and efficient hardware-oriented block ciphers. In Christophe Clavier and Kris Gaj, editors, *Cryptographic Hardware and Embedded Systems – CHES 2009*, volume 5747 of *Lecture Notes in Computer Science*, pages 272–288. Springer, Heidelberg, September 2009.

[CDL16] Anne Canteaut, Sébastien Duval, and Gaëtan Leurent. Construction of lightweight s-boxes using feistel and misty structures. In Orr Dunkelman and Liam Keliher, editors, *Selected Areas in Cryptography – SAC 2015*, volume 9566 of *Lecture Notes in Computer Science*, pages 373–393, Cham, 2016. Springer International Publishing.

[CDP17] Anne Canteaut, Sébastien Duval, and Léo Perrin. A generalisation of Dillon's APN permutation with the best known differential and nonlinear properties for all fields of size $2^{4k+2}$. *IEEE Transactions on Information Theory*, (to appear), 2017.

[CF09] Claude Carlet and Keqin Feng. An infinite class of balanced vectorial boolean functions with optimum algebraic immunity and good nonlinearity. In Yeow Meng Chee, Chao Li, San Ling, Huaxiong Wang, and Chaoping Xing, editors, *Coding and Cryptology: IWCC*, volume 5557 of *Lecture Notes in Computer Science*, pages 1–11. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[CFG+15] Anne Canteaut, Thomas Fuhr, Henri Gilbert, María Naya-Plasencia, and Jean-René Reinhard. Multiple differential cryptanalysis of round-reduced PRINCE. In Carlos Cid and Christian Rechberger, editors, *Fast Software Encryption – FSE 2014*, volume 8540 of *Lecture Notes in Computer Science*, pages 591–610. Springer, Heidelberg, March 2015.

[ÇKB12] Mustafa Çoban, Ferhat Karakoç, and Özkan Boztas. Biclique cryptanalysis of TWINE. In Josef Pieprzyk, Ahmad-Reza Sadeghi, and Mark Manulis, editors, *CANS 12: 11th International Conference on Cryptology and Network Security*, volume 7712 of *Lecture Notes in Computer Science*, pages 43–55. Springer, Heidelberg, December 2012.

[CLNP16] Anne Canteaut, Virginie Lallemand, and María Naya-Plasencia. Related-key attack on full-round PICARO. In Orr Dunkelman and Liam Keliher, editors, *Selected Areas in Cryptography – SAC 2015*, volume 9566 of *Lecture Notes in Computer Science*, pages 86–101, Cham, 2016. Springer International Publishing.

[CNO08] Nicolas T. Courtois, Karsten Nohl, and Sean O'Neil. Algebraic attacks on the crypto-1 stream cipher in mifare classic and oyster cards. Cryptology ePrint Archive, Report 2008/166, 2008. http://eprint.iacr.org/2008/166.

[Cop94] Don Coppersmith. The Data Encryption Standard (DES) and its strength against attacks. *IBM journal of research and development*, 38(3):243–250, 1994.

[CV95] Florent Chabaud and Serge Vaudenay. Links between differential and linear cryptoanalysis. In Alfredo De Santis, editor, *Advances in Cryptology – EUROCRYPT'94*, volume 950 of *Lecture Notes in Computer Science*, pages 356–365. Springer, Heidelberg, May 1995.

[CY04] Scott Contini and Yiqun Lisa Yin. Fast software-based attacks on SecurID. In Bimal K. Roy and Willi Meier, editors, *Fast Software Encryption – FSE 2004*, volume 3017 of *Lecture Notes in Computer Science*, pages 454–471. Springer, Heidelberg, February 2004.

[CYK+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 Aikaterini

Mitrokotsa and Serge Vaudenay, editors, *AFRICACRYPT 12: 5th International Conference on Cryptology in Africa*, volume 7374 of *Lecture Notes in Computer Science*, pages 270–286. Springer, Heidelberg, July 2012.

[Dae16] Joan Daemen. Spectral characterization of iterating lossy mappings. Cryptology ePrint Archive, Report 2016/090, 2016. http://eprint.iacr.org/2016/090.

[DBG+15] Dumitru-Daniel Dinu, Alex Biryukov, Johann Großschädl, Dmitry Khovratovich, Yann Le Corre, and Léo Perrin. FELICS – Fair Evaluation of Lightweight Cryptographic Systems. In *NIST Workshop on Lightweight Cryptography 2015*. National Institute of Standards and Technology (NIST), 2015.

[DDKL15] Itai Dinur, Orr Dunkelman, Thorsten Kranz, and Gregor Leander. Decomposing the ASASA block cipher construction. Cryptology ePrint Archive, Report 2015/507, 2015. http://eprint.iacr.org/2015/507.

[DDKS15] Itai Dinur, Orr Dunkelman, Nathan Keller, and Adi Shamir. New attacks on Feistel structures with improved memory complexities. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *Advances in Cryptology – CRYPTO 2015, Part I*, volume 9215 of *Lecture Notes in Computer Science*, pages 433–454. Springer, Heidelberg, August 2015.

[DEKM17] Christoph Dobraunig, Maria Eichlseder, Daniel Kales, and Florian Mendel. Practical key-recovery attack on mantis5. *IACR Transactions on Symmetric Cryptology*, 2016(2):248–260, 2017.

[DEMS16] C. Dobraunig, M. Eichlseder, F. Mendel, and M Schläffer. Ascon v1.2. Candidate for the CAESAR Competition. See also http://ascon.iaik.tugraz.at/, 2016.

[Dev16] The Sage Developers. *SageMath, the Sage Mathematics Software System (Version 7.2)*, 2016. http://www.sagemath.org.

[DFKP15] Stefan Dziembowski, Sebastian Faust, Vladimir Kolmogorov, and Krzysztof Pietrzak. Proofs of space. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *Advances in Cryptology – CRYPTO 2015, Part II*, volume 9216 of *Lecture Notes in Computer Science*, pages 585–605. Springer, Heidelberg, August 2015.

[DGV94] Joan Daemen, René Govaerts, and Joos Vandewalle. A new approach to block cipher design. In Ross J. Anderson, editor, *Fast Software Encryption – FSE'93*, volume 809 of *Lecture Notes in Computer Science*, pages 18–32. Springer, Heidelberg, December 1994.

[DHW+12] B. Driessen, R. Hund, C. Willems, C. Paar, and T. Holz. Don't trust satellite phones: A security analysis of two satphone standards. In *2012 IEEE Symposium on Security and Privacy*, pages 128–142, May 2012.

[Dic96] Leonard Eugene Dickson. The analytic representation of substitutions on a power of a prime number of letters with a discussion of the linear group. *Annals of Mathematics*, 11(1/6):65–120, 1896.

[DKR97] Joan Daemen, Lars R. Knudsen, and Vincent Rijmen. The block cipher Square. In Eli Biham, editor, *Fast Software Encryption – FSE'97*, volume 1267 of *Lecture Notes in Computer Science*, pages 149–165. Springer, Heidelberg, January 1997.

[DKS12] Orr Dunkelman, Nathan Keller, and Adi Shamir. Minimalism in cryptography: The Even-Mansour scheme revisited. In David Pointcheval and Thomas Johansson, editors, *Advances in Cryptology – EUROCRYPT 2012*, volume 7237 of *Lecture Notes in Computer Science*, pages 336–354. Springer, Heidelberg, April 2012.

[DLCK+15] Dumitru-Daniel Dinu, Yann Le Corre, Dmitry Khovratovich, Léo Perrin, Johann Großschädl, and Alex Biryukov. Triathlon of lightweight block ciphers for the internet of things. In *NIST Workshop on Lightweight Cryptography 2015*. National Institute of Standards and Technology (NIST), 2015.

[DLCK+17] Dumitru-Daniel Dinu, Yann Le Corre, Dmitry Khovratovich, Léo Perrin, Johann Großschädl, and Alex Biryukov. Triathlon of lightweight block ciphers for the internet of things. *Journal of Cryptology Engineering*, page To appear, 2017.

[DLPR14] Cécile Delerablée, Tancrède Lepoint, Pascal Paillier, and Matthieu Rivain. White-box security notions for symmetric encryption schemes. In Tanja Lange, Kristin Lauter, and Petr Lisonek, editors, *SAC 2013: 20th Annual International Workshop on Selected Areas in Cryptography*, volume 8282 of *Lecture Notes in Computer Science*, pages 247–264. Springer, Heidelberg, August 2014.

[Dol10a] V. Dolmatov. Gost 28147-89: Encryption, decryption, and message authentication code (mac) algorithms. http://www.rfc-editor.org/rfc/rfc5830.txt, March 2010. RFC 5830.

[Dol10b] V. Dolmatov. Gost r 34.11-94: Hash function algorithm. http://www.rfc-editor.org/rfc/rfc5831.txt, March 2010. RFC 5831.

[DP15] Patrick Derbez and Léo Perrin. Meet-in-the-middle attacks and structural analysis of round-reduced PRINCE. In Gregor Leander, editor, *Fast Software Encryption – FSE 2015*, volume 9054 of *Lecture Notes in Computer Science*, pages 190–216. Springer, Heidelberg, March 2015.

[DPU+16a] Daniel Dinu, Léo Perrin, Aleksei Udovenko, Vesselin Velichkov, Johann Großschädl, and Alex Biryukov. Design strategies for ARX with provable bounds: Sparx and LAX. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *Advances in Cryptology – ASIACRYPT 2016, Part I*, volume 10031 of *Lecture Notes in Computer Science*, pages 484–513. Springer, Heidelberg, December 2016.

[DPU$^+$16b] Daniel Dinu, Léo Perrin, Aleksei Udovenko, Vesselin Velichkov, Johann Großschädl, and Alex Biryukov. Design strategies for ARX with provable bounds: SPARX and LAX (full version). Cryptology ePrint Archive, Report 2016/984, 2016. http://eprint.iacr.org/2016/984.

[DPV01] Joan Daemen, Michael Peeters, and Gilles Van Assche. Bitslice ciphers and power analysis attacks. In Bruce Schneier, editor, *Fast Software Encryption – FSE 2000*, volume 1978 of *Lecture Notes in Computer Science*, pages 134–149. Springer, Heidelberg, April 2001.

[DPVAR00] Joan Daemen, Michaël Peeters, Gilles Van Assche, and Vincent Rijmen. Nessie proposal: NOEKEON. First Open NESSIE Workshop, see also http://gva.noekeon.org/papers/2000-NESSIE-Noekeon-Spec.pdf, 2000.

[DR98] Joan Daemen and Vincent Rijmen. AES proposal: Rijndael. AES submission. See also http://csrc.nist.gov/archive/aes/rijndael/, 1998.

[DR00] Joan Daemen and Vincent Rijmen. The block cipher BKSQ. In Jean-Jacques Quisquater and Bruce Schneier, editors, *Smart Card Research and Applications: Third International Conference, CARDIS'98, Louvain-la-Neuve, Belgium, September 14-16, 1998. Proceedings*, volume 1820 of *Lecture Notes in Computer Science*, pages 236–245, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg.

[DR01] Joan Daemen and Vincent Rijmen. The wide trail design strategy. In Bahram Honary, editor, *8th IMA International Conference on Cryptography and Coding*, volume 2260 of *Lecture Notes in Computer Science*, pages 222–238. Springer, Heidelberg, December 2001.

[DR02] Joan Daemen and Vincent Rijmen. *The design of Rijndael: AES-the advanced encryption standard*. Information Security and Cryptography. Springer-Verlag Berlin Heidelberg, 2002.

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

[DRL11] M. David, D. C. Ranasinghe, and T. Larsen. A2U2: A stream cipher for printed electronics RFID tags. In *2011 IEEE International Conference on RFID*, pages 176–183, April 2011.

[DRS15] S. M. Dehnavi, A. Mahmoodi Rishakani, and M. R. Mirzaee Shamsabad. A more explicit formula for linear probabilities of modular addition modulo a power of two. Cryptology ePrint Archive, Report 2015/026, 2015. http://eprint.iacr.org/2015/026.

[EJ03] Patrik Ekdahl and Thomas Johansson. A new version of the stream cipher SNOW. In Kaisa Nyberg and Howard M. Heys, editors, *SAC 2002: 9th Annual International Workshop on Selected Areas in Cryptography*, volume 2595 of *Lecture Notes in Computer Science*, pages 47–61. Springer, Heidelberg, August 2003.

[EKM$^+$08] Thomas Eisenbarth, Timo Kasper, Amir Moradi, Christof Paar, Mahmoud Salmasizadeh, and Mohammad T. Manzuri Shalmani. On the power of power analysis in the real world: A complete break of the keeloqcode hopping scheme. In David Wagner, editor, *Advances in Cryptology – CRYPTO 2008*, volume 5157 of *Lecture Notes in Computer Science*, pages 203–220. Springer, Heidelberg, August 2008.

[EM97] Shimon Even and Yishay Mansour. A construction of a cipher from a single pseudorandom permutation. *Journal of Cryptology*, 10(3):151–162, 1997.

[ES03] Niklas Eén and Niklas Sörensson. An extensible sat-solver. In Enrico Giunchiglia and Armando Tacchella, editors, *Theory and Applications of Satisfiability Testing, 6th International Conference, SAT 2003. Santa Margherita Ligure, Italy, May 5-8, 2003 Selected Revised Papers*, volume 2919 of *Lecture Notes in Computer Science*, pages 502–518. Springer, 2003.

[ES04] Niklas Eén and Niklas Sörensson. An extensible sat-solver. In Enrico Giunchiglia and Armando Tacchella, editors, *Theory and Applications of Satisfiability Testing: 6th International Conference, SAT 2003, Santa Margherita Ligure, Italy, May 5-8, 2003, Selected Revised Papers*, volume 2919 of *Lecture Notes in Computer Science*, pages 502–518, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[ESSS12] Daniel Engels, Markku-Juhani O. Saarinen, Peter Schweitzer, and Eric M. Smith. The Hummingbird-2 lightweight authenticated encryption algorithm. In Ari Juels and Christof Paar, editors, *RFID. Security and Privacy: 7th International Workshop, RFIDSec 2011, Amherst, USA, June 26-28, 2011, Revised Selected Papers*, volume 7055 of *Lecture Notes in Computer Science*, pages 19–31, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[ETS06a] ETSI/SAGE. Specification of the 3GPP confidentiality and integrity algorithms UEA2 & UIA2. Document 2: SNOW 3G specification. Technical report, ETSI/Sage, September 2006. Available at http://www.gsma.com/aboutus/wp-content/uploads/2014/12/snow3gspec.doc (Microsoft Word document).

[ETS06b] ETSI/Sage. Specification of the 3GPP Confidentiality and Integrity Algorithms UEA2 & UIA2. Document 5 : Design and Evaluation Report. Technical report, ETSI/Sage, September 2006. Available at http://www.gsma.com/aboutus/wp-content/uploads/2014/12/uea2designevaluation.pdf.

[ETS11] ETSI/Sage. Specification of the 3GPP Confidentiality and Integrity Algorithms 128-EEA3 & 128-EIA3. Document 4 : Design and Evaluation Report. Technical report, ETSI/Sage, September 2011. Available at http://www.gsma.com/aboutus/wp-content/uploads/2014/12/EEA3_EIA3_Design_Evaluation_v2_0.pdf.

[Fed12] Federal Agency on Technical Regulation and Metrology (GOST). Gost r 34.11-2012: Streebog hash function, 2012. https://www.streebog.net/.

[Fed15] Federal Agency on Technical Regulation and Metrology (GOST). (GOST R 34.12–2015) information technology – cryptographic data security – block ciphers, 2015. http://tc26.ru/en/standard/gost/GOST_R_34_12_2015_ENG.pdf.

[FKKM16] Pierre-Alain Fouque, Pierre Karpman, Paul Kirchner, and Brice Minaud. Efficient and provable white-box primitives. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *Advances in Cryptology – ASIACRYPT 2016, Part I*, volume 10031 of *Lecture Notes in Computer Science*, pages 159–188. Springer, Heidelberg, December 2016.

[FKL+01] Niels Ferguson, John Kelsey, Stefan Lucks, Bruce Schneier, Michael Stay, David Wagner, and Doug Whiting. Improved cryptanalysis of Rijndael. In Bruce Schneier, editor, *Fast Software Encryption – FSE 2000*, volume 1978 of *Lecture Notes in Computer Science*, pages 213–230. Springer, Heidelberg, April 2001.

[FL01] Scott R. Fluhrer and Stefan Lucks. Analysis of the E0 encryption system. In Serge Vaudenay and Amr M. Youssef, editors, *SAC 2001: 8th Annual International Workshop on Selected Areas in Cryptography*, volume 2259 of *Lecture Notes in Computer Science*, pages 38–48. Springer, Heidelberg, August 2001.

[FLY09] Keqin Feng, Qunying Liao, and Jing Yang. Maximal values of generalized algebraic immunity. *Designs, Codes and Cryptography*, 50(2):243–252, 2009.

[FO90] Philippe Flajolet and Andrew M. Odlyzko. Random mapping statistics. In Jean-Jacques Quisquater and Joos Vandewalle, editors, *Advances in Cryptology – EUROCRYPT'89*, volume 434 of *Lecture Notes in Computer Science*, pages 329–354. Springer, Heidelberg, April 1990.

[FWS+03] Niels Ferguson, Doug Whiting, Bruce Schneier, John Kelsey, Stefan Lucks, and Tadayoshi Kohno. Helix: Fast encryption and authentication in a single cryptographic primitive. In Thomas Johansson, editor, *Fast Software Encryption – FSE 2003*, volume 2887 of *Lecture Notes in Computer Science*, pages 330–346. Springer, Heidelberg, February 2003.

[FY95] Yair Frankel and Moti Yung. Escrow encryption systems visited: Attacks, analysis and designs. In Don Coppersmith, editor, *Advances in Cryptology – CRYPTO'95*, volume 963 of *Lecture Notes in Computer Science*, pages 222–235. Springer, Heidelberg, August 1995.

[GC91] Henri Gilbert and Guy Chassé. A statistical attack of the FEAL-8 cryptosystem. In Alfred J. Menezes and Scott A. Vanstone, editors, *Advances in Cryptology – CRYPTO'90*, volume 537 of *Lecture Notes in Computer Science*, pages 22–33. Springer, Heidelberg, August 1991.

[GdKGV14] Flavio D. Garcia, Gerhard de Koning Gans, and Roel Verdult. Wirelessly lockpicking a smart card reader. *International Journal of Information Security*, 13(5):403–420, 2014.

[GGH+98] Henri Gilbert, Marc Girault, Philippe Hoogvorst, Thomas Pornin, Guillaume Poupard, Jacques Stern, Serge Vaudenay, et al. Decorrelated Fast Cipher: an AES candidate. Candidate for the AES competition. Pdf available online at https://infoscience.epfl.ch/record/99484/files/GG+98.ps, 1998.

[GGNS13] Benoît Gérard, Vincent Grosso, María Naya-Plasencia, and François-Xavier Standaert. Block ciphers that are easier to mask: How far can we go? In Guido Bertoni and Jean-Sébastien Coron, editors, *Cryptographic Hardware and Embedded Systems – CHES 2013*, volume 8086 of *Lecture Notes in Computer Science*, pages 383–399. Springer, Heidelberg, August 2013.

[GJNS17] Jian Guo, Jérémy Jean, Ivica Nikolic, and Yu Sasaki. Meet-in-the-middle attacks on classes of contracting and expanding Feistel constructions. *IACR Transactions on Symmetric Cryptology*, 2016(2):307–337, 2017.

[GKM+11] Praveen Gauravaram, Lars R. Knudsen, Krystian Matusiewicz, Florian Mendel, Christian Rechberger, Martin Schläffer, and Søren S. Thomsen. Grøstl – a SHA-3 candidate. Submission to the SHA-3 competition. See also www.groestl.info/Groestl.pdf, 2011.

[GLS+14] Vincent Grosso, Gaëtan Leurent, François-Xavier Standaert, Kerem Varici, François Durvaux Anthony Journault, Lubos Gaspar, and Stéphanie Kerckhof. SCREAM & iSCREAM Side-Channel Resistant Authenticated Encryption with Masking. Candidate for the CAESAR Competition. See also http://perso.uclouvain.be/fstandae/SCREAM/, 2014.

[GLSV15] Vincent Grosso, Gaëtan Leurent, François-Xavier Standaert, and Kerem Varici. LS-designs: Bitslice encryption for efficient masked software implementations. In Carlos Cid and Christian Rechberger, editors, *Fast Software Encryption – FSE 2014*, volume 8540 of *Lecture Notes in Computer Science*, pages 18–37. Springer, Heidelberg, March 2015.

[GNL11] Zheng Gong, Svetla Nikova, and Yee Wei Law. KLEIN: A new family of lightweight block ciphers. In Ari Juels and Christof Paar, editors, *RFID. Security and Privacy - 7th International Workshop, RFIDSec*, volume 7055 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2011.

[Gol97] Jovan Dj. Golic. Cryptanalysis of alleged A5 stream cipher. In Walter Fumy, editor, *Advances in Cryptology – EUROCRYPT'97*, volume 1233 of *Lecture Notes in Computer Science*, pages 239–255. Springer, Heidelberg, May 1997.

[Gol13] Jovan Dj. Golic. Cryptanalytic attacks on MIFARE classic protocol. In Ed Dawson, editor, *Topics in Cryptology – CT-RSA 2013*, volume 7779 of *Lecture Notes in Computer Science*, pages 239–258. Springer, Heidelberg, February / March 2013.

[Göl15] Faruk Göloglu. Almost perfect nonlinear trinomials and hexanomials. *Finite Fields and Their Applications*, 33:258–282, 2015.

[GPP11] Jian Guo, Thomas Peyrin, and Axel Poschmann. The PHOTON family of lightweight hash functions. In Phillip Rogaway, editor, *Advances in Cryptology – CRYPTO 2011*, volume 6841 of *Lecture Notes in Computer Science*, pages 222–239. Springer, Heidelberg, August 2011.

[GPPR11] Jian Guo, Thomas Peyrin, Axel Poschmann, and Matthew J. B. Robshaw. The LED block cipher. In Bart Preneel and Tsuyoshi Takagi, editors, *Cryptographic Hardware and Embedded Systems – CHES 2011*, volume 6917 of *Lecture Notes in Computer Science*, pages 326–341. Springer, Heidelberg, September / October 2011.

[GPT15] Henri Gilbert, Jérôme Plût, and Joana Treger. Key-recovery attack on the ASASA cryptosystem with expanding S-boxes. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *Advances in Cryptology – CRYPTO 2015, Part I*, volume 9215 of *Lecture Notes in Computer Science*, pages 475–490. Springer, Heidelberg, August 2015.

[GR16] Lorenzo Grassi and Christian Rechberger. Practical low data-complexity subspace-trail cryptanalysis of round-reduced PRINCE. In Orr Dunkelman and Somitra Kumar Sanadhya, editors, *Progress in Cryptology – INDOCRYPT 2016*, volume 10095 of *Lecture Notes in Computer Science*, pages 322–342. Springer International Publishing, 2016.

[GT16] Peter Gazi and Stefano Tessaro. Provably robust sponge-based PRNGs and KDFs. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology – EUROCRYPT 2016, Part I*, volume 9665 of *Lecture Notes in Computer Science*, pages 87–116. Springer, Heidelberg, May 2016.

[GvRVWS10] Flavio D. Garcia, Peter van Rossum, Roel Verdult, and Ronny Wichers Schreur. Dismantling SecureMemory, CryptoMemory and CryptoRF. In *Proceedings of the 17th ACM Conference on Computer and Communications Security*, CCS '10, pages 250–259, New York, NY, USA, 2010. ACM.

[Hel80] Martin Hellman. A cryptanalytic time-memory trade-off. *IEEE transactions on Information Theory*, 26(4):401–406, 1980.

[HIK$^+$11] Shoichi Hirose, Kota Ideguchi, Hidenori Kuwakado, Toru Owada, Bart Preneel, and Hirotaka Yoshida. A lightweight 256-bit hash function for hardware and low-end devices: Lesamnta-LW. In Kyung Hyune Rhee and DaeHun Nyang, editors, *ICISC 10: 13th International Conference on Information Security and Cryptology*, volume 6829 of *Lecture Notes in Computer Science*, pages 151–168. Springer, Heidelberg, December 2011.

[HJ11] Martin Hell and Thomas Johansson. Breaking the stream ciphers F-FCSR-H and F-FCSR-16 in real time. *Journal of Cryptology*, 24(3):427–445, July 2011.

[HJKK13] T. Helleseth, C. J. A. Jansen, O. Kazymyrov, and A. Kholosha. State space cryptanalysis of the mickey cipher. In *2013 Information Theory and Applications Workshop (ITA)*, pages 1–10, Feb 2013.

[HJM07] Martin Hell, Thomas Johansson, and Willi Meier. Grain: a stream cipher for constrained environments. *Int. J. Wire. Mob. Comput.*, 2(1):86–93, May 2007.

[HK05] Jin Hong and Woo-Hwan Kim. TMD-tradeoff and state entropy loss considerations of streamcipher MICKEY. In Subhamoy Maitra, C. E. Veni Madhavan, and Ramarathnam Venkatesan, editors, *Progress in Cryptology - INDOCRYPT 2005: 6th International Conference in Cryptology in India*, volume 3797 of *Lecture Notes in Computer Science*, pages 169–182. Springer, Heidelberg, December 2005.

[HKM17] Matthias Hamann, Matthias Krause, and Willi Meier. LIZARD – A lightweight stream cipher for power-constrained devices. *IACR Transactions on Symmetric Cryptology*, 2017(1):45–79, 2017.

[HLK⁺14] Deukjo Hong, Jung-Keun Lee, Dong-Chan Kim, Daesung Kwon, Kwon Ho Ryu, and Dong-Geon Lee. LEA: A 128-bit block cipher for fast encryption on common processors. In Yongdae Kim, Heejo Lee, and Adrian Perrig, editors, *WISA 13: 14th International Workshop on Information Security Applications*, volume 8267 of *Lecture Notes in Computer Science*, pages 3–27. Springer, Heidelberg, August 2014.

[HMS⁺76] M. Hellman, R. Merkle, R. Schroeppel, L. Washington, W. Diffie, S. Pohlig, , and P. Schweitzer. Results of an initial attempt to cryptanalyze the NBS Data Encryption Standard. Technical report, Stanford University, Information Systems Laboratory, 1976. Available at http://www.merkle.com/papers/Attempt%20to%20Cryptanalyze%20DES%201976-11-10.pdf.

[HN⁺05] Johan Hastad, Mats Naslund, et al. The stream cipher polar bear. *eSTREAM, ECRYPT Stream Cipher Project Report*, 21:2005, 2005.

[HN10] Risto M. Hakala and Kaisa Nyberg. On the nonlinearity of discrete logarithm in $\mathbb{F}_{2^n}$. In Claude Carlet and Alexander Pott, editors, *Sequences and Their Applications – SETA*, volume 6338 of *Lecture Notes in Computer Science*, pages 333–345. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[HSH⁺06] Deukjo Hong, Jaechul Sung, Seokhie Hong, Jongin Lim, Sangjin Lee, Bon-Seok Koo, Changhoon Lee, Donghoon Chang, Jesang Lee, Kitae Jeong, Hyun Kim, Jongsung Kim, and Seongtaek Chee. HIGHT: A new block cipher suitable for low-resource device. In Louis Goubin and Mitsuru Matsui, editors, *Cryptographic Hardware and Embedded Systems – CHES 2006*, volume 4249 of *Lecture Notes in Computer Science*, pages 46–59. Springer, Heidelberg, October 2006.

[IKD⁺08] Sebastiaan Indesteege, Nathan Keller, Orr Dunkelman, Eli Biham, and Bart Preneel. A practical attack on KeeLoq. In Nigel P. Smart, editor, *Advances in Cryptology – EUROCRYPT 2008*, volume 4965 of *Lec-*

*ture Notes in Computer Science*, pages 1–18. Springer, Heidelberg, April 2008.

[ISSK09] Maryam Izadi, Babak Sadeghiyan, Seyed Saeed Sadeghian, and Hossein Arabnezhad Khanooki. MIBS: A new lightweight block cipher. In Juan A. Garay, Atsuko Miyaji, and Akira Otsuka, editors, *CANS 09: 8th International Conference on Cryptology and Network Security*, volume 5888 of *Lecture Notes in Computer Science*, pages 334–348. Springer, Heidelberg, December 2009.

[JH98] M. J. Jacobson and K. Huber. The MAGENTA block cipher algorithm. AES submission. See also http://www.madchat.fr/crypto/hash-lib-algo/magenta/magenta.pdf., 1998.

[JK01] Thomas Jakobsen and Lars R. Knudsen. Attacks on block ciphers of low algebraic degree. *Journal of Cryptology*, 14(3):197–210, 2001.

[JK12] Goce Jakimoski and Samant Khajuria. ASC-1: An authenticated encryption stream cipher. In Ali Miri and Serge Vaudenay, editors, *SAC 2011: 18th Annual International Workshop on Selected Areas in Cryptography*, volume 7118 of *Lecture Notes in Computer Science*, pages 356–372. Springer, Heidelberg, August 2012.

[JN16] Jérémy Jean and Ivica Nikolic. Efficient design strategies based on the AES round function. In Thomas Peyrin, editor, *Fast Software Encryption – FSE 2016*, volume 9783 of *Lecture Notes in Computer Science*, pages 334–353. Springer, Heidelberg, March 2016.

[JNP14a] Jérémy Jean, Ivica Nikolìc, and Thomas Peyrin. Joltik v1. Candidate for the CAESAR Competition. See also http://www1.spms.ntu.edu.sg/~syllab/m/index.php/Joltik, 2014.

[JNP⁺14b] Jérémy Jean, Ivica Nikolic, Thomas Peyrin, Lei Wang, and Shuang Wu. Security analysis of PRINCE. In Shiho Moriai, editor, *Fast Software Encryption – FSE 2013*, volume 8424 of *Lecture Notes in Computer Science*, pages 92–111. Springer, Heidelberg, March 2014.

[JSV17] Anthony Journault, François-Xavier Standaert, and Kerem Varici. Improving the security and efficiency of block ciphers based on ls-designs. *Designs, Codes and Cryptography*, 82(1):495–509, 2017.

[Kal92] B. Kaliski. The md2 message-digest algorithm. http://www.rfc-editor.org/rfc/rfc1319.txt, April 1992. RFC 1319.

[Kal00] B. Kaliski. PKCS #5: Password-based cryptography specification version 2. RFC 2898 (Informational), September 2000.

[KBC97] H. Krawczyk, M. Bellare, and R. Canetti. HMAC: Keyed-hashing for message authentication. https://tools.ietf.org/html/rfc2104, February 1997. RFC 2104.

[KDH13] Ferhat Karakoç, Hüseyin Demirci, and A. Emre Harmancı. ITUbee: A software oriented lightweight block cipher. In Gildas Avoine and

Orhun Kara, editors, *Lightweight Cryptography for Security and Privacy: LightSec 2013*, volume 8162 of *Lecture Notes in Computer Science*, pages 16–27, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[Ker83] Auguste Kerckhoffs. La cryptographie militaire. *Journal des sciences militaires*, IX:5–38, January 1883. Available online on the website of Fabien Petitcolas: http://www.petitcolas.net/kerckhoffs/crypto_militaire_1.pdf (the article was written in French).

[KG16] Pierre Karpman and Benjamin Grégoire. The LITTLUN S-box and the FLY block cipher. In *Lightweight Cryptography Workshop 2016, October 17-18 (informal proceedings)*. National Institute of Standards and Technology, 2016.

[KK06] John Kelsey and Tadayoshi Kohno. Herding hash functions and the Nostradamus attack. In Serge Vaudenay, editor, *Advances in Cryptology – EUROCRYPT 2006*, volume 4004 of *Lecture Notes in Computer Science*, pages 183–200. Springer, Heidelberg, May / June 2006.

[KK13] Oleksandr Kazymyrov and Valentyna Kazymyrova. Algebraic aspects of the russian hash standard GOST r 34.11-2012. Cryptology ePrint Archive, Report 2013/556, 2013. http://eprint.iacr.org/2013/556.

[KKO13] Oleksandr Kazymyrov, Valentyna Kazymyrova, and Roman Oliynykov. A method for generation of high-nonlinear s-boxes based on gradient descent. Cryptology ePrint Archive, Report 2013/578, 2013. http://eprint.iacr.org/2013/578.

[KKP+04] 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 Jong In Lim and Dong Hoon Lee, editors, *ICISC 03: 6th International Conference on Information Security and Cryptology*, volume 2971 of *Lecture Notes in Computer Science*, pages 432–445. Springer, Heidelberg, November 2004.

[KLL+14] Elif Bilge Kavun, Martin M. Lauridsen, Gregor Leander, Christian Rechberger, Peter Schwabe, and Tolga Yalçın. Prøst v1. Candidate for the CAESAR Competition. See also https://competitions.cr.yp.to/round1/sablierv1.pdf, 2014.

[KLPR10] Lars R. Knudsen, Gregor Leander, Axel Poschmann, and Matthew J. B. Robshaw. PRINTcipher: A block cipher for IC-printing. In Stefan Mangard and François-Xavier Standaert, editors, *Cryptographic Hardware and Embedded Systems – CHES 2010*, volume 6225 of *Lecture Notes in Computer Science*, pages 16–32. Springer, Heidelberg, August 2010.

[KMA+00] Masayuki Kanda, Shiho Moriai, Kazumaro Aoki, Hiroki Ueda, Youichi Takashima, Kazuo Ohta, and Tsutomu Matsumoto. E2 – a new 128-bit block cipher. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, E83-A(1):48–59, 12 2000.

[Knu95] Lars R. Knudsen. Truncated and higher order differentials. In Bart Preneel, editor, *Fast Software Encryption – FSE'94*, volume 1008 of *Lecture Notes in Computer Science*, pages 196–211. Springer, Heidelberg, December 1995.

[Knu98] Lars Ramkilde Knudsen. DEAL – a 128-bit block cipher, aes submission. AES Submission, 1998.

[Kol99] V.F. Kolchin. *Random Graphs*. Encyclopedia of Mathematics and its Applications. Cambridge University Press, 1999.

[KR94] Burton S. Kaliski Jr. and Matthew J. B. Robshaw. Linear cryptanalysis using multiple approximations. In Yvo Desmedt, editor, *Advances in Cryptology – CRYPTO'94*, volume 839 of *Lecture Notes in Computer Science*, pages 26–39. Springer, Heidelberg, August 1994.

[KR96] Joe Kilian and Phillip Rogaway. How to protect DES against exhaustive key search. In Neal Koblitz, editor, *Advances in Cryptology – CRYPTO'96*, volume 1109 of *Lecture Notes in Computer Science*, pages 252–267. Springer, Heidelberg, August 1996.

[KRW99] Lars R. Knudsen, Matthew J. B. Robshaw, and David Wagner. Truncated differentials and Skipjack. In Michael J. Wiener, editor, *Advances in Cryptology – CRYPTO'99*, volume 1666 of *Lecture Notes in Computer Science*, pages 165–180. Springer, Heidelberg, August 1999.

[KS05] John Kelsey and Bruce Schneier. Second preimages on n-bit hash functions for much less than 2n work. In Ronald Cramer, editor, *Advances in Cryptology – EUROCRYPT 2005*, volume 3494 of *Lecture Notes in Computer Science*, pages 474–490. Springer, Heidelberg, May 2005.

[KS07] Liam Keliher and Jiayuan Sui. Exact maximum expected differential and linear probability for 2-round Advanced Encryption Standard. *IET Information Security*, 1(2):53–57, 2007.

[KS12] Gohar M Kyureghyan and Valentin Suder. On inverses of APN exponents. In *IEEE International Symposium on Information Theory - ISIT 2012*, pages 1207–1211. IEEE, 2012.

[KSW97] John Kelsey, Bruce Schneier, and David Wagner. Related-key cryptanalysis of 3-WAY, Biham-DES, CAST, DES-X, NewDES, RC2, and TEA. In Yongfei Han, Tatsuaki Okamoto, and Sihan Qing, editors, *ICICS 97: 1st International Conference on Information and Communication Security*, volume 1334 of *Lecture Notes in Computer Science*, pages 233–246. Springer, Heidelberg, November 1997.

[KSW04] Chris Karlof, Naveen Sastry, and David Wagner. TinySec: a link layer security architecture for wireless sensor networks. In *Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 162–175. ACM, 2004.

[KW01] Lars Knudsen and David Wagner. On the structure of Skipjack. *Discrete Applied Mathematics*, 111(1–2):103 – 116, 2001. Coding and Cryptology.

[KW02] Lars R. Knudsen and David Wagner. Integral cryptanalysis. In Joan Daemen and Vincent Rijmen, editors, *Fast Software Encryption – FSE 2002*, volume 2365 of *Lecture Notes in Computer Science*, pages 112–127. Springer, Heidelberg, February 2002.

[Lan90] Philippe Langevin. Covering radius of RM (1, 9) in RM (3, 9). In Gérard D. Cohen and Pascale Charpin, editors, *International Symposium on Coding Theory and Applications - EUROCODE '90*, volume 514 of *Lecture Notes in Computer Science*, pages 51–59. Springer, 1990.

[Lea14] Gregor Leander. Lightweight block cipher design. Presentation slides available at https://summerschool-croatia.cs.ru.nl/2014/slides/Lightweight%20Block%20Cipher%20Design.pdf, 2014.

[Leu16] Gaëtan Leurent. Differential forgery attack against LAC. In Orr Dunkelman and Liam Keliher, editors, *Selected Areas in Cryptography – SAC 2015*, volume 9566 of *Lecture Notes in Computer Science*, pages 217–224, Cham, 2016. Springer International Publishing.

[LH98] Chae Hoon Lim and Hyo Sun Hwang. CRYPTON: A new 128-bit block cipher - specification and analysis. AES Submission, see also http://dasan.sejong.ac.kr/~chlim/pub/cryptonv05.ps. This paper presents the version 0.5 of Crypton (see [DPV01] for version 1.0), 1998.

[LJH$^+$07] Fen Liu, Wen Ji, Lei Hu, Jintai Ding, Shuwang Lv, Andrei Pyshkin, and Ralf-Philipp Weinmann. Analysis of the SMS4 block cipher. In Josef Pieprzyk, Hossein Ghodosi, and Ed Dawson, editors, *ACISP 07: 12th Australasian Conference on Information Security and Privacy*, volume 4586 of *Lecture Notes in Computer Science*, pages 158–170. Springer, Heidelberg, July 2007.

[LJW13] Leibo Li, Keting Jia, and Xiaoyun Wang. Improved meet-in-the-middle attacks on AES-192 and PRINCE. Cryptology ePrint Archive, Report 2013/573, 2013. http://eprint.iacr.org/2013/573.

[LK06] Chae Hoon Lim and Tymur Korkishko. mCrypton - a lightweight block cipher for security of low-cost RFID tags and sensors. In Jooseok Song, Taekyoung Kwon, and Moti Yung, editors, *WISA 05: 6th International Workshop on Information Security Applications*, volume 3786 of *Lecture Notes in Computer Science*, pages 243–258. Springer, Heidelberg, August 2006.

[LLLS14] Ruilin Li, Heng Li, Chao Li, and Bing Sun. A low data complexity attack on the GMR-2 cipher used in the satellite phones. In Shiho Moriai, editor, *Fast Software Encryption – FSE 2013*, volume 8424 of *Lecture Notes in Computer Science*, pages 485–501. Springer, Heidelberg, March 2014.

[LLY$^+$05] H.J. Lee, S.J. Lee, J.H. Yoon, D.H. Cheon, and J.I. Lee. The SEED Encryption Algorithm. RFC 4269 (Informational), December 2005.

[LM91] Xuejia Lai and James L. Massey. A proposal for a new block encryption standard. In Ivan Damgård, editor, *Advances in Cryptology – EUROCRYPT'90*, volume 473 of *Lecture Notes in Computer Science*, pages 389–404. Springer, Heidelberg, May 1991.

[LM02] Helger Lipmaa and Shiho Moriai. Efficient algorithms for computing differential properties of addition. In Mitsuru Matsui, editor, *Fast Software Encryption – FSE 2001*, volume 2355 of *Lecture Notes in Computer Science*, pages 336–350. Springer, Heidelberg, April 2002.

[LMV05] Yi Lu, Willi Meier, and Serge Vaudenay. The conditional correlation attack: A practical attack on bluetooth encryption. In Victor Shoup, editor, *Advances in Cryptology – CRYPTO 2005*, volume 3621 of *Lecture Notes in Computer Science*, pages 97–117. Springer, Heidelberg, August 2005.

[LN15] Virginie Lallemand and María Naya-Plasencia. Cryptanalysis of KLEIN. In Carlos Cid and Christian Rechberger, editors, *Fast Software Encryption – FSE 2014*, volume 8540 of *Lecture Notes in Computer Science*, pages 451–470. Springer, Heidelberg, March 2015.

[LPPS07] Gregor Leander, Christof Paar, Axel Poschmann, and Kai Schramm. New lightweight DES variants. In Alex Biryukov, editor, *Fast Software Encryption – FSE 2007*, volume 4593 of *Lecture Notes in Computer Science*, pages 196–210. Springer, Heidelberg, March 2007.

[LPTY16] Atul Luykx, Bart Preneel, Elmar Tischhauser, and Kan Yasuda. A MAC mode for lightweight block ciphers. In Thomas Peyrin, editor, *Fast Software Encryption – FSE 2016*, volume 9783 of *Lecture Notes in Computer Science*, pages 43–59. Springer, Heidelberg, March 2016.

[LR88] Michael Luby and Charles Rackoff. How to construct pseudorandom permutations from pseudorandom functions. *SIAM Journal on Computing*, 17(2):373–386, 1988.

[LRW02] Moses Liskov, Ronald L. Rivest, and David Wagner. Tweakable block ciphers. In Moti Yung, editor, *Advances in Cryptology – CRYPTO 2002*, volume 2442 of *Lecture Notes in Computer Science*, pages 31–46. Springer, Heidelberg, August 2002.

[LS15] Rodolphe Lampe and Yannick Seurin. Security analysis of key-alternating Feistel ciphers. In Carlos Cid and Christian Rechberger, editors, *Fast Software Encryption – FSE 2014*, volume 8540 of *Lecture Notes in Computer Science*, pages 243–264. Springer, Heidelberg, March 2015.

[LST$^{+}$09] Stefan Lucks, Andreas Schuler, Erik Tews, Ralf-Philipp Weinmann, and Matthias Wenzel. Attacks on the DECT authentication mechanisms. In Marc Fischlin, editor, *Topics in Cryptology – CT-RSA 2009*, volume 5473 of *Lecture Notes in Computer Science*, pages 48–65. Springer, Heidelberg, April 2009.

[Ltd06] Beijing Data Security Technology Co. Ltd. Specification of SMS4, Block Cipher for WLAN Products – SMS4 (in Chinese). Available at http://www.oscca.gov.cn/UpFile/200621016423197990.pdf, 2006.

[LV04] Yi Lu and Serge Vaudenay. Faster correlation attack on Bluetooth keystream generator E0. In Matthew Franklin, editor, *Advances in Cryptology – CRYPTO 2004*, volume 3152 of *Lecture Notes in Computer Science*, pages 407–425. Springer, Heidelberg, August 2004.

[LW14a] Yongqiang Li and Mingsheng Wang. Constructing differentially 4-uniform permutations over GF($2^{2m}$) from quadratic APN permutations over GF($2^{2m+1}$). *Designs, Codes and Cryptography*, 72(2):249–264, 2014.

[LW14b] Yongqiang Li and Mingsheng Wang. Constructing S-boxes for lightweight cryptography with Feistel structure. In Lejla Batina and Matthew Robshaw, editors, *Cryptographic Hardware and Embedded Systems – CHES 2014*, volume 8731 of *Lecture Notes in Computer Science*, pages 127–146. Springer, Heidelberg, September 2014.

[MAM17] Vasily Mikhalev, Frederik Armknecht, and Christian Müller. On ciphers that continuously access the non-volatile key. *IACR Transactions on Symmetric Cryptology*, 2016(2):52–79, 2017.

[Mas69] J. Massey. Shift-register synthesis and bch decoding. *IEEE Transactions on Information Theory*, 15(1):122–127, Jan 1969.

[Mas94] James L. Massey. SAFER K-64: A byte-oriented block-ciphering algorithm. In Ross J. Anderson, editor, *Fast Software Encryption – FSE'93*, volume 809 of *Lecture Notes in Computer Science*, pages 1–17. Springer, Heidelberg, December 1994.

[Mat94] Mitsuru Matsui. Linear cryptoanalysis method for DES cipher. In Tor Helleseth, editor, *Advances in Cryptology – EUROCRYPT'93*, volume 765 of *Lecture Notes in Computer Science*, pages 386–397. Springer, Heidelberg, May 1994.

[Mat97] Mitsuru Matsui. New block encryption algorithm MISTY. In Eli Biham, editor, *Fast Software Encryption – FSE'97*, volume 1267 of *Lecture Notes in Computer Science*, pages 54–68. Springer, Heidelberg, January 1997.

[MBTM16] Kerry A. McKay, Larry Bassham, Meltem Sönmez Turan, and Nicky Mouha. NISTIR 8114 – DRAFT report on lightweight cryptography. Available on the NIST website: http://csrc.nist.gov/publications/drafts/nistir-8114/nistir_8114_draft.pdf, 2016.

[MDFK15] Brice Minaud, Patrick Derbez, Pierre-Alain Fouque, and Pierre Karpman. Key-recovery attacks on ASASA. In Tetsu Iwata and Jung Hee Cheon, editors, *Advances in Cryptology – ASIACRYPT 2015, Part II*, volume 9453 of *Lecture Notes in Computer Science*, pages 3–27. Springer, Heidelberg, November / December 2015.

[MMH+14] Nicky Mouha, Bart Mennink, Anthony Van Herrewege, Dai Watanabe, Bart Preneel, and Ingrid Verbauwhede. Chaskey: An efficient MAC algorithm for 32-bit microcontrollers. In Antoine Joux and Amr M. Youssef, editors, *SAC 2014: 21st Annual International Workshop on Selected Areas in Cryptography*, volume 8781 of *Lecture Notes in Computer Science*, pages 306–323. Springer, Heidelberg, August 2014.

[Mou15] Nicky Mouha. The design space of lightweight cryptography. Cryptology ePrint Archive, Report 2015/303, 2015. http://eprint.iacr.org/2015/303.

[MS87a] Richard A. Mollin and Charles Small. On permutation polynomials over finite fields. *International Journal of Mathematics and Mathematical Sciences*, 10(3):535–543, 1987.

[MS87b] Judy H. Moore and Gustavus J. Simmons. Cycle structures of the DES with weak and semi-weak keys. In Andrew M. Odlyzko, editor, *Advances in Cryptology – CRYPTO'86*, volume 263 of *Lecture Notes in Computer Science*, pages 9–32. Springer, Heidelberg, August 1987.

[MY93] Mitsuru Matsui and Atsuhiro Yamagishi. A new method for known plaintext attack of FEAL cipher. In Rainer A. Rueppel, editor, *Advances in Cryptology – EUROCRYPT'92*, volume 658 of *Lecture Notes in Computer Science*, pages 81–91. Springer, Heidelberg, May 1993.

[MZ06] Ilya Mironov and Lintao Zhang. Applications of sat solvers to cryptanalysis of hash functions. In Armin Biere and CarlaP. Gomes, editors, *Theory and Applications of Satisfiability Testing - SAT 2006*, volume 4121 of *Lecture Notes in Computer Science*, pages 102–115. Springer Berlin Heidelberg, 2006.

[Nak08] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008.

[NESP08] Karsten Nohl, David Evans, Starbug Starbug, and Henryk Plötz. Reverse-engineering a cryptographic RFID tag. In *USENIX security symposium*, volume 28, 2008.

[Nik11] Ivica Nikolic. Tweaking AES. In Alex Biryukov, Guang Gong, and Douglas R. Stinson, editors, *SAC 2010: 17th Annual International Workshop on Selected Areas in Cryptography*, volume 6544 of *Lecture Notes in Computer Science*, pages 198–210. Springer, Heidelberg, August 2011.

[Nik15] Ivica Nikolić. Tiaoxin-346. Submission to the CAESAR competition, see also http://www1.spms.ntu.edu.sg/~syllab/m/index.php/Tiaoxin-346, 2015.

[NK95] Kaisa Nyberg and Lars R. Knudsen. Provable security against a differential attack. *Journal of Cryptology*, 8(1):27–37, 1995.

[NLS$^+$10] Ferguson Niels, Stefan Lucks, Bruce Schneier, Doug Whiting, Mihir Bellare, Tadayoshi Kohno, Jon Callas, and Jesse Walker. The Skein hash function family. *Submission to NIST*, (round 3), 2010.

[Nob94] Nobody. Thank you Bob Anderson. Mail to the cypherpunk mailing list from nobody@jpunix.com, available at https://web.archive.org/web/20010722163902/http://cypherpunks.venona.com/date/1994/09/msg00304.html, September 1994.

[NTW10] Karsten Nohl, Erik Tews, and Ralf-Philipp Weinmann. Cryptanalysis of the DECT standard cipher. In Seokhie Hong and Tetsu Iwata, editors, *Fast Software Encryption – FSE 2010*, volume 6147 of *Lecture Notes in Computer Science*, pages 1–18. Springer, Heidelberg, February 2010.

[NVP13] Valérie Nachef, Emmanuel Volte, and Jacques Patarin. Differential attacks on generalized Feistel schemes. In Michel Abdalla, Cristina Nita-Rotaru, and Ricardo Dahab, editors, *CANS 13: 12th International Conference on Cryptology and Network Security*, volume 8257 of *Lecture Notes in Computer Science*, pages 1–19. Springer, Heidelberg, November 2013.

[NW97] R. M. Needham and D. J. Wheeler. Tea extensions. Technical report, Cambridge University, Cambridge, UK, October 1997.

[NW06] Kaisa Nyberg and Johan Wallén. Improved linear distinguishers for SNOW 2.0. In Matthew J. B. Robshaw, editor, *Fast Software Encryption – FSE 2006*, volume 4047 of *Lecture Notes in Computer Science*, pages 144–162. Springer, Heidelberg, March 2006.

[Nyb94] Kaisa Nyberg. Differentially uniform mappings for cryptography. In Tor Helleseth, editor, *Advances in Cryptology – EUROCRYPT'93*, volume 765 of *Lecture Notes in Computer Science*, pages 55–64. Springer, Heidelberg, May 1994.

[Nyb96] Kaisa Nyberg. Generalized Feistel networks. In Kwangjo Kim and Tsutomu Matsumoto, editors, *Advances in Cryptology – ASIACRYPT'96*, volume 1163 of *Lecture Notes in Computer Science*, pages 91–104. Springer, Heidelberg, November 1996.

[O'C94] Luke O'Connor. On the distribution of characteristics in bijective mappings. In Tor Helleseth, editor, *Advances in Cryptology – EUROCRYPT'93*, volume 765 of *Lecture Notes in Computer Science*, pages 360–370. Springer, Heidelberg, May 1994.

[O'C95] Luke O'Connor. Properties of linear approximation tables. In Bart Preneel, editor, *Fast Software Encryption – FSE'94*, volume 1008 of *Lecture Notes in Computer Science*, pages 131–136. Springer, Heidelberg, December 1995.

[OC16] Colin O'Flynn and Zhizhang Chen. Power analysis attacks against IEEE 802.15.4 nodes. In *Constructive Side-Channel Analysis and Secure Design – COSADE 2016*, volume 9689 of *Lecture Notes in Computer Science*, Cham, 2016. Springer International Publishing.

[OGK$^+$15a] Roman Oliynykov, Ivan Gorbenko, Oleksandr Kazymyrov, Victor Ruzhentsev, Oleksandr Kuznetsov, Yurii Gorbenko, Artem Boiko, Oleksandr Dyrda, Viktor Dolgov, and Andrii Pushkaryov. A new standard of ukraine: The kupyna hash function. Cryptology ePrint Archive, Report 2015/885, 2015. http://eprint.iacr.org/2015/885.

[OGK$^+$15b] Roman Oliynykov, Ivan Gorbenko, Oleksandr Kazymyrov, Victor Ruzhentsev, Oleksandr Kuznetsov, Yurii Gorbenko, Oleksandr Dyrda, Viktor Dolgov, Andrii Pushkaryov, Ruslan Mordvinov, and Dmytro Kaidalov. A new encryption standard of Ukraine: The Kalyna block cipher. Cryptology ePrint Archive, Report 2015/650, 2015. http://eprint.iacr.org/2015/650.

[OMSK01] Kenji Ohkuma, Hirofumi Muratani, Fumihiko Sano, and Shin-ichi Kawamura. The block cipher Hierocrypt. In Douglas R. Stinson and Stafford E. Tavares, editors, *SAC 2000: 7th Annual International Workshop on Selected Areas in Cryptography*, volume 2012 of *Lecture Notes in Computer Science*, pages 72–88. Springer, Heidelberg, August 2001.

[Pat08] Jacques Patarin. Generic attacks on Feistel schemes. Cryptology ePrint Archive, Report 2008/036, 2008. http://eprint.iacr.org/2008/036.

[Per09] Colin Percival. Stronger key derivation via sequential memory-hard functions. *Self-published*, pages 1–16, 2009.

[PG97] Jacques Patarin and Louis Goubin. Asymmetric cryptography with S-boxes. In Yongfei Han, Tatsuaki Okamoto, and Sihan Qing, editors, *ICICS 97: 1st International Conference on Information and Communication Security*, volume 1334 of *Lecture Notes in Computer Science*, pages 369–380. Springer, Heidelberg, November 1997.

[PK15] Léo Perrin and Dmitry Khovratovich. Collision spectrum, entropy loss, T-sponges, and cryptanalysis of GLUON-64. In Carlos Cid and Christian Rechberger, editors, *Fast Software Encryption – FSE 2014*, volume 8540 of *Lecture Notes in Computer Science*, pages 82–103. Springer, Heidelberg, March 2015.

[PLW10] Axel Poschmann, San Ling, and Huaxiong Wang. 256 bit standardized crypto for 650 GE - GOST revisited. In Stefan Mangard and François-Xavier Standaert, editors, *Cryptographic Hardware and Embedded Systems – CHES 2010*, volume 6225 of *Lecture Notes in Computer Science*, pages 219–233. Springer, Heidelberg, August 2010.

[PMA07] Lea Troels Møller Pedersen, Carsten Valdemar Munk, and Lisbet Møller Andersen. Cryptography – the rise and fall of DVD encryption. Available online at http://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=3672D97255B2446765DA47DA97960CDF?doi=10.1.1.118.6103&rep=rep1&type=pdf, 2007.

[PRC12] Gilles Piret, Thomas Roche, and Claude Carlet. PICARO - a block cipher allowing efficient higher-order side-channel resistance. In Feng Bao, Pierangela Samarati, and Jianying Zhou, editors, *ACNS 12: 10th*

*International Conference on Applied Cryptography and Network Security*, volume 7341 of *Lecture Notes in Computer Science*, pages 311–328. Springer, Heidelberg, June 2012.

[PU16] Léo Perrin and Aleksei Udovenko. Algebraic insights into the secret feistel network. In Thomas Peyrin, editor, *Fast Software Encryption – FSE 2016*, volume 9783 of *Lecture Notes in Computer Science*, pages 378–398. Springer, Heidelberg, March 2016.

[PU17] Léo Perrin and Aleksei Udovenko. Exponential S-boxes: a link between the S-boxes of BelT and Kuznyechik/Streebog. *IACR Transactions on Symmetric Cryptology*, 2016(2):99–124, 2017.

[PUB16] Léo Perrin, Aleksei Udovenko, and Alex Biryukov. Cryptanalysis of a theorem: Decomposing the only known solution to the big APN problem. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology – CRYPTO 2016, Part II*, volume 9815 of *Lecture Notes in Computer Science*, pages 93–122. Springer, Heidelberg, August 2016.

[RBF08] Vincent Rijmen, Paulo S.L.M. Barreto, and Décio L. Gazzoni Filho. Rotation symmetry in algebraically generated cryptographic substitution tables. *Information Processing Letters*, 106(6):246 – 250, 2008.

[RDP$^+$96] Vincent Rijmen, Joan Daemen, Bart Preneel, Anton Bossalaers, and Erik De Win. The cipher SHARK. In Dieter Gollmann, editor, *Fast Software Encryption – FSE'96*, volume 1039 of *Lecture Notes in Computer Science*, pages 99–111. Springer, Heidelberg, February 1996.

[RH03] Gregory G. Rose and Philip Hawkes. Turing: A fast stream cipher. In Thomas Johansson, editor, *Fast Software Encryption – FSE 2003*, volume 2887 of *Lecture Notes in Computer Science*, pages 290–306. Springer, Heidelberg, February 2003.

[Riv95] Ronald L. Rivest. The RC5 encryption algorithm. In Bart Preneel, editor, *Fast Software Encryption – FSE'94*, volume 1008 of *Lecture Notes in Computer Science*, pages 86–96. Springer, Heidelberg, December 1995.

[Rob16] Matt Robshaw. Lightweight cryptography and RAIN RFID. Invited presentation at the NIST Workshop on Lightweight Cryptography 2016. Slides available at https://www.nist.gov/sites/default/files/documents/2016/10/17/robshaw-presentation-lwc2016.pdf, 2016.

[Röc08] Andrea Röck. Stream ciphers using a random update function: Study of the entropy of the inner state. In Serge Vaudenay, editor, *AFRICACRYPT 08: 1st International Conference on Cryptology in Africa*, volume 5023 of *Lecture Notes in Computer Science*, pages 258–275. Springer, Heidelberg, June 2008.

[Rog15] Phillip Rogaway. The moral character of cryptographic work. Cryptology ePrint Archive, Report 2015/1162, 2015. http://eprint.iacr.org/2015/1162.

[ROSW16] Eyal Ronen, Colin O'Flynn, Adi Shamir, and Achi-Or Weingarten. IoT goes nuclear: Creating a Zigbee chain reaction. Cryptology ePrint Archive, Report 2016/1047, 2016. http://eprint.iacr.org/2016/1047.

[RR16a] Shahram Rasoolzadeh and Håvard Raddum. Cryptanalysis of 6-round PRINCE using 2 known plaintexts. Cryptology ePrint Archive, Report 2016/132, 2016. http://eprint.iacr.org/2016/132.

[RR16b] Shahram Rasoolzadeh and Håvard Raddum. Cryptanalysis of PRINCE with minimal data. In David Pointcheval, Abderrahmane Nitaj, and Tajjeeddine Rachidi, editors, *Progress in Cryptology – AFRICACRYPT 2016*, volume 9646 of *Lecture Notes in Computer Science*, pages 109–126. Springer International Publishing, 2016.

[RSA78] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, February 1978.

[RSW96] R. L. Rivest, A. Shamir, and D. A. Wagner. Time-lock puzzles and timed-release crypto. Technical report, Massachusetts Institute of Technology, Cambridge, MA, USA, 1996.

[Rud15] V. Rudskoy. Note on streebog constants origin. Technical report, Federal Agency on Technical Regulation and Metrology (GOST), 2014-2015.

[SB15] Markku-Juhani O. Saarinen and Billy Bob Brumley. Whirlbob, the whirlpool based variant of STRIBOB. In Sonja Buchegger and Mads Dam, editors, *Secure IT Systems, 20th Nordic Conference, NordSec*, volume 9417 of *Lecture Notes in Computer Science*, pages 106–122. Springer, 2015.

[SBY$^+$15] Hadi Soleimany, Céline Blondeau, Xiaoli Yu, Wenling Wu, Kaisa Nyberg, Huiling Zhang, Lei Zhang, and Yanfeng Wang. Reflection cryptanalysis of PRINCE-like ciphers. *Journal of Cryptology*, 28(3):718–744, July 2015.

[Sch94] Bruce Schneier. Description of a new variable-length key, 64-bit block cipher (Blowfish). In Ross J. Anderson, editor, *Fast Software Encryption – FSE'93*, volume 809 of *Lecture Notes in Computer Science*, pages 191–204. Springer, Heidelberg, December 1994.

[Sch95] Bruce Schneier. the S-1 Algorithm. mail to the cypherpunk mailing list, https://groups.google.com/forum/#!msg/sci.crypt/l4nsjVu27lU/LIycdmVxJZ0J, September 1995.

[Sch14] Felix Schuster. Reverse engineering of chiasmus from gstool. Presentation at the HGI-Kolloquium. Slides available at https://prezi.com/ehrz4krw2z0d/hgi-chm/, January 2014.

[Sco85] Robert Scott. Wide-open encryption design offers flexible implementations. *Cryptologia*, 9(1):75–91, 1985.

[SDL+14] Vasily Shishkin, Denis Dygin, Ivan Lavrikov, Grigory Marshalko, Vladimir Rudskoy, and Dmitry Trifonov. Low-weight and hi-end: Draft russian encryption standard. https://mjos.fi/doc/rus/gh_ctcrypt.pdf, 2014. Invited presentation at CTCrypt'14. The slides are available at https://mjos.fi/doc/rus/gh_ctcrypt.pdf.

[Shi13] Vasilij Shishkin. Принципы синтеза перспективного алгоритма блочного шифрования с длиной блока 128 бит [the principles of synthesis of a promising block cipher algorithm with a block length of 128 bits], 2013. Available online at http://www.ruscrypto.ru/resource/summary/rc2013/ruscrypto_2013_066.zip.

[SIH+11] Kyoji Shibutani, Takanori Isobe, Harunaga Hiwatari, Atsushi Mitsuda, Toru Akishita, and Taizo Shirai. Piccolo: An ultra-lightweight blockcipher. In Bart Preneel and Tsuyoshi Takagi, editors, *Cryptographic Hardware and Embedded Systems – CHES 2011*, volume 6917 of *Lecture Notes in Computer Science*, pages 342–357. Springer, Heidelberg, September / October 2011.

[SKW+98] Bruce Schneier, John Kelsey, Doug Whiting, David Wagner, Chris Hall, and Niels Ferguson. Twofish: A 128-bit block cipher. AES Submission, see also https://www.schneier.com/academic/paperfiles/paper-twofish-paper.pdf, 1998.

[SM10] Tomoyasu Suzaki and Kazuhiko Minematsu. Improving the generalized Feistel. In Seokhie Hong and Tetsu Iwata, editors, *Fast Software Encryption – FSE 2010*, volume 6147 of *Lecture Notes in Computer Science*, pages 19–39. Springer, Heidelberg, February 2010.

[SMMK13] Tomoyasu Suzaki, Kazuhiko Minematsu, Sumio Morioka, and Eita Kobayashi. TWINE : A lightweight block cipher for multiple platforms. In Lars R. Knudsen and Huapeng Wu, editors, *SAC 2012: 19th Annual International Workshop on Selected Areas in Cryptography*, volume 7707 of *Lecture Notes in Computer Science*, pages 339–354. Springer, Heidelberg, August 2013.

[SPGQ06] François-Xavier Standaert, Gilles Piret, Neil Gershenfeld, and Jean-Jacques Quisquater. Sea: A scalable encryption algorithm for small embedded applications. In Josep Domingo-Ferrer, Joachim Posegga, and Daniel Schreckling, editors, *Smart Card Research and Advanced Applications: CARDIS 2006*, volume 3928 of *Lecture Notes in Computer Science*, pages 222–236, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[SPR+04] François-Xavier Standaert, Gilles Piret, Gaël Rouvroy, Jean-Jacques Quisquater, and Jean-Didier Legat. ICEBERG: An involutional cipher efficient for block encryption in reconfigurable hardware. In Bimal K. Roy and Willi Meier, editors, *Fast Software Encryption – FSE 2004*, volume 3017 of *Lecture Notes in Computer Science*, pages 279–299. Springer, Heidelberg, February 2004.

[SS16] Peter Schwabe and Ko Stoffelen. All the AES you need on cortex-M3 and M4. In *Selected Areas in Cryptography – SAC 2016*, volume To appear of *Lecture Notes in Computer Science*, 2016.

[SSA$^+$07] Taizo Shirai, Kyoji Shibutani, Toru Akishita, Shiho Moriai, and Tetsu Iwata. The 128-bit blockcipher CLEFIA (extended abstract). In Alex Biryukov, editor, *Fast Software Encryption – FSE 2007*, volume 4593 of *Lecture Notes in Computer Science*, pages 181–195. Springer, Heidelberg, March 2007.

[STW13] Jan Schejbal, Erik Tews, and Julian Wälde. Reverse engineering of chiasmus from gstool. Presentation at the Chaos Computer Club (CCC). Slides available at https://events.ccc.de/congress/2013/Fahrplan/system/attachments/2197/original/gstool_slides.pdf, 2013.

[SV00] Jacques Stern and Serge Vaudenay. CS-Cipher. First Open NESSIE Workshop, see also https://www.cosic.esat.kuleuven.be/nessie/workshop/submissions/cs-cipher.zip, 2000.

[SYY$^+$02] Takeshi Shimoyama, Hitoshi Yanami, Kazuhiro Yokoyama, Masahiko Takenaka, Kouichi Itoh, Jun Yajima, Naoya Torii, and Hidema Tanaka. The block cipher SC2000. In Mitsuru Matsui, editor, *Fast Software Encryption – FSE 2001*, volume 2355 of *Lecture Notes in Computer Science*, pages 312–327. Springer, Heidelberg, April 2002.

[TBA$^+$13] Sui-Guan Teo, Harry Bartlett, Ali Alhamdan, Leonie Simpson, Kenneth Koon-Ho Wong, and Ed Dawson. State convergence in bit-based stream ciphers. Cryptology ePrint Archive, Report 2013/096, 2013. http://eprint.iacr.org/2013/096.

[TCG92] Anne Tardy-Corfdir and Henri Gilbert. A known plaintext attack of FEAL-4 and FEAL-6. In Joan Feigenbaum, editor, *Advances in Cryptology – CRYPTO'91*, volume 576 of *Lecture Notes in Computer Science*, pages 172–181. Springer, Heidelberg, August 1992.

[Tie16] Tyge Tiessen. Polytopic cryptanalysis. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology – EUROCRYPT 2016, Part I*, volume 9665 of *Lecture Notes in Computer Science*, pages 214–239. Springer, Heidelberg, May 2016.

[Tod15a] Yosuke Todo. Integral cryptanalysis on full MISTY1. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *Advances in Cryptology – CRYPTO 2015, Part I*, volume 9215 of *Lecture Notes in Computer Science*, pages 413–432. Springer, Heidelberg, August 2015.

[Tod15b] Yosuke Todo. Structural evaluation by generalized integral property. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology – EUROCRYPT 2015, Part I*, volume 9056 of *Lecture Notes in Computer Science*, pages 287–314. Springer, Heidelberg, April 2015.

[UDCI$^+$11] Markus Ullrich, Christophe De Canniere, Sebastiaan Indesteege, Özgül Küçük, Nicky Mouha, and Bart Preneel. Finding optimal bitsliced implementations of 4× 4-bit s-boxes. In *SKEW 2011, Symmetric Key Encryption Workshop*, 2011.

[UMHA16] Rei Ueno, Sumio Morioka, Naofumi Homma, and Takafumi Aoki. A high throughput/gate AES hardware architecture by compressing encryption and decryption datapaths - toward efficient CBC-mode implementation. In Benedikt Gierlichs and Axel Y. Poschmann, editors, *Cryptographic Hardware and Embedded Systems – CHES 2016*, volume 9813 of *Lecture Notes in Computer Science*, pages 538–558. Springer, Heidelberg, August 2016.

[U.S98] U.S. Department Of Commerce/National Institute of Standards and Technology. Skipjack and KEA algorithms specifications, v2.0, 1998. http://csrc.nist.gov/groups/ST/toolkit/documents/skipjack/skipjack.pdf.

[U.S99] U.S. Department Of Commerce/National Institute of Standards and Technology. Data Encryption Standard. *Federal Information Processing Standards Publication (FIPS)*, 1999. Available on the NIST website: http://csrc.nist.gov/publications/fips/fips46-3/fips46-3.pdf.

[U.S15a] U.S. Department Of Commerce/National Institute of Standards and Technology. Secure Hash Standard (SHS). *Federal Information Processing Standards Publication (FIPS)*, 2015. Available on the NIST website: http://csrc.nist.gov/publications/fips/fips180-4/fips-180-4.pdf.

[U.S15b] U.S. Department Of Commerce/National Institute of Standards and Technology. SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions. *Federal Information Processing Standards Publication (FIPS)*, 2015. Available on the NIST website: http://ws680.nist.gov/publication/get_pdf.cfm?pub_id=919061.

[Vau98] Serge Vaudenay. Provable security for block ciphers by decorrelation. In Michel Morvan, Christoph Meinel, and Daniel Krob, editors, *STACS 98: 15th Annual Symposium on Theoretical Aspects of Computer Science*, volume 1373 of *Lecture Notes in Computer Science*, pages 249–275, Berlin, Heidelberg, 1998. Springer Berlin Heidelberg.

[VGB12] Roel Verdult, Flavio D. Garcia, and Josep Balasch. Gone in 360 seconds: Hijacking with hitag2. In *Proceedings of the 21st USENIX Conference on Security Symposium*, Security'12, pages 37–37, Berkeley, CA, USA, 2012. USENIX Association.

[VGE13] Roel Verdult, Flavio D Garcia, and Baris Ege. Dismantling Megamos crypto: Wirelessly lockpicking a vehicle immobilizer. In *Supplement to the 22nd USENIX Security Symposium (USENIX Security 13)*, pages 703–718. USENIX Association, August 2013.

[VJ04] Serge Vaudenay and Pascal Junod. Device and method for encrypting and decrypting a block of data. United States Patent (20040247117), see also "Fox, a New Family of Block Ciphers" http://crypto.junod.info/sac04a.pdf, 2004.

[Wag95] David A. Wagner. Cryptanalysis of S-1. sci.crypt (Usenet), https://groups.google.com/forum/#!topic/sci.crypt.research/5N0zarU4XPs, August 1995.

[Wag99] David Wagner. The boomerang attack. In Lars R. Knudsen, editor, *Fast Software Encryption – FSE'99*, volume 1636 of *Lecture Notes in Computer Science*, pages 156–170. Springer, Heidelberg, March 1999.

[Wal03a] Johan Wallén. Linear approximations of addition modulo $2^n$. In Thomas Johansson, editor, *Fast Software Encryption – FSE 2003*, volume 2887 of *Lecture Notes in Computer Science*, pages 261–273. Springer, Heidelberg, February 2003.

[Wal03b] Johan Wallén. On the Differential and Linear Properties of Addition. Master's thesis, Helsinki University of Technology, 2003.

[WFY$^+$02] Dai Watanabe, Soichi Furuya, Hirotaka Yoshida, Kazuo Takaragi, and Bart Preneel. A new keystream generator MUGI. In Joan Daemen and Vincent Rijmen, editors, *Fast Software Encryption – FSE 2002*, volume 2365 of *Lecture Notes in Computer Science*, pages 179–194. Springer, Heidelberg, February 2002.

[WIK$^+$08] Dai Watanabe, Kota Ideguchi, Jun Kitahar, Kenichiro Muto, Hiroki Furuichi, and Toshinobu Kaneko. Enocoro-80: A hardware oriented stream cipher. In *The Third International Conference on Availability, Reliability and Security — ARES 08*, pages 1294–1300, 2008.

[WSD$^+$99] David Wagner, Leone Simpson, Ed Dawson, John Kelsey, William Millan, and Bruce Schneier. Cryptanalysis of ORYX. In Stafford E. Tavares and Henk Meijer, editors, *SAC 1998: 5th Annual International Workshop on Selected Areas in Cryptography*, volume 1556 of *Lecture Notes in Computer Science*, pages 296–305. Springer, Heidelberg, August 1999.

[WSK97] David Wagner, Bruce Schneier, and John Kelsey. Cryptoanalysis of the cellular encryption algorithm. In Burton S. Kaliski Jr., editor, *Advances in Cryptology – CRYPTO'97*, volume 1294 of *Lecture Notes in Computer Science*, pages 526–537. Springer, Heidelberg, August 1997.

[Wu16] Hongjun Wu. ACORN: A lighweight authenticated cipher (v3). Candidate for the CAESAR Competition. See also https://competitions.cr.yp.to/round3/acornv3.pdf, 2016.

[WW05] Ralf-Philipp Weinmann and Kai Wirt. Analysis of the DVB Common Scrambling Algorithm. In David Chadwick and Bart Preneel, editors, *Communications and Multimedia Security: 8th IFIP TC-6 TC-11 Conference on Communications and Multimedia Security, Sept. 15–18, 2004, Windermere, The Lake District, United Kingdom*, volume 175 of *IFIP – The International Federation for Information Processing*, Boston, MA, 2005. Springer US.

[WZ11] Wenling Wu and Lei Zhang. LBlock: A lightweight block cipher. In Javier Lopez and Gene Tsudik, editors, *ACNS 11: 9th International Conference on Applied Cryptography and Network Security*, volume 6715 of *Lecture Notes in Computer Science*, pages 327–344. Springer, Heidelberg, June 2011.

[YKPH11] Huihui Yap, Khoongming Khoo, Axel Poschmann, and Matt Henricksen. EPCBC - a block cipher suitable for electronic product code encryption. In Dongdai Lin, Gene Tsudik, and Xiaoyun Wang, editors, *CANS 11: 10th International Conference on Cryptology and Network Security*, volume 7092 of *Lecture Notes in Computer Science*, pages 76–97. Springer, Heidelberg, December 2011.

[Yuv97] Gideon Yuval. Reinventing the Travois: Encryption/MAC in 30 ROM bytes. In Eli Biham, editor, *Fast Software Encryption – FSE'97*, volume 1267 of *Lecture Notes in Computer Science*, pages 205–209. Springer, Heidelberg, January 1997.

[YWH+14] Dingfeng Ye, Peng Wang, Lei Hu, Liping Wang, Yonghong Xie, Siwei Sun, and Ping Wang. Panda v1. Candidate for the CAESAR Competition. See also https://competitions.cr.yp.to/round1/pandav1.pdf, 2014.

[YZS+15] Gangqiang Yang, Bo Zhu, Valentin Suder, Mark D. Aagaard, and Guang Gong. The simeck family of lightweight block ciphers. In Tim Güneysu and Helena Handschuh, editors, *Cryptographic Hardware and Embedded Systems – CHES 2015*, volume 9293 of *Lecture Notes in Computer Science*, pages 307–329. Springer, Heidelberg, September 2015.

[ZBL+15] Wentao Zhang, Zhenzhen Bao, Dongdai Lin, Vincent Rijmen, Bohan Yang, and Ingrid Verbauwhede. RECTANGLE: A bit-slice lightweight block cipher suitable for multiple platforms. *Science China Information Sciences*, 58(12):1–15, 2015.

[ZJ14] Xuexin Zheng and Keting Jia. Impossible differential attack on reduced-round TWINE. In Hyang-Sook Lee and Dong-Guk Han, editors, *ICISC 13: 16th International Conference on Information Security and Cryptology*, volume 8565 of *Lecture Notes in Computer Science*, pages 123–143. Springer, Heidelberg, November 2014.

[ZSX+14] Bin Zhang, Zhenqing Shi, Chao Xu, Yuan Yao, and Zhenqi Li. Sablier v1. Candidate for the CAESAR Competition. See also https://competitions.cr.yp.to/round1/sablierv1.pdf, 2014.

[ZWW+14] Lei Zhang, Wenling Wu, Yanfeng Wang, Shengbao Wu, and Jian Zhang. LAC: A lightweight authenticated encryption cipher. Candidate for the CAESAR Competition, 2014.