Hardware-Oriented SPN Block Ciphers

Fault Injection Countermeasures and Low-Latency Designs

Shahram Rasoolzadeh

Hardware-Oriented SPN Block Ciphers

Fault Injection Countermeasures and Low-Latency Designs

Dissertation Thesis

Shahram Rasoolzadeh

November 2, 2020

Submitted as the requirement for the degree of *Doctor-Engineer*
to the Faculty of Electrical Engineering and Information Technology
at Ruhr-University Bochum

Vorgelegt zur Erlangung des Grades eines *Doktor-Ingenieurs*
an der Fakultät für Elektrotechnik und Informationstechnik
der Ruhr-Universität Bochum

Supervisors:
Prof. Dr. Gregor Leander
Priv.-Doz. Dr. Amir Moradi

Disputation Date:
December 3, 2020

IMPRINT

Hardware-Oriented SPN Block Ciphers
Fault Injection Countermeasures and Low-Latency Designs

Published by the Ruhr-Universität Bochum, Bochum, Germany.

COLOPHON

This thesis was typeset using LaTeX and the `memoir` documentclass. It is based on Aaron Turon's thesis *Understanding and expressing scalable concurrency*[1], itself a mixture of `classicthesis`[2] by André Miede and `tufte-latex`[3], based on Edward Tufte's *Beautiful Evidence*.

The bibliography was processed by Biblatex. All graphics and plots are made with PGF/Ti*k*Z.

The body text is set 10/14pt (long primer) on a 26pc measure. The margin text is set 8/9pt (brevier) on a 12pc measure. Matthew Carter's Charter acts as both the text and display typeface. Monospaced text uses Jim Lyles's `Bitstream Vera Mono` ("Bera Mono").

[1] https://people.mpi-sws.org/~turon/turon-thesis.pdf

[2] https://bitbucket.org/amiede/classicthesis/

[3] https://github.com/Tufte-LaTeX/tufte-latex

Doing research is challenging as well as attractive.
It is like being lost in a jungle and trying to use all the knowledge
that you can gather to come up with some new tricks,
and with some luck you might find a way out.

Maryam Mirzakhani, Field's Medal Winner, August 2014

Abstract

Block ciphers are fundamental building blocks of modern cryptography. They are not only used for encryption but are also the basic components in the construction of other cryptographic mechanisms. Even though the cryptographic security of block ciphers is still the main parameter in their design and application, the efficiency of their implementations became more interesting with the rise of lightweight cryptography. Of equal importance, by the widespread increase of the number of IoT devices in the last decade, embedding cryptographic devices with some countermeasures against physical attacks became a necessity to ensure the physical security of these devices. The research in this thesis is related to two main topics in the direction of the design and the security of hardware-oriented SPN block ciphers: low-latency designs and countermeasures against fault analysis attacks.

Chapter 3 deals with countermeasures against fault analysis attacks. It presents a comprehensive methodology for implementing code-based concurrent error-detection and error-correction schemes. Besides, the tweakable block cipher CRAFT is presented. Here the efficient protection of its implementation against fault analysis has been the main criterion during design.

Chapter 4 is focused on low-latency designs. The low-latency block cipher PRINCEv2 is presented. By changing the key schedule of its predecessor PRINCE, the new design provides more security with (almost) no overhead on its implementation. Moreover, a new approach is presented for building low-latency Boolean functions and S-boxes.

Chapter 5 studies the Expected Differential Probability (EDP) of (truncated) differentials and the Expected Linear Potential (ELP) of (multidimensional) linear hulls in SPN block ciphers. It is shown that the previous main method for approximating the EDP of a truncated differential may produce estimations very different from the correct value. New methods are introduced that are based only on the independent round-keys assumption, allow to practically compute the EDP of (truncated) differentials and the ELP of (multidimensional) linear hulls.

Finally, we provide the security analysis for the block ciphers CRAFT and PRINCEv2 against several attacks including differential, linear, impossible differential, zero-correlation, meet-in-the-middle, time-data-memory trade-off, integral, division property, and invariant attacks in Chapter 6.

Zusammenfassung

Blockchiffren sind fundamentale Bausteine der modernen Kryptographie. Sie werden nicht nur zur Verschlüsselung eingesetzt, sondern auch als grundlegende Komponenten verschiedener kryptographischer Mechanismen. Obwohl die kryptographische Sicherheit von Blockchiffren nach wie vor der wichtigste Parameter für ihr Design und ihre Anwendung ist, erlangte mit dem Aufkommen der Lightweight-Kryptographie auch die Effizienz der Implementierungen zunehmende Bedeutung. Von großer Bedeutung ist es außerdem, die kryptographischen Implementierungen in den weiterverbeiteten Kleinstgeräten des Internet of Things (IoT) gegen physikalische Angriffe abzusichern. Diese Doktorarbeit beschäftigt sich mit zwei Hauptaspekten des Designs und der Sicherheit hardwarebasierter SPN-Blockchiffren: Designs mit geringer Latenz und Gegenmaßnahmen gegen Fehlerinjektionsangriffe.

Kapitel 3 behandelt die Gegenmaßnahmen gegen Fehlerinjektionsangriffe. Es wird eine umfassende Methodik präsentiert um codebasierte nebenläufige Fehlerdetektionsverfahren und Fehlerkorrekturverfahren zu implementieren. Des Weiteren wird die Blockchiffre CRAFT präsentiert, welche die Eigenschaft erfüllt "tweakable" zu sein. Ein Hauptkriterium ihres Designs war der effiziente Schutz ihrer Implementierung gegen Fehlerinjektionsangriffe.

Kapitel 4 konzentriert sich auf Designs mit geringer Latenz. Zunächst wird die Blockchiffre PRINCEv2 wird präsentiert, welche eine sehr geringe Latenz bietet. Durch eine Veränderung der Schlüsselableitungsfunktion ihres Vorgängers PRINCE bietet das neue Design eine höhere Sicherheit (fast) ohne dabei die Implementierungskosten zu erhöhen. Des Weiteren wird ein neuer Ansatz präsentiert um Boolesche Funktionen und S-Boxen mit geringer Latenz zu konstruieren.

Kapitel 5 untersucht die Erwartete Differenzwahrscheinlichkeit (EDP) von (abgeschnittenen) Differentialen und die Erwartetes lineares Potential (ELP) von (mehrdimensionalen) linearen Hüllen in SPN-Blockchiffren. Es wird gezeigt, dass die bisherige Hauptmethode zur Approximation der EDP eines abgeschnittenen Differentials zu Ergebnissen führen kann, die sich stark vom korrekten Wert unterscheiden. Weiterhin werden neue Methoden vorgestellt, die nur auf der Annahme unabhängiger Rundenschlüssel basieren und es erlauben, die EDP von (abgeschnittenen) Differentialen und die ELP von (mehrdimensionalen) linearen Hüllen praktisch zu berechnen.

Abschließend stellen wir in Kapitel 6 die Sicherheitsanalyse der Blockchiffren CRAFT und PRINCEv2 gegen verschiedene Angriffe vor, einschließlich differentieller, linearer, unmöglich-differentieller, Null-Korrelations-, Meet-in-the-Middle, Time-Data-Memory Trade-off, Integral-, Divisionseigenschafts- und invariante Angriffe.

Acknowledgments

I believe that no one achieves success alone and this success is not complete without acknowledging the help of others.

First of all, I want to appreciate Gregor and Amir for giving me the opportunity to be their student and to work in their group. Their great supervision, support, and advice helped me a lot in my research during the last four years. I thank them for their patience for my (dummy) questions and mistakes, for giving me plenty of time to hear my (trivial) ideas and for discussions, and for leading me to the correct direction. If I am ever going to be a supervisor in the future, they taught me how to be a good one.

I would also like to express my gratitude to Irmgard and Marion for helping out in every possible situation and for lots of administrative tasks.

I also want to appreciate Christof, Friedrich, Steffen and Thorben. I believe that I could not succeed in my Ph.D. studies without your helps. Thank you for all the times you gave me for research and non-research discussions, for helping me to find out solutions to my academic and non-academic problems, and especially for proof-reading and their comments on parts of this thesis. I will always be in your debt.

Further, I am very thankful to all of my co-authors with whom I collaborated during my Ph.D. studies. I would especially like to thank Anita Aghaei, Maria Eichlseder, Virginie Lallemand, Baptiste Lambin, Aein Rezaei Shahmirzadi, Falk Schellenberg, Tobias Schneider, and Yosuke Todo for all the work and researches we have done together.

I had the opportunity to be a member of two great groups of Horst Görtz Institute; EmSec and SymCrypt. I like to thank all HGI members, especially my colleagues from these groups and the colleagues from Wasserstraße 221 building for all the great moments that we had together.

Last but foremost, I like to thank Ayda for her understanding, support and endless love, and whole my family, especially my parents and beloved sister, Aziz, Farideh and Habibeh, for supporting me at every moment of my life and in everything I decided to do. I am sure my words cannot express all my gratitude to you.

Shahram Rasoolzadeh
Bochum
December 2020

Contents

List of Figures

List of Tables

List of Algorithms

Part I

PROLOGUE

1

Introduction

"It began long ago, in a land far away . . . ."
—Bilbo Baggins in "The Hobbit" By J. R. R. Tolkien

- CRYPTOGRAPHY is used for centuries to protect secret information. Until the early 20th century, encryption methods were using pen and paper, or perhaps some simple mechanical aids, which are usually called *classic cryptography*. The invention of complex mechanical and electromechanical machines provided more complicated encryption methods, such as the Enigma. While the introduction of electronics and computers has allowed sophisticated techniques of still greater complexity, most of which are entirely unsuited to pen and paper.

- MODERN CRYPTOGRAPHY has started by Shannon's paper entitled "*A Mathematical Theory of Cryptography*" in 1945 [Sha45]. It is a common belief that this paper was the starting point for the transition of cryptography from an art to a science. Later in 1971, Feistel introduced the block cipher LUCIFER [Fei71], a direct precursor to the Data Encryption Standard (DES). DES is the first publicly accessible cipher which was standardized by the National Bureau of Standards (NBS) (a US government organization)[1] for commercial applications [FIP77].

[Sha45] Shannon, *A Mathematical Theory of Cryptography*

[Fei71] Feistel, *Block Cipher Cryptographic System*

[1] Its new name is the National Institute of Standards and Technology (NIST).

[FIP77] FIPS, "46: Data Encryption Standard"

- BLOCK CIPHERS are fundamental building blocks of modern cryptography. They are not only used for encryption but are also the primary components in the construction of other cryptographic mechanisms. Even though the DES is the most applied block cipher in the history of cryptography, due to the small key size and the possibility of brute force attacks, it was finally replaced by the block cipher RIJNDAEL [DR02] as the Advanced Encryption Standard (AES) after a public competition organized by the NIST [FIP01]. After about two decades of cryptanalysis of the AES, we believe it is today's best understood design and, importantly, a secure block cipher.

[DR02] Daemen and Rijmen, *The Design of Rijndael: AES - The Advanced Encryption Standard*

[FIP01] FIPS, "197: Advanced encryption standard (AES)"

- LIGHTWEIGHT CRYPTOGRAPHY. Even though the cryptographic security of block ciphers is still the main parameter in their design and application, and the block cipher RIJNDAEL was chosen as the successor of the AES competition due to its implementation efficiency comparing to the other candidates, the efficiency of the implementations became more and more interesting with the rise of the Internet-of-Things (IoT). The IoT is becoming more prevalent in our daily lives as it connects various kinds of everyday

objects via networks and transfers (personal) data between them. Small embedded devices in IoT have a vital role in these systems, highlighting their security significance. Hence, their security is a non-deniable requirement in relevant services like secure communication or storage of private and sensitive data. On the other hand, usually, it is considered that the AES is not a suitable design for such resource-constraint devices.

In symmetric cryptography, motivated by new application scenarios, *lightweight* cryptography has been a very active research area in the last decade. While there is no strict definition of lightweight cryptography, it is usually understood as cryptography with a strong focus on efficiency, while the efficiency can be measured according to various criteria and their combination.

- PHYSICAL ATTACKS started with the introduction of fault injection attacks in 1997. Since then, it is a common belief that an adversary with access to the cryptographic device can recover the secret information in the device, even though the underlying cryptographic primitive is mathematically secure. Thus, not only the underlying cryptographic scheme has to fulfill mathematical security requirements, the device that implements the scheme should also be physically secure. With the widespread increase in the number of IoT devices in the last decade, embedding cryptographic devices with some countermeasures against physical attacks became necessary to ensure the physical security of these devices. Therefore, integrating countermeasures to prevent such physical attacks is mandatory for widespread products that deal with security and privacy concepts. However, those countermeasures usually come at a significant cost.

1.1 MOTIVATION

The research in this thesis is related to two main topics in the direction of the design and the security of hardware-oriented SPN block ciphers: countermeasures against fault analysis attacks, and low-latency designs. In the following, we briefly explain the topics of the research in this thesis.

FAULT ANALYSIS introduced serious threats to cryptographic hardware implementations. With the improvement of the tools for injecting faults over the last decade, this attack becomes more powerful. An adversary can easily apply fault attacks to extract information from any unprotected cryptographic implementation. This threat necessitates the utilization of proper countermeasures against this attack.

LOW-LATENCY ENCRYPTION. The need for low-latency block cipher is ever increasing with the widespread of multiple technologies using microcontrollers. The NIST set a specific security level for lightweight cryptography that the known low-latency ciphers cannot reach. As a low-latency cipher would probably be deployed in a larger environment using such an Authenticated Encryption with Associated Data (AEAD) primitive, it makes sense to expect this low-latency cipher to reach the same security level.

BOOLEAN FUNCTIONS are the smallest building components in the symmetric cryptography. Studying the properties of all n-bit Boolean functions is not an easy task when $n > 5$. The applied vectorial Boolean functions in cryptography with $n > 5$ are usually found by their mathematical properties, and their implementations are considered in second priority. One of the hardware properties of (vectorial) Boolean functions is their latency. Finding the minimum possible latency of implementing a Boolean function with a synthesizer is not possible if the number of inputs is high and only uses the truth table of the Boolean function.

SECURITY ANALYSIS of cryptographic primitives is an important task not only during the design process by the designers but also afterward by the other researchers. There are several kinds of attacks and techniques of security analysis for block ciphers. The *differential* and the *linear* cryptanalyses are the two most prominent statistical attacks; thus, block ciphers' resistance against them must be studied carefully. However, evaluating or even approximating the probability of a differential or the correlation of a linear hull is not an easy task. As *truncated characteristics* are the essential objects for assessing many modern ciphers' security against differential and linear cryptanalysis, the absence of suitable algorithms to tackle this problem was – and still is – an interesting research area within symmetric cryptography.

1.2 OUTLINE AND CONTRIBUTIONS

In Chapter 2, we provide the necessary background information used in this thesis. It includes the notations and basics about Boolean functions, an overview of block ciphers, some cryptanalysis techniques, and the hardware concerns in implementing block ciphers.

In Chapter 3, first, we present a methodology for implementing the code-based concurrent error-detection schemes. Later, we present an implementation strategy based on the underlying code guarantees detection of up to a certain number of faults. Then, we present the tweakable block cipher CRAFT where the efficient protection of its implementations against differential fault analysis has been one of the main design criteria. We also introduce a methodology that applies error-correction-codes as a countermeasure against statistical ineffective fault analysis. The proposed concurrent error-correction scheme guarantees the correction of faults up to a certain number of faults.

In Chapter 4, we modify the block cipher PRINCE key schedule to improve the cipher's security level while minimizing the induced overhead, especially in a situation where the PRINCE is already deployed. We propose the block cipher PRINCEV2 that only has minimal overhead compared to PRINCE while still reaching the required higher security level. We also study the latency of the S-boxes in this chapter. We introduce a new metric to mathematically measure the latency of a given Boolean function together with an equivalency class of the Boolean functions that the latency metric of the functions within the same class is invariant under this equivalency. We present an algorithm to find all the low-latency Boolean functions efficiently. Later, using the low-latency balanced Boolean functions and an efficient algorithm, we build

up to 6-bit bijective low-latency S-boxes. As a result, we present several 5- and 6-bit low-latency S-boxes with cryptographically good properties.

In Chapter 5, we study the EDP of (truncated) differentials and the ELP of (multidimensional) linear hulls. We first show that the previous primary method for approximating the EDP of a truncated differential is not reliable and may produce estimations very different from the correct value. Then, we introduce new methods based only on the independent round-keys assumption, allowing us to practically compute the EDP of (truncated) differentials and the ELP of (multidimensional) linear hulls. Later, we apply these methods to various recent SPN ciphers.

In Chapter 6, we provide a detailed analysis of the security of CRAFT and PRINCEv2. We provide the argument to show the security claim for CRAFT considering the cipher's security against accelerated exhaustive search, time-data-memory trade-off, (impossible) differential, (zero-correlation) linear, meet-in-the-middle, integral, division property, and nonlinear invariant attacks. Then, based on the previously published analysis of PRINCE, we analyze the security of PRINCEv2. It is shown that several attacks successful against PRINCE, such as specific accelerated exhaustive search and MitM attacks, do not apply to PRINCEv2. Additionally, we provide several new analyses that include linear attack, division-property-based integral distinguisher, boomerang attack, and a new Demirci-Selçuk meet-in-the-middle attack. Finally, we provide a better understanding of differential attacks of the block cipher PRIDE. We show that two previous differential attacks on the cipher are incorrect, and based on this understanding, we show how to mount a differential attack properly.

Contribution

Large parts of this thesis are based on joint work with other co-authors. Therefore, we start each chapter that contains joint work by mentioning the particular author's contribution to the results.

1.3 PUBLICATIONS

During my doctoral studies at the Ruhr-University Bochum, I worked on several projects. The following list is the published outcome for some of those projects.

> Anita Aghaie, Amir Moradi, Shahram Rasoolzadeh, Aein Rezaei Shahmirzadi, Falk Schellenberg, and Tobias Schneider. "Impeccable Circuits". In: *IEEE Trans. Computers* 69.3 (2020), pp. 361–376. DOI: 10.1109/TC.2019.2948617. URL: https://doi.org/10.1109/TC.2019.2948617

> Aein Rezaei Shahmirzadi, Shahram Rasoolzadeh, and Amir Moradi. "Impeccable Circuits II". in: *57th ACM/IEEE Design Automation Conference, DAC 2020, San Francisco, CA, USA, July 20-24, 2020*. IEEE, 2020, pp. 1–6. DOI: 10.1109/DAC18072.2020.9218615. URL: https://doi.org/10.1109/DAC18072.2020.9218615

Christof Beierle, Gregor Leander, Amir Moradi, and Shahram Rasoolzadeh. “CRAFT: Lightweight Tweakable Block Cipher with Efficient Protection Against DFA Attacks”. In: *IACR Trans. Symm. Cryptol.* 2019.1 (2019), pp. 5–45. ISSN: 2519-173X. DOI: 10.13154/tosc.v2019.i1.5-45

Dušan Božilov, Maria Eichlseder, Miroslav Knežević, Baptiste Lambin, Gregor Leander, Thorben Moos, Ventzislav Nikov, Shahram Rasoolzadeh, Yosuke Todo, and Friedrich Wiemer. “PRINCEv2: More Security for (Almost) No Overhead”. In: *SAC 2020*. Ed. by Orr Dunkelman, Michael J. Jacobson Jr., and Colin O’Flynn. Vol. x. LNCS. Springer, Heidelberg, Dec. 2020, pp. x–x

Maria Eichlseder, Gregor Leander, and Shahram Rasoolzadeh. “Computing Expected Differential Probability of (Truncated) Differentials and Expected Linear Potential of (Multidimensional) Linear Hulls in SPN Block Ciphers”. In: *INDOCRYPT 2020*. Ed. by Karthikeyan Bhargavan, Elisabeth Oswald, and Manoj Prabhakaran. Vol. 12578. LNCS. Springer, Heidelberg, Dec. 2020, pp. 345–369. DOI: 10.1007/978-3-030-65277-7_16

Virginie Lallemand and Shahram Rasoolzadeh. “Differential Cryptanalysis of 18-Round PRIDE”. in: *INDOCRYPT 2017*. Ed. by Arpita Patra and Nigel P. Smart. Vol. 10698. LNCS. Springer, Heidelberg, Dec. 2017, pp. 126–146

Anne Canteaut, Eran Lambooij, Samuel Neves, Shahram Rasoolzadeh, Yu Sasaki, and Marc Stevens. “Refined Probability of Differential Characteristics Including Dependency Between Multiple Rounds”. In: *IACR Trans. Symm. Cryptol.* 2017.2 (2017), pp. 203–227. ISSN: 2519-173X. DOI: 10.13154/tosc.v2017.i2.203-227

2

Preliminaries

"Let's have a bit of fun, shall we?" —Neville Longbottom saying to Harry Potter in "Deathly Hallows" By J. K. Rowling

In this chapter, we provide the necessary background information. First, the notations and the basics about Boolean functions are discussed in Section 2.1. Then an overview of block ciphers is given in Section 2.2 before some analysis techniques are introduced in Section 2.3. Finally, the hardware concerns in implementing block ciphers are treated in Section 2.4.

2.1 BASICS AND NOTATIONS

We will use $\mathbb{N}, \mathbb{Z}$ for denoting the set of the natural numbers $\{1, 2, 3, \ldots\}$ and the set of the integers $\{\ldots, -2, -1, 0, 1, 2, \ldots\}$, resp. Besides, we will use $\mathbb{Z}_n$ to denote the finite set $\{0, 1, \ldots, n-1\}$ that is the set of non-negative integers smaller than a natural number n.

By $\mathbb{F}_2$, we denote the finite field of two elements, i. e., $\{0, 1\}$, and call it the *binary field* where the addition of this field will be denoted by $\oplus$ and called *XOR*. By $\mathbb{F}_2^n$ with $n \in \mathbb{N}$, we denote the binary vector space of dimension n and call it the *space of n-bit vectors*. Besides, by $\mathbb{F}_{2^n}$, we denote the extension of the binary field with 2^n elements.[1]

Let a be in $\mathbb{F}_2^n$, then by $a[i]$ with $i \in \mathbb{Z}_n$, we denote the i-th[2] element of a, i. e., $a = (a[0], \ldots, a[n-1])$.

Let $a, b \in \mathbb{F}_2^n$ be two n-bit binary vectors. We use $\langle a, b \rangle$ to denote the inner product between a and b which is defined as $\langle a, b \rangle = \bigoplus_{i=0}^{n-1} a[i]b[i]$. Also, by $\mathrm{hw}(a)$ and $\mathrm{hp}(a)$, we denote $\sum_{i=0}^{n-1} a[i]$ and $\bigoplus_{i=0}^{n-1} a[i]$, those are called the *Hamming weight* and the *Hamming parity* of a, resp.

We write $a \preceq b$ if and only if, for all i values, we have $a[i] \leq b[i]$, i. e., $a[i]$ is 0 whenever $b[i]$ is 0.

To denote concatenation of two vectors $a \in \mathbb{F}_2^n$ and $b \in \mathbb{F}_2^m$, we use $(a \| b)$ that is $(a[0], \ldots, a[n-1], b[0], \ldots, b[m-1])$.

By $(\mathbb{F}_2^s)^m$ with $s, m \in \mathbb{N}$, we denote the 2-dimensional binary vectors, i. e., for $a \in (\mathbb{F}_2^s)^m$, each of $a[i]$ with $i \in \mathbb{Z}_m$, is an s-bit binary vector. Then $a[i][j]$ denotes the j-th bit of $a[i]$. Besides, we call $a[i]$, the i-th *word* of a.

Definition 1 ($\phi_{s,m}$ Mapping)**.** We use $\phi_{s,m}$ as the mapping of elements in $(\mathbb{F}_2^s)^m$ to the elements in $\mathbb{F}_2^{s \cdot m}$ by concatenating the vectors. That is $\phi_{s,m} : (\mathbb{F}_2^s)^m \to \mathbb{F}_2^{s \cdot m}$ such that if $A = \phi_{s,m}(a)$ then $A[i \cdot s + j] = a[i][j]$ for all $i \in \mathbb{Z}_m$ and $j \in \mathbb{Z}_s$.

[1] In some literature, it is also denoted by $\mathrm{GF}(2^n)$ and is usually called the *Galois Field with 2^n elements*.

[2] Note that in this thesis, we always count starting from 0.

To make it easier and space-efficient to display a binary vector, for $a \in \mathbb{F}_2^n$, instead of displaying its all binary elements, we will show it by its corresponding integer value in $\mathbb{Z}_{2^n}$. Precisely, we use the simple mapping of elements in $\mathbb{F}_2^n$ to the elements in $\mathbb{Z}_{2^n}$ that maps any $a \in \mathbb{F}_2^n$ to $\sum_{i=0}^{n-1} a[i] \cdot 2^{n-i-1}$.

Similarly, we use a similar mapping of elements in $(\mathbb{F}_2^s)^m$ to the elements in $\mathbb{Z}_{2^{s \cdot m}}$ that maps each $a \in (\mathbb{F}_2^s)^m$ to $\sum_{i=0}^{m-1} 2^{s\cdot(m-i-1)} \cdot \left(\sum_{j=0}^{s-1} a[i][j] \cdot 2^{s-j+1}\right) = \sum_{i=0}^{s\cdot m-1} \phi_{s,m}(a)[i] \cdot 2^{s\cdot m-i+1}$.

Also, for $a \in \mathbb{F}_2^n$, $a[0]$ and $a[n-1]$ are called the *Most Significant Bit (MSB)* and the *Least Significant Bit (LSB)*, resp., while for $a \in (\mathbb{F}_2^s)^m$, $a[0]$ and $a[n-1]$ are called the *most significant word* and the *least significant word*, resp.

2.1.1 *Boolean Functions*

The functions from the vector space $\mathbb{F}_2^n$ to the binary field $\mathbb{F}_2$ are called *Boolean functions* with n-variables or simply n-bit Boolean functions. We use $\mathcal{B}_n$ to denote the set of all n-bit Boolean functions.

The Boolean functions have an important role in cryptography: the cryptographic transformations (e. g., generators in stream ciphers, S-boxes in block ciphers) can be designed by using an appropriate composition of non-linear Boolean functions. For efficiency reasons, the S-boxes used in most block ciphers are the ones with at most 8 variables, and in the case of key-stream generators of the stream ciphers, the number of variables is most often limited to 20.

The number of n-bit Boolean functions, i. e., the size of $\mathcal{B}_n$, is 2^{2^n}, and this number is too large to study the properties of each n-bit Boolean function when $n > 5$. For this reason, determining and studying those Boolean functions satisfying the desired conditions is not feasible through an exhaustive computer search.[3] Therefore, it is necessary to find solutions that make it easier to study Boolean functions' properties or find Boolean functions with desired properties.

[3] Consider determining whether an n-bit Boolean function has the desired properties needs one millisecond (10^{-6} seconds). Then it would need about half a million years to visit all 6-bit Boolean functions.

In the following, we briefly explain the necessary terms and notations of Boolean functions used in this thesis.

TRUTH TABLE: It is the most basic way to represent a Boolean function. Let f be an n-bit Boolean function, then the truth table of f is a binary vector $T_f \in \mathbb{F}_2^{2^n}$ such that for any $x \in \mathbb{F}_2^n$, $T_f[x]$ shows the value of $f(x)$, where x is represented as an integer.

ALGEBRAIC NORMAL FORM (ANF): Among the classical representations of Boolean functions, the one most often used in cryptography is the ANF that is the n-variable polynomial representation over $\mathbb{F}_2$ of the form

$$f(x) = \bigoplus_{I \in \mathbb{F}_2^n} a_I x^I = \bigoplus_{I \in \mathbb{F}_2^n} a_I \Big(\prod_{i=0}^{n-1} x_i^{I[i]}\Big),$$

where it is considered x_i to be the variable corresponding to the i-th bit of x, i. e., $x[i]$. Note that every coordinate x_i appears in this polynomial with exponents at most 1. This is because every bit in $\mathbb{F}_2$ equals its square, and

each a_I is a binary value. It is well-known that the ANF representation is unique[4] and can be computed for the given truth table with a complexity of $n \cdot 2^n$ operations [Car07, Proposition 1].

[4] Precisely, the ANF polynomial in $\mathbb{F}_2[x_0, \ldots, x_{n-1}]/(x_0^2 - x_0, \ldots, x_{n-1}^2 - x_{n-1})$ is unique.

[Car07] Carlet, "Boolean Functions for Cryptography and Error Correcting Codes"

The Hamming weight of a Boolean function's truth table is called its *weight*, which is the number of $x \in \mathbb{F}_2^n$ such that $f(x) = 1$. *Balanced Boolean functions* are the ones whose weight is equal to 2^{n-1}, i. e., for half of $x \in \mathbb{F}_2^n$, it maps to 1, and for the other half, it maps to 0.

ALGEBRAIC ANF DEGREE: It is the maximum Hamming weight of all occurring monomials in the ANF representation of a function which we denote by $\deg(f)$, i. e.,

$$\deg(f) = \max_{\substack{I \in \mathbb{F}_2^n \\ a_I = 1}} \mathrm{hw}(I).$$

Besides, the algebraic ANF degree of the constant-zero function is equal to $-\infty$.

Linear Boolean functions are those Boolean functions for which, for any $a, b \in \mathbb{F}_2^n$, we have $f(a \oplus b) = f(a) \oplus f(b)$. It is noteworthy that each linear Boolean function in $\mathcal{B}_n$ can be represented as $\ell_\alpha(x) = \langle \alpha, x \rangle$, where $\alpha \in \mathbb{F}_2^n$. Therefore, there are 2^n linear functions in $\mathcal{B}_n$, and their algebraic degree is always 1 (except for the case that $\alpha = 0$ that then the algebraic degree is 0 for the constant-one function and $-\infty$ for constant-zero function). Besides, as it is shown in [Car07, Lemma 1], each of these functions is balanced.

Separating Boolean functions by their algebraic degree, the ones with degree one, two, or three are called *affine*, *quadratic*, and *cubic* functions, resp. Affine functions, which are the extension of linear functions by a constant XOR in the output, can be displayed as $\langle \alpha, x \rangle \oplus c$ with corresponding $\alpha \in \mathbb{F}_2^n$ and $c \in \mathbb{F}_2$.

Vectorial Boolean Function

While Boolean functions map n-bit vectors to a one-bit value, *Vectorial Boolean functions* map n-bit vectors to m-bit vectors. To specify the input and output bit size of these functions, we call them n to m-bit Boolean functions, and when the input and output bit sizes are the same, we simply call them n-bit vectorial Boolean functions. Clearly, the vectorial Boolean functions include the (single-output) Boolean functions which correspond to $m = 1$.

Let F be an n to m-bit Boolean function, then the Boolean functions $f_0, \ldots, f_{m-1}$ defined by $F(x) = \big(f_0(x), \ldots, f_{m-1}(x)\big)$, $x \in \mathbb{F}_2^n$ are called *coordinate functions* of F. Given F function, to refer to its i-th coordinate function f_i, we use the notation $F[i]$. Also, for every non-zero $\alpha \in \mathbb{F}_2^m$, the Boolean function $x \mapsto \langle \alpha, F(x) \rangle$ is called a *component function* of F, and we denote this function by $\langle \alpha, F(x) \rangle$. In this thesis, to denote the truth table of F easily, we use an array of 2^n elements in $\mathbb{Z}_{2^m}$, i. e., $\big(F(0), \ldots, F(2^n - 1)\big)$.

As for Boolean functions, balancedness plays a crucial role in vectorial Boolean functions. An n to m-bit Boolean function F is called balanced if it takes every value of $\mathbb{F}_2^m$ the same number of times, i. e., 2^{n-m} times. The balanced n-bit vectorial Boolean functions are the permutations on $\mathbb{F}_2^n$. It is shown in [LN96] that an n to m-bit Boolean function F is balanced if and

[LN96] Lidl and Niederreiter, *Finite Fields*

only if all component functions are balanced.

The algebraic degree of F is the maximum of the algebraic degrees of all coordinate functions. Hence, we use the same definition for linear, affine, quadratic, and cubic functions of the vectorial Boolean functions.

It is noteworthy to mention that each n to m-bit linear Boolean function L can be displayed by an $n \times m$ binary matrix M in such a way that

$$L : (x_0, \ldots, x_{n-1}) \mapsto (x_0, \ldots, x_{n-1}) \cdot M.$$

[5] S is shortened for *Substitution*.

In cryptography, vectorial Boolean functions are usually called *S-boxes*,[5] which provide confusion in the cipher. The S-boxes play a primary role in the *Key-Alternating Block Ciphers*, especially in the Substitution-Permutation-Network (SPN) ones. It is explained in detail in Section 2.2.

Equivalences

To study properties of vectorial Boolean functions, it is sometimes easier to partition them by a defined equivalence relation for which the studying properties are invariant. *Linear equivalence* and *affine equivalence* are the most applied ones in studying Boolean functions.

Definition 2 (Linear, Affine, and Bit Permutation Equivalences)**.** Two n to m-bit Boolean functions F and G are called *linear equivalent* if there exist an n to n-bit linear bijection L_i and an m to m-bit linear bijection L_o in such a way that $F = L_o \circ G \circ L_i$.

In the same way, *affine equivalence* and *bit permutation equivalence* are defined. F and G are called affine equivalent if there exist an n to n-bit affine bijection A_i and an m to m-bit affine bijection A_o in such a way that $F = A_o \circ G \circ A_i$; and they are called bit permutation equivalent, if there exist a bit permutation of n bits P_i and a bit permutation of m bits P_o in such a way that $F = P_o \circ G \circ P_i$. Note that the Boolean function P is called bit permutation of n bits, if it maps $(x[0], \ldots, x[n-1])$ to $\big(x[\pi(0)], \ldots, x[\pi(n-1)]\big)$ where π is a permutation of $\mathbb{Z}_n$.

It is clear that if F and G are bit permutation equivalent, then they are also linear and affine equivalents. Besides, if they are linear equivalent, then they are also affine equivalent.

2.2 BLOCK CIPHERS

Definition 3 (Block Cipher)**.** Let $\mathcal{K}$ and $\mathcal{M}$ be two finite sets. A *block cipher* is function $\mathtt{Enc} : \mathcal{K} \times \mathcal{M} \to \mathcal{M}$ such that for each fixed K, $\mathtt{Enc}_K(\cdot) := \mathtt{Enc}(K, \cdot)$ is a permutation of $\mathcal{M}$.

$\mathcal{K}$ is called the *key space* and $\mathcal{M}$ the *message space*. $\mathtt{Enc}_K$ is referred to as *encryption* (with key K) and its inverse $\mathtt{Dec}_K := \mathtt{Enc}_K^{-1}$ is the *decryption* (with key K). K is called the *master key* while P and C are the *plaintext* and *ciphertext*, resp.

Nowadays, since the modern ciphers are always implemented on computers, the message and the key spaces are almost always considered to be binary vector spaces. That is $\mathcal{K} = \mathbb{F}_2^\kappa$ and $\mathcal{M} = \mathbb{F}_2^n$, which κ is the bit size of

the key and n is the bit size for message blocks and are called *key size* and *block size*, resp.

Block ciphers' block size, depending on the application, are often chosen from some multiplications of 16; that is one of the values 32, 48, 64, 80, 96, 128, and 256. While implementing block ciphers of larger block size usually has expensive costs, using smaller ones reduces limits on the maximum number of plaintext blocks to be encrypted that can affect the cipher's security.[6] Therefore, dependent on block cipher's application, designer chooses large enough block size for whose it is possible to provide efficient implementation.

[6] For instance, in some modes of operations, ciphertexts of a 32-bit block cipher are distinguishable from a random sequence using about 2^{16} blocks.

The key size used in block ciphers is also often a multiplication of 16. In 2015, NIST published its new recommendations on the key sizes of cryptographic algorithms [BR15]. There it is recommended that the newly designed ciphers should provide a key length of at least 112 bits.

[BR15] Barker and Roginsky, "Recommendation for Transitioning the Use of Cryptographic Algorithms and Key Lengths Report on Lightweight Cryptography"

Product Block Cipher

For the first time, Shannon [Sha45, Part III, Section 37 and 38] introduced the *product cipher*'s[7] concept, and almost all of the block ciphers can be defined as a product cipher. The structure of this cipher is depicted in Figure 2.1.

[Sha45] Shannon *A Mathematical Theory of Cryptography*

[7] It is also called the *iterative* cipher.

Definition 4 (Product Block Cipher)**.** Let $\mathtt{R}_0, \ldots,$ and $\mathtt{R}_{r-1}$ be r block ciphers with the message space of $\mathbb{F}_2^n$ and the key spaces of $\mathbb{F}_2^{\kappa_0}, \ldots$ and $\mathbb{F}_2^{\kappa_{r-1}}$, resp. That is, for each $i \in \mathbb{Z}_r$, $\mathtt{R}_i : \mathbb{F}_2^{\kappa_i} \times \mathbb{F}_2^n \to \mathbb{F}_2^n$. Let also $\mathtt{KeySch} : \mathbb{F}_2^{\kappa} \to \mathbb{F}_2^{\kappa_0} \times \ldots \times \mathbb{F}_2^{\kappa_{r-1}}$. The block cipher $\mathtt{Enc}$ defined with $\mathtt{Enc}_K = \mathtt{R}_{r-1\ K_{r-1}} \circ \ldots \circ \mathtt{R}_{0\ K_0}$ and $(K_0, \ldots, K_{r-1}) = \mathtt{KeySch}(K)$ is called an r-round *product cipher*.

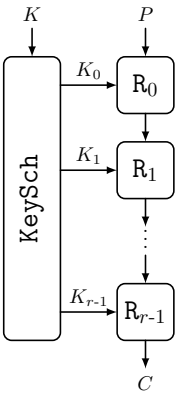


FIGURE 2.1: A Product Cipher.

In other words, a product cipher is built by iterating several (simpler) block ciphers, so-called *rounds*. Each round $\mathtt{R}_i$ uses a specific *round key* K_i derived from the master key K using the so-called *key-schedule* function $\mathtt{KeySch}$.

Using product ciphers provides both efficient implementation and more straightforward security analysis in the design of block cipher. In case that the round functions are similar to each other, it is possible to implement only one general round function that can be changed to each of the rounds in the block cipher with a simple modification. Besides, if the round functions are realized by *simple* functions that are possible to their cryptographic properties, it is possible to analyze the cipher's security more easily.

Key-Alternating Block Cipher

It is one kind of product cipher that uses a specific (simple) way to involve the round keys in the round functions. The key adding function is usually a simple function using a group operation, like (vectorial) modular addition of the round key with (part of) the input or output of each round.

The bit-wise XOR, which is also the vectorial modular addition modulo 2, is the most common key addition in symmetric cryptography. The addition modulo 2^s and swapping are other commonly applied key addition functions; however, their implementations depending on the platform, are usually more expensive than that of the bit-wise XOR.

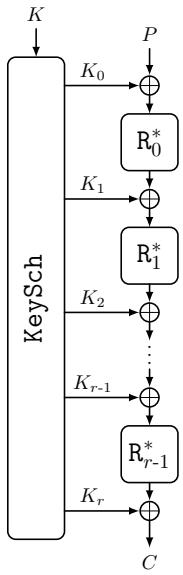


FIGURE 2.2: A Key-Alternating Cipher.

[8] It also called *key-less round function*.

[9] If we consider each round to start with the round permutation as the first component and then the key addition, the first key addition ($\mathtt{A}_{K_0}$) is called a *pre-whitening key* and serves the same purpose.

[10] In the case that the round key size is less than n, we can define an equivalent key schedule where the key-less part is filled by $\mathtt{0}$.

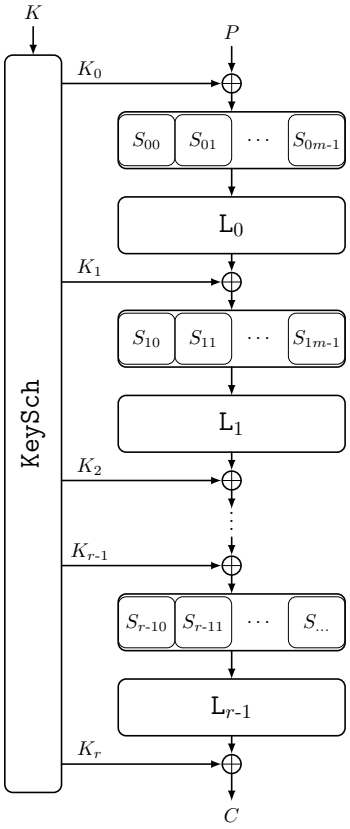


FIGURE 2.3: An SPN Cipher.

We simply use $\mathtt{A} : \mathbb{F}_2^{\kappa'} \times \mathbb{F}_2^n \to \mathbb{F}_2^n$ to denote the key adding function, and to specify that the key K is getting added, we define $\mathtt{A}_K := \mathtt{A}(K, \cdot)$.

Definition 5 (Key-Alternating Block Cipher)**.** Let $\mathtt{R}_0^*, \ldots,$ and $\mathtt{R}_{r-1}^*$ be r permutations of $\mathbb{F}_2^n$, $\mathtt{KeySch} : \mathbb{F}_2^\kappa \to (\mathbb{F}_2^{\kappa'})^{r+1}$ and $\mathtt{A} : \mathbb{F}_2^{\kappa'} \times \mathbb{F}_2^n \to \mathbb{F}_2^n$ be a key adding function. The block cipher $\mathtt{Enc}$ defined by $\mathtt{Enc}_K = \mathtt{A}_{K_r} \circ \mathtt{R}_{r-1}^* \circ \mathtt{A}_{K_{r-1}} \circ \ldots \circ \mathtt{A}_{K_1} \circ \mathtt{R}_0^* \circ \mathtt{A}_{K_0}$ and $(K_0, \ldots, K_r) = \mathtt{KeySch}(K)$ is called an r-round *key-alternating block cipher*.

Each round $\mathtt{R}_i$ consists of a single key addition $\mathtt{A}_{K_i}$ and a round permutation[8] $\mathtt{R}_i^*$, i. e., $\mathtt{R}_i = \mathtt{R}_i^* \circ \mathtt{A}_{K_i}$. By the input of a round, we always refer to the input of the key addition and by the output, we refer to the output of the round permutation. After r rounds, there is an extra key addition (*post-whitening key*), which ensures that an adversary cannot peel off the last round without any guesses on the round key.[9]

In this thesis, we consider that the key adding function is the bit-wise XOR of the round key with the state, i. e., for any $K \in \mathbb{F}_2^n$, the function $\mathtt{A}_K := \mathtt{A}(K, \cdot)$ maps X to $X \oplus K$. Note that here, we consider that the size of each round key is n bits.[10] The structure of the considered key-alternating cipher is depicted in Figure 2.2.

Substitution-Permutation-Network (SPN)

In the same article as the one for product ciphers [Sha45, Part III, Section 35], Shannon introduced the idea of *confusion* and *diffusion*. While confusion suggests using non-linear functions for locally spreading the encryption's statistical properties, diffusion suggests using linear functions between two consecutive non-linear functions to spread these properties globally. This iterative usage of Shannon's confusion and diffusion principle is the standard design of modern block ciphers.

SPN block ciphers are key-alternating block ciphers that use Shannon's principle to define a particular structure of the rounds. Thereby, the round permutations $\mathtt{R}_i^*$ consist of a *non-linear* function $\mathtt{S}_i$ (called *substitution layer* or *S-box layer*) for confusion and a *linear* function $\mathtt{L}_i$ (also called *linear layer*) for diffusion. Besides, the S-box layer is realized by applying several parallel smaller functions, so-called *S-boxes*.

Definition 6 (Substitution-Permutation-Network)**.** Let each $\mathtt{S}_i$ with $i \in \mathbb{Z}_r$ be the function of applying m smaller functions $S_{i,0}, \ldots, S_{i,m-1}$ in parallel where all the $S_{i,j}$ S-boxes are permutations of $\mathbb{F}_2^s$ and $n = s \cdot m$; i. e., $\mathtt{S}_i : \mathbb{F}_2^{s \cdot m} \to \mathbb{F}_2^{s \cdot m}$ and for all $X \in \mathbb{F}_2^{s \cdot m}$ and $x \in (\mathbb{F}_2^s)^m$ such that $X = \phi_{s,m}(x)$, we have

$$\mathtt{S}_i(X) = \phi_{s,m}\Big(\big(S_{i,0}(x[0]), \ldots, S_{i,m-1}(x[m-1])\big)\Big).$$

Let $\mathtt{L}_i$ be a linear bijection in $\mathbb{F}_2^n$. The key-alternating block cipher $\mathtt{Enc}$ defined by rounds $\mathtt{R}_i^* = \mathtt{L}_i \circ \mathtt{S}_i$ and $\mathtt{KeySch} : \mathbb{F}_2^\kappa \to (\mathbb{F}_2^n)^{r+1}, (K_0, \ldots, K_r) = \mathtt{KeySch}(K)$ is called an r-round *SPN block cipher*.

$$\mathtt{Enc}_K = \mathtt{A}_{K_r} \circ \mathtt{L}_{r-1} \circ \mathtt{S}_{r-1} \circ \mathtt{A}_{K_{r-1}} \circ \ldots \circ \mathtt{L}_0 \circ \mathtt{S}_0 \circ \mathtt{A}_{K_0}\,.$$

The parameter s is the bit size of the S-boxes and called *S-box size*, and it can be any integer larger than 2. In general, the bit size of the S-boxes

can be different, but we consider that all of them have the same size for simplicity.

The structure of an SPN block cipher is depicted in Figure 2.3. This is the general structure that we use in the rest of the thesis. To denote the input or output of each layer – so-called *states*–, we use X_i, Y_i, and Z_i, to refer to the states before $\mathtt{S}_i$, before $\mathtt{L}_i$, and after the $\mathtt{L}_i$ layers of the i-th round, resp. That is

$$\begin{aligned} X_i &= \mathtt{A}_{K_i} \circ \mathtt{L}_{i-1} \circ \mathtt{S}_{i-1} \circ \mathtt{A}_{K_{i-1}} \circ \ldots \circ \mathtt{L}_0 \circ \mathtt{S}_0 \circ \mathtt{A}_{K_0}(P), \\ Y_i &= \mathtt{S}_i \circ \mathtt{A}_{K_i} \circ \mathtt{L}_{i-1} \circ \mathtt{S}_{i-1} \circ \mathtt{A}_{K_{i-1}} \circ \ldots \circ \mathtt{L}_0 \circ \mathtt{S}_0 \circ \mathtt{A}_{K_0}(P), \\ Z_i &= \mathtt{L}_i \circ \mathtt{S}_i \circ \mathtt{A}_{K_i} \circ \mathtt{L}_{i-1} \circ \mathtt{S}_{i-1} \circ \mathtt{A}_{K_{i-1}} \circ \ldots \circ \mathtt{L}_0 \circ \mathtt{S}_0 \circ \mathtt{A}_{K_0}(P). \end{aligned}$$

Note that, as mentioned before, to show the state's value, we will use the corresponding integer value in $\mathbb{Z}_{2^{s \cdot m}}$.

All the $S_{i,j}$ functions can be the same or chosen from a smaller set of S-boxes. If all of the S-boxes are the same, we will use S to denote them. Similarly, if all the $\mathtt{L}_i$ are the same functions, we will use L to denote them.

MATRIX OF LINEAR LAYER: The corresponding matrix of each linear layer $\mathtt{L}_i$ is denoted by M_i. That is for any $X \in \mathbb{F}_2^n$, we have $\mathtt{L}_i(X) = X \cdot M_i$. Besides, $\mathtt{L}_i^{-1}$ and $\mathtt{L}_i^T$ are used to denote the linear mappings corresponding to the matrix multiplication with the inverse and transposed matrix of M_i, resp.

Since the S-box layer is separated to m smaller S-boxes, sometimes it is easier also to separate the linear layer to m smaller linear operations. Precisely, consider that M, the corresponding matrix for linear layer L, is split to m^2 smaller $s \times s$ sub-matrices in the following way:

$$M = \begin{bmatrix} M_{0,0} & \cdots & M_{0,m-1} \\ \cdots & \cdots & \cdots \\ M_{m-1,0} & \cdots & M_{m-1,m-1} \end{bmatrix}.$$

Then for any $x \in (\mathbb{F}_2^s)^m$, we have $\mathtt{L} \circ \phi_{s,m}(x) = \phi_{s,m}(y)$ with $y \in (\mathbb{F}_2^s)^m$ such that each of its elements can be given as

$$y[k] = \bigoplus_{j=0}^{m-1} x[j] \cdot M_{j,k}.$$

Furthermore, the linear operation related to multiplying with matrix $M_{k,j}$ is denoted by $L_{k,j}$:

$$y[k] = \bigoplus_{j=0}^{m-1} L_{j,k}(x[j]).$$

Note that in general, it is not necessary that all the $M_{j,k}$ matrices to be non-singular, while it is necessary that M to be a non-singular matrix.

Definition 7 ($\mathbb{F}_{2^s}$ Multiplication or Binary Multiplication Linear Layer)**.** A linear layer L which each of its corresponding sub-matrices $M_{j,k}$ being the corresponding matrix for multiplication with $a_{j,k}$, an element of $\mathbb{F}_{2^s}$, is called a $\mathbb{F}_{2^s}$ *multiplication linear layer*. Besides, if each of the corresponding sub-functions $M_{j,k}$ is either the identity matrix or the all-zeros matrix, then the linear layer is called a *binary multiplication linear layer*.

For a binary multiplication linear layer, the output of L for any $x \in (\mathbb{F}_2^s)^m$, can be realized by a scalar products. Therefore, each element of y will be equal to

$$y[k] = \bigoplus_{j=0}^{m-1} b_{j,k} \cdot x[j].$$

Furthermore, the $m \times m$ matrix B build by $b_{j,k}$ elements, is called *binary representative matrix* of the linear layer L.

For instance, the linear layer of AES is a $\mathbb{F}_{2^8}$ multiplication linear layer, and the linear layers for SKINNY, MANTIS [Bei+16], and MIDORI [Ban+15] block ciphers are all binary multiplication ones.

[Bei+16] Beierle, Jean, Kölbl, Leander, Moradi, Peyrin, Sasaki, Sasdrich, and Sim, "The SKINNY Family of Block Ciphers and Its Low-Latency Variant MANTIS"

[Ban+15] Banik, Bogdanov, Isobe, Shibutani, Hiwatari, Akishita, and Regazzoni, "Midori: A Block Cipher for Low Energy"

Other Kinds of Key-Alternating Block Ciphers

An SPN is not the only instantiation of a block cipher. Feistel and ARX are the other well-known structures to build block ciphers. Since these structures are not the focus of this thesis, we do not explain them in detail.

FEISTEL NETWORK: Another important structure for constructing block ciphers is the so-called *Feistel network* or *Feistel cipher* named after its inventor Horst Feistel [Fei71]. LUCIFER [Sor84], the predecessor of the DES, is the first block cipher using this structure. Also, the DES, which was developed in the '70s and was published as a US FIPS standard in 1977 [FIP77], is the most famous block cipher before the AES in 2000.

[Fei71] Feistel, *Block Cipher Cryptographic System*

[Sor84] Sorkin, "Lucifer, A Cryptographic Algorithm"

[FIP77] FIPS, "46: Data Encryption Standard"

ADDITION ROTATION XOR (ARX) CIPHERS: ARX ciphers, instead of using small S-boxes as of the non-linear operation, use arithmetic operations. Precisely, for efficiency reasons, they apply the most basic arithmetic operations, that the most common operations are (i) addition in modulo 2^s (with $s > 1$), (ii) bit-wise rotation (or shifting), and (iii) XOR. The cipher FEAL cipher designed by Shimizu and Miyaguchi [SM88] was the first of this kind.

[SM88] Shimizu and Miyaguchi "Fast Data Encipherment Algorithm FEAL"

Key Schedule

The key schedule calculates the round keys from the master key in the product ciphers, as mentioned previously. The key schedule is usually an updating function that, using the previous state of the key, computes a new one, and therefore some part of the key state is used as the round key. The updating function for each round is generally the same, except for round constants that are fixed values to omit the equality between the rounds.

Recently, in many block ciphers, the key schedule is designed quite simply; the κ-bit key is split to p pieces of $\frac{\kappa}{p}$ bits, and these p pieces XORed with a constant value are used repeatedly for the rounds of the cipher.

In some works, it is shown that the change of a key schedule can affect the security of the block cipher,[11] but generally, this effect is not well studied. However, in the analysis of block ciphers, it is always assumed that all the round keys are independent (see Assumption 16) which helps for a more straightforward analysis of the cipher, but of course, it is an unrealistic assumption for practical ciphers.

[11] For instance, KATAN and KTANTAN block ciphers [DDK09] are using the same encryption with different key schedules. It is shown in [BR11] that the security of KTANTAN is much less than the other one's security.

ROUND CONSTANTS: After the introduction of *Slide attacks* [BW99], *round constant* are included in the key schedule to provide security against these attacks. The attack works on a cipher if it is possible to break it down into multiple sub-ciphers with an identical function. The role of round constants is to eliminate the existence of such identical functions.

[BW99] Biryukov and Wagner, "Slide Attacks"

FX CONSTRUCTION: In 1984, Ronald Rivest proposed a simple solution (known as DESX) to address the concern regarding the small key size of DES. The DESX construction XORs two independent whitening keys at the beginning and the end of the core DES encryption process, resp. Later, this construction was generalized to the so-called FX construction by Kilian and Rogaway [KR96].

[KR96] Kilian and Rogaway "How to Protect DES Against Exhaustive Key Search"

Definition 8 (FX Construction)**.** Let $\texttt{Enc}^* : \mathbb{F}_2^\kappa \times \mathbb{F}_2^n \mapsto \mathbb{F}_2^n$ be a block cipher with key and block size of κ and n bits, resp. The FX construction $\texttt{Enc}$ with the core cipher of $\texttt{Enc}^*$ is defined as

$$\texttt{Enc} : (\mathbb{F}_2^n \times \mathbb{F}_2^n \times \mathbb{F}_2^\kappa) \times \mathbb{F}_2^n \mapsto \mathbb{F}_2^n,$$
$$\texttt{Enc}_{K_2,K_1,K_0}(X) = \texttt{Enc}^*_{K_0}(X \oplus K_1) \oplus K_2\,.$$

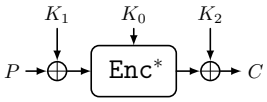


FIGURE 2.4: The FX Construction.

Figure 2.4 depicts the FX construction. It is important to mention that using independent K_1 and K_2 whitening keys does not improve the whole construction's security. Due to this reason, in the application of the FX construction, K_2 is a bijective mapping (and usually a simple one) of K_1. The block ciphers PRIDE [Alb+14] and PRINCE [Bor+12] follow the FX construction where in the first one $K_2 = K_1$ and in the second one $K_2 = L(K_1)$ where L is a simple linear bijection.

[Alb+14] Albrecht, Driessen, Kavun, Leander, Paar, and Yalçin, "Block Ciphers - Focus on the Linear Layer (feat. PRIDE)"

[Bor+12] Borghoff et al., "PRINCE - A Low-Latency Block Cipher for Pervasive Computing Applications - Extended Abstract"

Tweakable Block Ciphers

Liskov, Rivest, and Wagner proposed a generalized version of block ciphers called *tweakable block ciphers* [LRW02]. Along with its usual plaintext (or ciphertext) and key inputs, a tweakable block cipher accepts another input called the tweak. While in the simple (un-tweakable) block cipher, the key selects the permutation computed by the cipher, in the tweakable block cipher, the tweak and the key together select the permutation. If changing tweaks is sufficiently low-cost (compared with a usually relatively expensive key setup operation), this primitive allows for better encryption modes and efficient constructions of authenticated encryption schemes [LRW02].

[LRW02] Liskov, Rivest, and Wagner, "Tweakable Block Ciphers"

Definition 9 (Tweakable Block Cipher)**.** A *tweakable block cipher* is function $\texttt{Enc} : \mathcal{K} \times \mathcal{T} \times \mathcal{M} \to \mathcal{M}$ with $C = \texttt{Enc}(K, T, P)$ such that for each fixed K and T, $\texttt{Enc}_K^T(\cdot) := \texttt{Enc}(K, T, \cdot)$ is a permutation of $\mathcal{M}$.

$\mathcal{T}$ is called the *tweak space*, and it is considered to be the same as $\mathbb{F}_2^t$. Besides, t is called the bit size of tweak.

Like the simple block ciphers, the tweakable block cipher can also be built by the product and iterative block ciphers. Thereby, instead of $\texttt{KeySch}$, we need a *tweakey schedule* $\texttt{TKSch}$ to compute all the round keys K_i and all the round tweaks T_i.

Since efficiency matters, sometimes the tweakey schedule is designed in a way that instead of computing one round key and one round tweak

for each round, it only computes a *tweakey* TK_i that is a function of both the master key K and the master tweak T. In this case, the structure of the cipher will be similar to the ones in Figures 2.1 and 2.2 where `KeySch` with input K is replaced by `TKSch` with inputs K and T, and round tweakeys TK_i replace round keys K_i. This structure is called the *tweakey framework* that is introduced by Jean, Nikolic, and Peyrin [JNP14] where SKINNY, MANTIS [Bei+16] and QARMA [Ava17] block ciphers follow this structure.

[JNP14] Jean, Nikolic, and Peyrin "Tweaks and Keys for Block Ciphers: The TWEAKEY Framework"

2.3 CRYPTANALYSIS OF BLOCK CIPHERS

The primary purpose of block ciphers is to provide *confidentiality*, i. e., protecting data from a third party, the so-called *adversary* or, the *attacker*. Thus, the designer needs to design a secure encryption scheme that does not allow the adversary to deduce any sensitive information of the encryption. Depending on the adversary's capabilities, this sensitive information can be any information from the plaintext or the key. Therefore, it is necessary to be precise about what kind of capabilities the adversary has and what type of security the encryption scheme needs to provide.

In the following, we provide informal definitions[12] of the adversary models and different kinds of attacks.

[12] Since it is out of the scope of this thesis, we do not bring the precise and formal security definitions of the security notions.

2.3.1 *Adversary Models and Attack Types*

Typically, there are two categorizations for the adversary model. The first one is based on the type of access to the data in which the adversary is modeled as COA, KPA, CPA, or CCA.

CIPHERTEXT-ONLY ATTACK (COA): The adversary has only access to some ciphertexts from encryption with the same key (or related keys) and aims to gain some information on the plaintext(s) or the key.

KNOWN-PLAINTEXT ATTACK (KPA): The adversary has access to some plaintext/ciphertext pairs from encryption with the same key (or related keys) without the capability of choosing the plaintext or the ciphertext.

CHOSEN-PLAINTEXT ATTACK (CPA): The adversary has access to some plaintext/ciphertext pairs from encryption with the same key (or related keys) with the capability of choosing the plaintext. Therefore, he can encrypt any chosen plaintext.

CHOSEN-CIPHERTEXT ATTACK (CCA): The adversary has access to some plaintext/ciphertext pairs from encryption with the same key (or related keys) with the capability of choosing the value of either the plaintext or the ciphertext. Therefore, he can encrypt any chosen plaintext or decrypt any chosen ciphertext.

The second categorization is based on the key used in the encryptions.

FIXED-KEY MODEL: All the plaintext/ciphertext pairs are from encryption with a key that is fixed to a constant value chosen by or known to the

adversary. Here, the adversary aims to distinguish the cipher from a random permutation.

SINGLE-KEY MODEL: All the plaintext/ciphertext pairs are from encryption with the same key, and its value is unknown to the adversary.

RELATED-KEY MODEL: The plaintext/ciphertext pairs are from encryption with different keys, but these keys are related to each other, and this relation is known or chosen by the adversary.

There is a similar categorization about the tweak for tweakable block ciphers, i. e., the adversary can be modeled as either fixed-, single-, or related-tweak model. Besides, these categorizations can be mixed to be more precise about the capabilities of the adversary. For instance, in the single-key related-tweak model, all the plaintext/ciphertext pairs are from the encryption using the same key but from different tweak values. Typically, it is considered that the adversary model is single-key (or single-tweak), and in this case, for simplicity, it is not mentioned in the model type.

In the first categorization, KPA is the weakest, and CCA the strongest attacker model. Also, in the second categorization, the related-key model is the most strongest one.

More about the adversary, it is also important to clarify his aim, which is the so-called type of attack. *Distinguishing*, *Plaintext/Ciphertext Recovery*, and *Key Recovery* are the usual types of attacks.

DISTINGUISHING: The adversary aims to distinguish whether the plaintext/ciphertext pairs are from the target cipher or a random permutation.

PLAINTEXT/CIPHERTEXT RECOVERY: The adversary aims to encrypt a plaintext that is not queried yet.

KEY RECOVERY: The adversary aims to recover the secret key used in the encryption(s).

An attacker who can successfully recover the key or the plaintext/ciphertext can also make a successful distinguishing attack. Therefore, providing security against a distinguishing attack is the most generic goal of the designer.

In the following, we explain the most important techniques used in the analysis of block ciphers.

2.3.2 *(Accelerated) Exhaustive Search*

In an exhaustive search, the attacker guesses the key value and fully encrypts a known plaintext, and checks if it matches the given corresponding ciphertext. The exhaustive search has two variants: the online exhaustive search called the *brute force attack*, and the offline exhaustive search called the *dictionary attack*. In the brute force attack, by having a know plaintext/ciphertext pair, the attacker tries to guess the key value used in the encryption. But, in the dictionary attack, before mounting the attack, the attacker computes the corresponding ciphertext of a fixed chosen-plaintext with all possible key values and saves the ciphertext/key pairs in a sorted table (dictionary). Then in the attack, he queries the chosen plaintext's ciphertext, he finds the key using the dictionary.

While the computational complexity of brute-forcing to find the key of each encryption is 2^κ, in the dictionary attack, the complexity of attack is only one look-up to the dictionary, and this is in cost of doing 2^κ pre-computations and memory blocks in advance.

The *DES Cracker Machine* build by Electronic Frontier Foundation (EFF) [EFF98] and *COPACOBANA* [Kum+06; Cop] are examples of optimized machines to apply an exhaustive search.

[EFF98] EFF, *DES Cracker Machine*

[Kum+06] Kumar, Paar, Pelzl, Pfeiffer, and Schimmler, "Breaking Ciphers with COPACOBANA - A Cost-Optimized Parallel Code Breaker"

[Cop], *COPACOBANA: A Codebreaker for DES and other Ciphers*

While an exhaustive search always needs 2^κ (pre-)computations, sometimes, using some of the cipher's properties, it is possible to accelerate it.

Example 10. Examples for such accelerated exhaustive searches are

- DES's complementary property reduces the complexity of the exhaustive search by one bit.[13]
- α-reflection property introduced in [Bor+12] reduces the complexity of the exhaustive search by one bit for the cipher PRINCE.[14]
- In [Knu93], Knudsen found that even though the key size in LOKI block cipher is 64 bits [BPS90], its effective key size is only 60 bits. There, he showed that the encryption with key $K \in (\mathbb{F}_2^4)^{16}$ in LOKI cipher is the same as $K \oplus (x, \ldots, x)$ with any $x \in \mathbb{F}_2^4$. Therefore, in an exhaustive search, the attacker can fix the first nibble of the K to be equal to 0 and search through the other 15 nibbles.

[13] Complementary property of DES is that if $\mathtt{Enc}_K(P) = C$, then $\mathtt{Enc}_{\bar{K}}(\bar{P}) = \bar{C}$ where $\bar{X}$ denotes the bit-wise complement value of X.

[Bor+12] Borghoff et al., "PRINCE - A Low-Latency Block Cipher for Pervasive Computing Applications - Extended Abstract"

[14] A block cipher follows α-reflection property if $\mathtt{Dec}_K = \mathtt{Enc}_{K \oplus \alpha}$.

[Knu93] Knudsen, "Cryptanalysis of LOKI"

[BPS90] Brown, Pieprzyk, and Seberry, "LOKI - A Cryptographic Primitive for Authentication and Secrecy Applications"

Since 2015, after NIST's new recommendation about the key sizes of cryptographic algorithms [BR15], which is 112 bit for the block ciphers, to be resistant against brute force attacks, block cipher designers usually use at least 128-bit keys.

[BR15] Barker and Roginsky, "Recommendation for Transitioning the Use of Cryptographic Algorithms and Key Lengths Report on Lightweight Cryptography"

2.3.3 *Time-Data-Memory (TDM) Trade-offs*

Hellman [Hel80] was the first who introduced the TDM trade-off technique. This technique has two phases, so-called *offline* and *online* phases. The main idea behind this technique is that in the offline phase, the attacker chooses a fixed plaintext P_0 and m random values from the key space $\mathcal{K}$, namely $SK_0, \ldots, SK_{m-1}$, which are called *start points*. Then, using the function $F := \mathtt{Enc}(\cdot, P_0)$,[15] he computes the value of end points $EK_i := F^t(SK_i)$ for each start point; i. e., each end point is the t times iteratively encryption of the plaintext P_0 with consecutive key values starting with the corresponding start point. He saves the (SK_i, EK_i) pairs in a sorted table to use them in the next phase. In the online phase, by querying the corresponding ciphertext C_0 for the plaintext P_0, and computing $C_j := F^j(C)$ with $j < t$, the attacker checks to see if C_j is equal to one of the end points. If $C_j = EK_i$, then he starts with SK_i and computes $F^{t-j}(SK_i)$ to get the key value used in the encryption.

[Hel80] Hellman "A Cryptanalytic Time-Memory Trade-off"

[15] Note that here for simplicity, we assumed the block and the key length of the cipher is the same, but in general, it is possible to mount the attack for other cases.

The success probability of the above description is $m \cdot t \cdot 2^{-\kappa}$. To cover whole the key space and have success probability equal to 1, the attacker needs to repeat the offline phase t times but with differently modified F functions (this modification can be a very simple modification, e. g., bit rotation in the output). Therefore, the complexity of this technique is equal

to $m \cdot t^2 = 2^\kappa$ pre-computations in the offline phase, t^2 computations in the online phase, $m \cdot t$ blocks of memory and one chosen-plaintext data.

Comparing to the exhaustive search attacks, TDM trade-offs as it stands for making trade between the brute-force attack with high online computations and the dictionary attack with high memory use.

While Hellman's technique was only a time-memory trade-off, later, some techniques used data as another parameter for the trade-off. For instance, in [Din15], Dinur introduced new TDM trade-offs on the FX-constructions and showed that the security margins for PRINCE [Bor+12] and PRIDE [Alb+14] block ciphers are not that much as expected by the designers.

[Din15] Dinur, "Cryptanalytic Time-Memory-Data Tradeoffs for FX-Constructions with Applications to PRINCE and PRIDE"

2.3.4 *Meet-in-the-Middle (MitM) Attacks*

The basic MitM attack is a technique presented by Diffie and Hellman to analyze the security of the DES [DH77]. The main idea behind this technique is to decompose the encryption $\mathtt{Enc}_K$ into to smaller functions namely $\mathtt{Enc}_{1\ K_1}$ and $\mathtt{Enc}_{2\ K_2}$ such that $\mathtt{Enc}_K = \mathtt{Enc}_{2\ K_2} \circ \mathtt{Enc}_{1\ K_1}$. Then, for a given plaintext/ciphertext pair (P, C), since $C = \mathtt{Enc}_K(P)$, then it is clear that $\mathtt{Enc}_{1\ K_1}(P) = \mathtt{Dec}_{2\ K_2}(C)$. Usually, the equation's left side is called *the forward direction*, and its right side is called *the backward direction*. Therefore, the attacker can compute $X_1(K_1) := \mathtt{Enc}_{1\ K_1}(P)$ for all possible K_1 values and save the computed values in a table, and one more time, he computes $X_2(K_2) := \mathtt{Enc}_{2\ K_2}(C)$ for all possible K_2 values and check if there is a matching value in the table.

[DH77] Diffie and Hellman, "Special Feature Exhaustive Cryptanalysis of the NBS Data Encryption Standard"

The computational complexity of this attack is $2^{\kappa_1} + 2^{\kappa_2}$ where κ_1 and κ_2 are the effective bit length of K_1 and K_2, resp. In the case that $\kappa_1, \kappa_2 < \kappa$, the time complexity of the MitM attack is less than the time complexity of the exhaustive search. On the other hand, the MitM attack needs $2^{\min(\kappa_1,\kappa_2)}$ blocks of memory, which in the brute force attack it is not necessary.

Example 11. Assume the Double-DES encryption which is two iterative DES encryptions, i. e., $\mathtt{Enc}^*_{K_1,K_2} = \mathtt{Enc}_{K_2} \circ \mathtt{Enc}_{K_1}$. Using the basic MitM attack is possible to recover the 112-bit key (K_1, K_2) with only $2 \cdot 2^{56}$ DES encryptions and using 2^{56} memory blocks.

Many extension has been developed since Diffie and Hellman proposed the MITM attacks, making MitM attacks more effective or allowing them to be applicable in more situations, while the basic variant cannot. In the following, we briefly explain some of them.

PARTIAL MATCHING: It is an extension of the MitM in which the attacker uses a Boolean function $F : \mathbb{F}_2^n \mapsto \mathbb{F}_2^{n'}$ with $n' < n$ and defines $\mathtt{Enc}'_{1\ K'_1} = F \circ \mathtt{Enc}_{1\ K_1}$ and $\mathtt{Dec}'_{2\ K'_2} := F \circ \mathtt{Dec}_{2\ K_2}$.[16] Then, for a given (P, C) pair in the attack, he checks if there is a matching using $\mathtt{Enc}'_{1\ K'_1}(P) = \mathtt{Dec}'_{2\ K'_2}(C)$. If the effective size for K'_1 or K'_2 is smaller than that of for K_1 or K_2, resp., then the computation complexity of the MitM will be improved.

[16] It is usually a simple linear function.

3-SUBSET MITM: For the first time, in [BR11], Bogdanov and Rechberger introduced the 3-subset MitM attack to cryptanalyze KTANTAN [DDK09] block cipher. The idea behind this technique is to reduce the memory complexity

[BR11] Bogdanov and Rechberger, "A 3-Subset Meet-in-the-Middle Attack: Cryptanalysis of the Lightweight Block Cipher KTANTAN"

of the partial matching MitM by taking the benefit of the mutual information between K_1' and K_2'. Considering that K_c is the mutual information of both keys (e. g., common bits), this technique reduces the memory complexity of the attack by a factor of 2^{κ_c} that κ_c is the bit length of K_c.

Initial Structure and *Splice-and-Cut* by Sasaki and Aoki [SA09b], *Sieve-in-the-Middle* by Canteaut, Naya-Plasencia, and Vayssière [CNV13], and *Multidimensional MitM* by Zhu and Gong [ZG11] are the other extensions for the MitM attack that we mention without explaining the details.

[SA09b] Sasaki and Aoki "Meet-in-the-Middle Preimage Attacks on Double-Branch Hash Functions: Application to RIPEMD and Others"

[CNV13] Canteaut, Naya-Plasencia, and Vayssière "Sieve-in-the-Middle: Improved MITM Attacks"

[ZG11] Zhu and Gong *Multidimensional Meet-in-the-Middle Attack and Its Applications to KATAN32/48/64*

2.3.5 *Differential Attack*

Differential cryptanalysis was first introduced by Biham and Shamir [BS91]. Although the first target of Biham and Shamir was the reduced-round DES, it turned out to be a powerful technique, and nowadays, it is one of the important statistical cryptanalysis in symmetric cryptography. Hence, it is expected that the designers of new ciphers provide strong and sufficient arguments about the resistance of the new design against differential cryptanalysis.

[BS91] Biham and Shamir "Differential Cryptanalysis of DES-like Cryptosystems"

In differential cryptanalysis, the attacker wants to find non-uniformity in the occurrence of differences[17] between two different plaintexts P and P' (denoted by ΔP) and differences in their corresponding ciphertexts C and C' (denoted by ΔC).[18] By using this non-uniformity, which is usually for a specific ΔP and ΔC with high occurrence probability, the attacker can distinguish the cipher from a family of random permutation. The details of this attack are explained in the following.

[17] In fact, the "difference" can be defined with any group operation on the message space. However, since in this thesis, the focus is on the XOR key addition, only XOR differences are considered.

[18] Here, for simplicity, only the distinguishing attack is considered. In the key recovery attack, either P or C or both are considered to be an intermediate state.

Definition 12 (Differential Probability over a Boolean Function)**.** Let $F : \mathbb{F}_2^{n_1} \to \mathbb{F}_2^{n_2}$ be a vectorial Boolean function. For any $\alpha \in \mathbb{F}_2^{n_1}$ and any $\beta \in \mathbb{F}_2^{n_2}$, the probability of the differential (α, β) over F is defined as

$$\mathrm{Prob}(\alpha \xrightarrow{F} \beta) := 2^{-n_1} \cdot \left| \left\{ X \in \mathbb{F}_2^{n_1} \mid F(X) \oplus F(X \oplus \alpha) = \beta \right\} \right|.$$

Note that the probability is described over uniformly distributed variable $X \in \mathbb{F}_2^{n_1}$.

Thus, for an encryption $\mathtt{Enc}_K$, for any $\alpha, \beta \in \mathbb{F}_2^n$, we have

$$\mathrm{Prob}(\alpha \xrightarrow{\mathtt{Enc}_K} \beta) = 2^{-n} \cdot \left| \left\{ P \in \mathbb{F}_2^n \mid \mathtt{Enc}_K(P) \oplus \mathtt{Enc}_K(P \oplus \alpha) = \beta \right\} \right|$$

which is clearly a key dependent value. This probability is usually called the *fixed-key differential probability*.

Since the attacker has no knowledge of the actual key initiated in $\mathtt{Enc}_K$, he wants to find a differential for which the corresponding fixed-key differential probabilities are significantly higher than 2^{-n} for almost all the keys in $\mathbb{F}_2^\kappa$. Thus, he wants to find a differential that holds with a high *expected differential probability*.

Definition 13 (Expected Differential Probability (EDP) of a Block Cipher)**.** For any α and $\beta \in \mathbb{F}_2^n$, EDP of differential (α, β) over encryption $\mathtt{Enc}$ is defined as

$$\begin{aligned} \mathrm{EDP}(\alpha \xrightarrow{\mathtt{Enc}} \beta) :=& 2^{-\kappa} \cdot \sum_{K \in \mathbb{F}_2^\kappa} \mathrm{Prob}(\alpha \xrightarrow{\mathtt{Enc}_K} \beta) \\ =& 2^{-(n+\kappa)} \cdot \sum_{K \in \mathbb{F}_2^\kappa} \left| \left\{ P \in \mathbb{F}_2^n \mid \mathtt{Enc}_K(P) \oplus \mathtt{Enc}_K(P \oplus \alpha) = \beta \right\} \right| \end{aligned}$$

which is described over uniformly distributed random key $K \in \mathbb{F}_2^\kappa$.

Lai, Massey, and Murphy [LMM91] introduced the *Hypothesis of Stochastic Equivalence*, which assumes that the fixed-key probability of a differential is to a large extent independent of the actual key used.

[LMM91] Lai, Massey, and Murphy "Markov Ciphers and Differential Cryptanalysis"

Assumption 14 (Stochastic Equivalence for Differential Probabilities)**.** *For the given block cipher* Enc *and any differential* (α, β)*, for "almost" all of the keys* $K \in \mathbb{F}_2^\kappa$

$$\mathrm{EDP}(\alpha \xrightarrow{\mathtt{Enc}} \beta) \approx \mathrm{Prob}(\alpha \xrightarrow{\mathtt{Enc}_K} \beta).$$

Under Assumption 14, by finding a differential (α, β) over Enc that holds with high expected differential probability, i. e., $\mathrm{EDP}(\alpha \xrightarrow{\mathtt{Enc}} \beta) \gg 2^{-n}$, the attacker can now distinguish the encryption from a random permutation. By querying the oracle for *enough* randomly chosen[19,20] input pairs with difference α and checking if the output difference equals β as often as expected by the probability, he decides whether the oracle is the encryption or a random permutation.

[19] Note that the differential cryptanalysis is an example of a Chosen Plaintext Attack (CPA).

[20] Biham and Shamir indicate that about $c \cdot \left(\mathrm{EDP}(\alpha \xrightarrow{\mathtt{Enc}} \beta)\right)^{-1}$ differential pairs are enough to distinguish with a reasonable high advantage, where c is some small constant.

In the following, the focus is on the differential probability of the key-alternating cipher.

Differential Probability of a Key-Alternating Cipher

For a key-alternating cipher, besides of only considering the plaintext and ciphertext difference, one can consider the difference in all of the intermediate states between the rounds. This leads us to the following definition.

Definition 15 (Differential Characteristics)**.** An r-round *differential characteristic*[21] $(\alpha_0, \ldots, \alpha_r)$ is a vector of $r+1$ elements in $\mathbb{F}_2^n$. For a key-alternating cipher $\mathtt{Enc}_K = \mathtt{A}_{K_r} \circ \mathtt{R}^*_{r-1} \circ \mathtt{A}_{K_{r-1}} \circ \ldots \circ \mathtt{R}^*_1 \circ \mathtt{A}_{K_1} \circ \mathtt{R}^*_0 \circ \mathtt{A}_{K_0}$ with $(K_0, K_1, \ldots, K_r) = \mathtt{KeySch}(K)$, the probability of such a differential characteristic is defined by

[21] It is also called *differential trail*.

$$\mathrm{Prob}(\alpha_0 \xrightarrow{\mathtt{R}_0} \alpha_1 \to \ldots \to \alpha_{r-1} \xrightarrow{\mathtt{R}_{r-1}} \alpha_r) = 2^{-n} \cdot \Big|\{P \in \mathbb{F}_2^n \mid \forall i \in \mathbb{Z}_r,$$
$$\mathtt{R}^*_i \circ \mathtt{A}_{K_i} \circ \ldots \circ \mathtt{R}^*_0 \circ \mathtt{A}_{K_0}(P) \oplus \mathtt{R}^*_i \circ \mathtt{A}_{K_i} \circ \ldots \circ \mathtt{R}^*_0 \circ \mathtt{A}_{K_0}(P \oplus \alpha_0) = \alpha_{i+1}\}\Big|$$

which depends on the value of key K. In a similar way, we define the EDP of the differential characteristic by

$$\mathrm{EDP}(\alpha_0 \xrightarrow{\mathtt{R}_0} \alpha_1 \to \ldots \to \alpha_{r-1} \xrightarrow{\mathtt{R}_{r-1}} \alpha_r) = 2^{-k} \cdot \sum_{K \in \mathbb{F}_2^\kappa} \mathrm{Prob}(\alpha_0 \xrightarrow{\mathtt{R}_0} \ldots \xrightarrow{\mathtt{R}_{r-1}} \alpha_r).$$

It follows that for a key-alternating cipher, the probability of a differential is the sum of the probabilities of all containing characteristics whose plaintext and ciphertext differences are the same as in the differential, i. e.,

$$\mathrm{Prob}(\alpha_0 \xrightarrow{\mathtt{Enc}_K} \alpha_r) = \sum_{\alpha_1, \ldots, \alpha_{r-1}} \mathrm{Prob}(\alpha_0 \xrightarrow{\mathtt{R}_0} \alpha_1 \to \ldots \to \alpha_{r-1} \xrightarrow{\mathtt{R}_{r-1}} \alpha_r).$$

Computing these probabilities is not an easy task; since for a given differential (characteristic), they need computation over all the message and the key space, i. e., the computation complexity is $\mathcal{O}(2^{n+\kappa})$. Even for the smallest applied values of n and κ, this is an unapproachable complexity.

Therefore, it comes to question how one can efficiently compute the differential (characteristic) probability. One assumption that makes it easier to compute EDP of differential characteristics is the *Independent Round Keys* assumption.[22]

[22] In [LMM91], the authors used the term *Markov Cipher* assumption. Since the only way to reach Markov cipher assumption in the key-alternating ciphers is to have independent round keys, only the latter term is used.

Assumption 16 (Independent Round Keys). *For a key-alternating cipher* $\mathtt{Enc}_K = \mathtt{A}_{K_r} \circ \mathtt{R}^*_{r-1} \circ \mathtt{A}_{K_{r-1}} \circ \ldots \circ \mathtt{R}^*_0 \circ \mathtt{A}_{K_0}$*, it is assumed that the all* $r+1$ *round keys are independent of each other, i. e., simply it is assumed that* $K = \phi_{n,r+1}(K_0, \ldots, K_r)$.

Theorem 17. *In a key-alternating cipher* $\mathtt{Enc}_K = \mathtt{A}_{K_r} \circ \mathtt{R}^*_{r-1} \circ \mathtt{A}_{K_{r-1}} \circ \ldots \circ \mathtt{R}^*_0 \circ \mathtt{A}_{K_0}$ *with* $r+1$ *independent round keys* $K_0, \ldots, K_r$*:*

$$\mathrm{EDP}(\alpha_0 \xrightarrow{\mathtt{R}_0} \alpha_1 \to \ldots \to \alpha_{r-1} \xrightarrow{\mathtt{R}_{r-1}} \alpha_r) = \prod_{i=0}^{r-1} \mathrm{Prob}(\alpha_i \xrightarrow{\mathtt{R}^*_i} \alpha_{i+1}), \tag{2.1}$$

$$\mathrm{EDP}(\alpha_0 \xrightarrow{\mathtt{Enc}} \alpha_r) = \sum_{\alpha_1,\ldots,\alpha_{r-1}} \prod_{i=0}^{r-1} \mathrm{Prob}(\alpha_i \xrightarrow{\mathtt{R}^*_i} \alpha_{i+1}). \tag{2.2}$$

Since this theorem has an essential role in providing arguments about the resistance of key-alternating ciphers against the differential attack, and it is one of the concepts used in Chapter 5, we are going to prove it.

Proof. Considering X_i as the input of $\mathtt{R}_i$ for plaintext P, (i. e., $X_0 = P$ and $X_{i+1} = \mathtt{R}^*_i \circ \mathtt{A}_{K_i}(X_i)$ for each $0 < i \leq r$), by definition, the EDP of such a differential characteristic is equal to

$$2^{-n\cdot(r+2)} \cdot \left| \left\{ (P, K_0, \ldots, K_r) \in (\mathbb{F}_2^n)^{r+2} \mid \forall i \; \mathtt{R}^*_i \circ \mathtt{A}_{K_i}(X_i) \oplus \mathtt{R}^*_i \circ \mathtt{A}_{K_i}(X_i \oplus \alpha_i) = \alpha_{i+1} \right\} \right| =$$
$$2^{-n\cdot(r+2)} \cdot \left| \left\{ (P, K_0, \ldots, K_r) \in (\mathbb{F}_2^n)^{r+2} \mid \forall i \; \mathtt{R}^*_i(X_i \oplus K_i) \oplus \mathtt{R}^*_i(X_i \oplus K_i \oplus \alpha_i) = \alpha_{i+1} \right\} \right|.$$

[23] Notice that since all K_i are independent of each other, each $K'_i (= K_i \oplus X_i)$ is independent of X_i and also of K'_js with $j < i$.

By replacing K_i with $K'_i \oplus X_i$ (which is a bijective mapping)[23] for each $i \in \mathbb{Z}_r$, the above equation will be simplified to

$$2^{-n\cdot(r+2)} \cdot \left| \left\{ (P, K'_0, \ldots, K'_{r-1}, K_r) \in (\mathbb{F}_2^n)^{r+2} \mid \forall i \; \mathtt{R}^*_i(K'_i) \oplus \mathtt{R}^*_i(K'_i \oplus \alpha_i) = \alpha_{i+1} \right\} \right|,$$

that because of independence of K'_i values, K_r and of P, it is equal to

$$2^{-n\cdot(r+2)} \cdot \left|\mathbb{F}_2^n\right|^2 \cdot \prod_{i=0}^{r-1} \left| \left\{ K'_i \in \mathbb{F}_2^n \mid \mathtt{R}^*_i(K'_i) \oplus \mathtt{R}^*_i(K'_i \oplus \alpha_i) = \alpha_{i+1} \right\} \right| =$$
$$\prod_{i=0}^{r-1} 2^{-n} \cdot \left| \left\{ K'_i \in \mathbb{F}_2^n \mid \mathtt{R}^*_i(K'_i) \oplus \mathtt{R}^*_i(K'_i \oplus \alpha_i) = \alpha_{i+1} \right\} \right| =$$
$$\prod_{i=0}^{r-1} \mathrm{Prob}(\alpha_i \xrightarrow{\mathtt{R}^*_i} \alpha_{i+1}).$$

Furthermore, using Eq. (2.1), it is easy to reach Eq. (2.2). □

Theorem 17 shows that the independent round keys assumption makes the computation much easier, especially if the round functions are quite simple ones. In differential cryptanalysis of key-alternating ciphers, it is common to assume that all the round keys are independent. Then, EDP of a differential (characteristic) can be computed as given in Eqs. (2.1) and (2.2). This assumption allows computing EDP without considering the value of the master key. Using this assumption, it is easy to compute EDP

of a differential characteristic. However, computing EDP of a differential using this assumption remains challenging since it requires considering all differential characteristics with different $\alpha_1, \ldots, \alpha_{r-1}$.

Differential Probability of an SPN Cipher

Recalling that in an SPN cipher $\mathrm{R}_i^* = \mathrm{L}_i \circ \mathrm{S}_i$ with S_i being a parallel application of smaller bijective S-boxes and L_i being a bijective linear mapping, using the following lemmas, it simplifies the computation of the EDP considering Assumption 16.

Lemma 18. *Let $\mathrm{L} : \mathbb{F}_2^{n_1} \to \mathbb{F}_2^{n_2}$ be a linear mapping.*[24] *Then for any $\alpha \in \mathbb{F}_2^{n_1}$ and $\beta \in \mathbb{F}_2^{n_2}$, we have*

$$\mathrm{Prob}(\alpha \xrightarrow{\mathrm{L}} \beta) = \begin{cases} 1 & \text{if } \beta = \mathrm{L}(\alpha), \\ 0 & \text{otherwise.} \end{cases}$$

[24] Note that this is the general case for all linear mappings, while the linear mappings in SPN, all are bijections.

Lemma 19. *Let $\mathrm{S} : \mathbb{F}_2^{s \cdot m} \to \mathbb{F}_2^{s \cdot m}$ be the parallel application of m bijective S-boxes $S_j : \mathbb{F}_2^s \to \mathbb{F}_2^s$ with $j \in \mathbb{Z}_m$ such as defined in Definition 6. Then for any $\alpha, \beta \in (\mathbb{F}_2^s)^m$, we have*

$$\mathrm{Prob}\left(\phi_{s,m}(\alpha) \xrightarrow{\mathrm{S}} \phi_{s,m}(\beta)\right) = \prod_{j=0}^{m-1} \mathrm{Prob}(\alpha[j] \xrightarrow{S_j} \beta[j]).$$

Proposition 20. *Let $\mathtt{Enc}$ be the r-round SPN block cipher as defined in Definition 6 with $r+1$ independent round keys $K_0, \ldots, K_r$. Then the EDP of the characteristic $(\alpha_0, \ldots, \alpha_r)$ is equal to*

$$\mathrm{EDP}(\alpha_0 \xrightarrow{\mathrm{R}_0} \alpha_1 \to \ldots \to \alpha_{r-1} \xrightarrow{\mathrm{R}_{r-1}} \alpha_r) = \prod_{i=0}^{r-1} \prod_{j=0}^{m-1} \mathrm{Prob}\left(\phi_{s,m}^{-1}(\alpha_i)[j] \xrightarrow{S_{ij}} \phi_{s,m}^{-1}(\beta_i)[j]\right), \quad (2.3)$$

where $\beta_i := \mathrm{L}_i^{-1}(\alpha_{i+1})$ with $i \in \mathbb{Z}_r$. Furthermore,

$$\mathrm{EDP}(\alpha_0 \xrightarrow{\mathtt{Enc}} \alpha_r) = \sum_{\alpha_1, \ldots, \alpha_{r-1}} \prod_{i=0}^{r-1} \prod_{j=0}^{m-1} \mathrm{Prob}\left(\phi_{s,m}^{-1}(\alpha_i)[j] \xrightarrow{S_{ij}} \phi_{s,m}^{-1}(\beta_i)[j]\right). \quad (2.4)$$

Proof. To prove Eqs. (2.3) and (2.4) it is enough to show that for one round of an SPN cipher, we have

$$\mathrm{Prob}(\alpha_i \xrightarrow{\mathrm{R}_i^*} \alpha_{i+1}) = \prod_{j=0}^{m-1} \mathrm{Prob}\left(\phi_{s,m}^{-1}(\alpha_i)[j] \xrightarrow{S_{ij}} \phi_{s,m}^{-1}(\beta_i)[j]\right).$$

By definition, we have

$$\begin{aligned} \mathrm{Prob}(\alpha_i \xrightarrow{\mathrm{R}_i^*} \alpha_{i+1}) &= 2^{-n} \cdot \left| \left\{ X \in \mathbb{F}_2^n \mid \mathrm{R}_i(X) \oplus \mathrm{R}_i(X \oplus \alpha_i) = \alpha_{i+1} \right\} \right| \\ &= 2^{-n} \cdot \left| \left\{ X \in \mathbb{F}_2^n \mid \mathrm{L}_i \circ \mathrm{S}_i(X) \oplus \mathrm{L}_i \circ \mathrm{S}_i(X \oplus \alpha_i) = \alpha_{i+1} \right\} \right|. \end{aligned}$$

Since, in an SPN, L_i is a linear bijection, the last equation simplifies to

$$\mathrm{Prob}(\alpha_i \xrightarrow{\mathrm{R}_i^*} \alpha_{i+1}) = 2^{-n} \cdot \left| \left\{ X \in \mathbb{F}_2^n \mid \mathrm{S}_i(X) \oplus \mathrm{S}_i(X \oplus \alpha_i) = \beta_i \right\} \right|$$

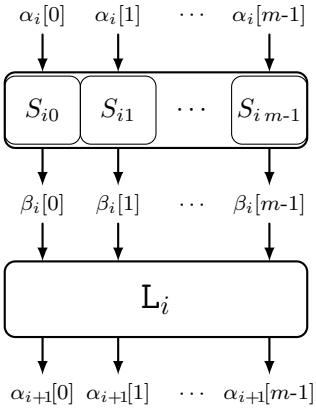


FIGURE 2.5: Differential Transitions over an SPN Round. Note that the given values are the differential vlaue for the corresponding state.

where $\beta_i := \mathrm{L}_i^{-1}(\alpha_{i+1})$. From other side, using Lemma 19, we have

$$\mathrm{Prob}(\alpha_i \xrightarrow{\mathrm{R}_i^s} \alpha_{i+1}) = \mathrm{Prob}(\alpha_i \xrightarrow{\mathrm{S}_i} \beta_i) = \prod_{j=0}^{m-1} \mathrm{Prob}(\alpha_i[j] \xrightarrow{S_{ij}} \beta_i[j]),$$

and this proves the proposition. □

The differential transitions over one round of an SPN cipher is shown in Figure 2.5. In the following, some more definitions regarding the differential properties of SPN block ciphers are brought that will be necessary for the rest of the thesis.

Definition 21 (Difference Distribution Table (DDT))**.** Let $F : \mathbb{F}_2^{n_1} \to \mathbb{F}_2^{n_2}$ be a vectorial Boolean function. The DDT of F is a table whose (α, β)-th element, i. e., $\mathrm{DDT}[\alpha][\beta]$, with $\alpha \in \mathbb{F}_2^{n_1}$ and $\beta \in \mathbb{F}_2^{n_2}$ shows the number of $X \in \mathbb{F}_2^{n_1}$ such that $F(X) \oplus F(X \oplus \alpha) = \beta$, i. e., $\left| \left\{ X \in \mathbb{F}_2^{n_1} \mid F(X) \oplus F(X \oplus \alpha) = \beta \right\} \right|$.

Definition 22 (Differential Uniformity)**.** For the vectorial Boolean function $F : \mathbb{F}_2^{n_1} \to \mathbb{F}_2^{n_2}$, *differential uniformity* or simply *uniformity* is defined as

$$\mathrm{uni}(F) = \max_{\substack{\alpha \in \mathbb{F}_2^{n_1} \setminus \{0\}, \\ \beta \in \mathbb{F}_2^{n_2}}} \left| \{ X \in \mathbb{F}_2^{n_1} \mid F(x) + F(X + \alpha) = \beta \} \right|.$$

It is noteworthy to mention that in an s-bit S-box, it is always $\mathrm{DDT}[0][0] = 2^s$ and if it is a bijective S-box, then for any non-zero $\alpha \in \mathbb{F}_2^s$, $\mathrm{DDT}[0][\alpha] = \mathrm{DDT}[\alpha][0] = 0$. This means that input difference of a bijective S-box is zero if and only if its output difference is zero. In a differential characteristic, an S-box is called *active* if its input (or output) difference is non-zero. $\delta(X)$ is used to denote the *activity pattern* in X which is defined as below.

Definition 23 (Activity Function)**.** The function $\delta : \mathbb{F}_2^{s \cdot m} \to \mathbb{F}_2^m$ defined as

$$\delta(X)[i] = \begin{cases} 1 & \text{if } \phi_{s,m}^{-1}(X)[i] \neq 0 \\ 0 & \text{if } \phi_{s,m}^{-1}(X)[i] = 0 \end{cases}$$

is called *activity function*; i. e., the i-th bit of $\delta(X)$ is 1 if and only if the i-th word of X is non-zero.

Since in our case, the S-box is considered to be a bijection, the activity pattern of differences in the input of the S-box layer is the same as the one in the output of the S-box layer: $\delta(\Delta X_i) = \delta(\Delta Y_i)$ with $i \in \mathbb{Z}_r$. This pattern is denoted by δ_i and we use $(\delta_0, \ldots, \delta_{r-1})$ to show the activity pattern of the characteristic.

Using DDT of each S-box S_{ij}, it is possible to compute the EDP of a characteristic (Eq. (2.3)) with less than $m \cdot r$ table look-ups.[25] However, computing the EDP of a differential (Eq. (2.4)) requires computing the probability of all underlying differential characteristics with different $\alpha_1, \ldots, \alpha_{r-1}$.

[25] It is enough to only consider the active S-boxes of the characteristic.

Related-Tweak Probabilities

For related-tweak adversary models, we denote the EDP of a differential by $\mathrm{EDP}(\alpha \xrightarrow{\mathrm{Enc}} \beta \mid \Delta T = \gamma)$ and it is defined as

$$2^{-n-t-\kappa} \cdot \left| \left\{ K \in \mathbb{F}_2^\kappa, T \in \mathbb{F}_2^t, P \in \mathbb{F}_2^n \mid \mathrm{Enc}_K^T(P) \oplus \mathrm{Enc}_K^{T \oplus \gamma}(P \oplus \alpha) = \beta \right\} \right|.$$

Similarly, in a key-alternating cipher with tweakey framework, EDP of a differential characteristic is denoted by $\mathrm{EDP}(\alpha_0 \xrightarrow{\mathtt{R}_0} \alpha_1 \to \ldots \to \alpha_{r-1} \xrightarrow{\mathtt{R}_{r-1}} \alpha_r \mid \Delta T = \gamma)$ that it is defined as

$$2^{-n-t-\kappa} \cdot \Big| \{K \in \mathbb{F}_2^\kappa, T \in \mathbb{F}_2^t, P \in \mathbb{F}_2^n \mid \forall i \in \mathbb{Z}_r, \\ \mathtt{R}_i^* \circ \mathtt{A}_{TK_i} \circ \ldots \circ \mathtt{R}_0^* \circ \mathtt{A}_{TK_0}(P) \oplus \mathtt{R}_i^* \circ \mathtt{A}_{TK_i'} \circ \ldots \circ \mathtt{R}_0^* \circ \mathtt{A}_{TK_0'}(P \oplus \alpha_0) = \alpha_{i+1}\} \Big|.$$

where TK_i values are the round tweakeys derived from key K and tweak T, while TK_i' are the ones from key K and tweak $T \oplus \gamma$.

In the case that the tweak schedule is linear and independent of the key schedule, and each round tweakey is the XOR of round key and round tweak,[26] it is possible to modify Theorem 17 to use it for the related-tweak models. This is explained in more detail in the Proposition 24 and since the proof is similar to the proof of Theorem 17, we do not bring it here.

[26] Note that it is possible to consider a more general case than the one we assumed here. But since the general cases are not applied in the thesis, we concentrate on this simplified case.

Proposition 24. *In a key-alternating tweakable block cipher* $\mathtt{Enc}_K^T = \mathtt{A}_{K_r} \circ \mathtt{R}_{r-1}^* \circ \mathtt{A}_{T_{r-1} \oplus K_{r-1}} \circ \ldots \circ \mathtt{R}_0^* \circ \mathtt{A}_{T_0 \oplus K_0}$ *with* $r+1$ *independent round keys* $K_0, \ldots, K_r$ *and a linear tweak schedule* $(T_0, \ldots, T_{r-1}) = \mathtt{TwkSch}(T)$:[27]

$$\mathrm{EDP}(\alpha_0 \xrightarrow{\mathtt{R}_0} \alpha_1 \to \ldots \to \alpha_{r-1} \xrightarrow{\mathtt{R}_{r-1}} \alpha_r \mid \Delta T = \gamma) = \prod_{i=0}^{r-1} \mathrm{Prob}(\alpha_i \oplus \gamma_i \xrightarrow{\mathtt{R}_i^*} \alpha_{i+1}), \tag{2.5}$$

$$\mathrm{EDP}(\alpha_0 \xrightarrow{\mathtt{Enc}} \alpha_r \mid \Delta T = \gamma) = \sum_{\alpha_1, \ldots, \alpha_{r-1}} \prod_{i=0}^{r-1} \mathrm{Prob}(\alpha_i \oplus \gamma_i \xrightarrow{\mathtt{R}_i^*} \alpha_{i+1}) \tag{2.6}$$

where $(\gamma_0, \ldots, \gamma_{r-1}) = \mathtt{TwkSch}(\gamma)$.

[27] Recalling that each round $\mathtt{R}_i$ starts with tweakey addition and follows by the permutation $\mathtt{R}_i^*$. Besides, without loss of generality and just for simplicity, we considered that in the last post-whitening key, there is no tweak added.

Extensions and Generalizations

Several variants of differential cryptanalysis have been proposed. Here, only three of the most known variants are explained that will be necessary for this thesis: *truncated differentials*, *higher-order differentials*, and *impossible differentials*.

TRUNCATED DIFFERENTIAL: It is a generalization of the differential analysis that is introduced by Knudsen [Knu95]. In a truncated differential, the plaintext difference can be chosen from a set $\mathcal{U}$ and the ciphertext difference from a set $\mathcal{V}$. Its probability is defined as follows,

[Knu95] Knudsen "Truncated and Higher Order Differentials"

$$\mathrm{EDP}(\mathcal{U} \xrightarrow{\mathtt{Enc}} \mathcal{V}) = \frac{1}{|\mathcal{U}|} \cdot \sum_{\substack{\alpha \in \mathcal{U} \\ \beta \in \mathcal{V}}} \mathrm{EDP}(\alpha_0 \xrightarrow{\mathtt{Enc}} \alpha_r)$$

which is based on the assumption that all the differences in the set $\mathcal{U}$ occur uniformly.

Note that in classical truncated differentials, only the differences in some bits of the plaintext and ciphertext are determined, while the other bits can take any values. In this case, $\mathcal{U}$ and $\mathcal{V}$ are a special kind of linear subspaces excluding the zero value. In this thesis, the general truncated differential definition is used, and it will be explicitly mentioned in the case that the sets are linear subspaces.

If attacker can find sets $\mathcal{U}$ and $\mathcal{V}$ sets such that $\mathrm{EDP}(\mathcal{U} \xrightarrow{\texttt{Enc}} \mathcal{V})$ is significantly larger than $|\mathcal{V}| \cdot 2^{-n}$, then he can use this truncated differential to distinguish the Enc from a random permutation. We define an extra notation for the truncated differential, named *Expected Differential Distinguishability (EDD)* which is the average of the differential probability for all differentials in the truncated differential, i. e.,

$$\mathrm{EDD}(\mathcal{U} \xrightarrow{\texttt{Enc}} \mathcal{V}) = \frac{1}{|\mathcal{U}| \cdot |\mathcal{V}|} \cdot \sum_{\substack{\alpha \in \mathcal{U} \\ \beta \in \mathcal{V}}} \mathrm{EDP}(\alpha \xrightarrow{\texttt{Enc}} \beta).$$

A truncated differential can be useful for distinguishing if the corresponding EDD value is significantly larger than 2^{-n}.

HIGHER-ORDER DIFFERENTIAL: While the basic differential applies properties between only two plaintext/ciphertext pairs, the *higher-order differential* studies the differential properties of a larger set of plaintext/ciphertext pairs in which the set of these plaintexts forms an affine subspace.

In 1994, Lai [Lai94] used the notion of a higher-order differential, explained its concept, and showed that the differentials are a particular case of higher-order derivatives. Later in the same year, Knudsen [Knu95] applied the concept of the higher-order derivatives for attacking block ciphers.

[Lai94] Lai "Higher Order Derivatives and Differential Cryptanalysis"

[Knu95] Knudsen "Truncated and Higher Order Differentials"

Even though the higher-order differential attack is superior to the typical differential attack, in practice, due to the lack of appropriate tools, it is impossible to study the higher-order differential behavior of the ciphers. Therefore, usually, only the deterministic higher-order differentials are applied in the attacks.[28]

[28] In other meaning, the corresponding differential probability is equal to 1 for any key value.

A notable example for higher-order differential cryptanalysis is breaking the block cipher KN-cipher by Jakobsen and Knudsen [JK97], which previously it was proven by the designers that the cipher is immune against standard differential cryptanalysis [NK95].

[JK97] Jakobsen and Knudsen "The Interpolation Attack on Block Ciphers"

[NK95] Nyberg and Knudsen, "Provable Security Against a Differential Attack"

IMPOSSIBLE DIFFERENTIAL: Knudsen was the first one who applied this technique to analyze his new design, the block cipher DEAL [Knu98]. Later, Biham, Biryukov, and Shamir [BBS99a] introduced the name *impossible differential* and used the technique to analyze the block cipher SKIPJACK [NIS98]. The same authors presented an efficient method for finding impossible differentials that they called *miss-in-the-middle* attack Biham, Biryukov, and Shamir [BBS99b].

[Knu98] Knudsen, "DEAL - A 128-bit Block Cipher"

[BBS99a] Biham, Biryukov, and Shamir "Cryptanalysis of Skipjack Reduced to 31 Rounds Using Impossible Differentials"

[NIS98] NIST, "SKIPJACK and KEA Algorithm Specifications"

[BBS99b] Biham, Biryukov, and Shamir "Miss in the Middle Attacks on IDEA and Khufu"

A differential (α, β) is called an *impossible differential* over a (reduced-round) encryption Enc if for all key values K, $\mathrm{Prob}(\alpha \xrightarrow{\mathrm{Enc}_K} \beta) = 0$. Such a distinguisher over a reduced-round version of the cipher might be used for a key-recovery attack over a larger number of rounds by filtering all the key candidates which leads to the intermediate state values with differences α and β, i. e., the intermediate state values fulfilling the impossible differential.

2.3.6 *Linear Cryptanalysis*

The linear cryptanalysis was discovered by Matsui, who first applied the technique to the FEAL block cipher in [MY93]. Eventually, Matsui applied the technique to the DES [Mat94], and later, he published an attack on the

[MY93] Matsui and Yamagishi, "A New Method for Known Plaintext Attack of FEAL Cipher"

[Mat94] Matsui, "Linear Cryptanalysis Method for DES Cipher"

DES, which was the first reported experimental cryptanalysis of the cipher [Mat95].

[Mat95] Matsui, "On Correlation Between the Order of S-boxes and the Strength of DES"

The general idea of this attack is to approximate a linear relation between bits of the plaintext, ciphertext, and the master key [MY93; Mat94]. More precisely, the attacker is interested in the correlation of $\langle \alpha, P \rangle \oplus \langle \beta, \mathtt{Enc}_K(P) \rangle \oplus \langle \gamma, K \rangle$, where $\alpha, \beta \in \mathbb{F}_2^n$ and $\gamma \in \mathbb{F}_2^\kappa$ are called *linear masks* of the plaintext, ciphertext, and the key, resp.[29] The strength of a linear approximation is measured by its correlation. By exploiting a linear approximation for which the approximation holds with a high absolute correlation,[30] the attacker can distinguish the cipher from a family of random permutation. In the following, the most important concepts of this attack are explained.

[29] Here again similar to the differential attack, for simplicity, only the distinguishing attack is considered.

[30] That means, it holds for many values of P and K or only for few values.

Definition 25 (Correlation of a Boolean Function)**.** For a Boolean function $f : \mathbb{F}_2^n \to \mathbb{F}_2$, its *correlation* is defined as

$$c_f = 2^{-n} \cdot \Big(\big| \{x \in \mathbb{F}_2^n \mid f(X) = 0\} \big| - \big| \{x \in \mathbb{F}_2^n \mid f(X) = 1\} \big| \Big).$$

In some literature, instead of correlation, they used *bias* of the function (denoted by ϵ_f) which fulfills $c_f = 2 \cdot \epsilon_f$. In this thesis, only the correlation is used.

Definition 26 (Linear Approximation of a Vectorial Boolean Function)**.** Let $F : \mathbb{F}_2^{n_1} \to \mathbb{F}_2^{n_2}$ be a vectorial Boolean function. For any $\alpha \in \mathbb{F}_2^{n_1}$ and any $\beta \in \mathbb{F}_2^{n_2}$, $\langle \alpha, X \rangle \oplus \langle \beta, F(x) \rangle$ is a linear approximation of F and $\mathrm{Cor}(\alpha \xrightarrow{F} \beta)$ is used to denote $c_{\langle \alpha, x \rangle \oplus \langle \beta, F(x) \rangle}$. Besides, α and β are called the *input mask* and *output mask*, resp.

The *Walsh transform* is closely related to the concept of correlation and offers a useful tool for its computation.

Definition 27 (Walsh Transform a Vectorial Boolean Function)**.** Walsh transform of a boolean function $f : \mathbb{F}_2^{n_1} \to \mathbb{F}_2^{n_2}$ is defined as

$$\hat{f}(\alpha) = \sum_{x \in \mathbb{F}_2^{n_1}} (-1)^{\langle \alpha, x \rangle \oplus f(x)}.$$

The Walsh transform of a vectorial Boolean function $F : \mathbb{F}_2^{n_1} \to \mathbb{F}_2^{n_2}$ is defined as

$$\hat{F}(\alpha, \beta) = \sum_{x \in \mathbb{F}_2^{n_1}} (-1)^{\langle \alpha, x \rangle \oplus \langle \beta, F(x) \rangle}.$$

Therefore,

$$\mathrm{Cor}(\alpha \xrightarrow{F} \beta) = 2^{-n_1} \cdot \hat{F}(\alpha, \beta).$$

If an adversary can find a linear approximation which holds with high absolute correlation, i. e., $\left| \mathrm{Cor}(\alpha \xrightarrow{\mathtt{Enc}_K} \beta) \right| > 2^{-n/2}$, for some values of K, then he can distinguish the encryption $\mathtt{Enc}_K$ from a random permutation. By querying the oracle for *enough* random known[31,32] inputs and by checking if the linear approximation occurs as often as the linear approximation expected by the correlation of the linear approximation, he decides whether the oracle is the encryption or a random permutation.

However, the correlation of a linear approximation is a key dependent value. This means that for any $\alpha, \beta \in \mathbb{F}_2^n$, $\mathrm{Cor}(\alpha \xrightarrow{\mathtt{Enc}_K} \beta)$ depends on K.

[31] Note that in contrast to differential cryptanalysis where the adversary needs to choose input pairs, the linear cryptanalysis is a Known Plaintext Attack (KPA).

[32] Matsui claimed that about $c \cdot \big(\mathrm{ELC}(\alpha \xrightarrow{\mathtt{Enc}} \beta) \big)^{-2}$ plaintexts are enough to distinguish, where c is some small constant.

[Nyb01] Nyberg "Correlation Theorems in Cryptanalysis"

However, Nyberg [Nyb01][Theorem 3] showed that the correlation of a linear approximation over $\mathtt{Enc}_K$ can be given as a signed sum of correlations of linear approximations over $\mathtt{Enc}(\cdot,\cdot)$, i. e.,

$$\mathrm{Cor}(\alpha \xrightarrow{\mathtt{Enc}_K} \beta) = \sum_{\gamma \in \mathbb{F}_2^\kappa} (-1)^{\langle \gamma, K\rangle} \mathrm{Cor}\left((\gamma,\alpha) \xrightarrow{\mathtt{Enc}} \beta\right),$$

where $\mathrm{Cor}\left((\gamma,\alpha) \xrightarrow{\mathtt{Enc}} \beta\right)$ denotes $c_{\langle \alpha, P\rangle \oplus \langle \gamma, K\rangle \oplus \langle \beta, \mathtt{Enc}_K(P)\rangle}$ and is usually called *fixed-key linear approximation correlation*. This equation shows that a linear approximation of an encryption depends on the distribution of the fixed-key linear approximation correlations of the whole key space. While this distribution is fixed for given K, the signed distribution is not. And this means that the absolute linear approximation of the encryption can be very high for some keys while can be very small to some others; i. e., the attack might work only for some keys which are usually referred to as *weak keys*. Since the attacker does not know the initiated key, he wants to find a linear approximation that the fixed-key linear approximation correlation is higher than the $2^{-n/2}$ for almost all the keys. Thus, he wants to find a linear approximation that holds with a high *expected linear approximation correlation*.

Definition 28 (Expected Linear Correlation (ELC) of a Block Cipher)**.** For any α and $\beta \in \mathbb{F}_2^n$, the ELC of linear approximation (α,β) over encryption $\mathtt{Enc}$ is defined as

$$\mathrm{ELC}(\alpha \xrightarrow{\mathtt{Enc}} \beta) := 2^{-\kappa} \cdot \sum_{K \in \mathbb{F}_2^\kappa} \mathrm{Cor}(\alpha \xrightarrow{\mathtt{Enc}_K} \beta)$$

which is described over uniformly distributed random key $K \in \mathbb{F}_2^\kappa$.

Similar to the Assumption 14, Matsui assumed that the fixed-key linear approximation correlation for almost all the keys is about the same as the ELC.

Assumption 29 (Stochastic Equivalence for Linear Approximation Correlation)**.** *For the given block cipher* $\mathtt{Enc}$ *and any linear approximation* (α,β)*, for "almost" all of the keys* $K \in \mathbb{F}_2^\kappa$

$$\mathrm{ELC}(\alpha \xrightarrow{\mathtt{Enc}} \beta) \approx \mathrm{Cor}(\alpha \xrightarrow{\mathtt{Enc}_K} \beta).$$

In the following, the focus is on the ELC of key-alternating ciphers that most of the materials are similar to the ones for differential probability.

Linear Approximation Correlation of a Key-Alternating Cipher

For a key-alternating cipher, similar to the differential characteristics, besides of only considering the plaintext and ciphertext linear masks, one can consider linear masks in all of the intermediate states between the rounds.

[33] It is also called *linear trail*.

Definition 30 (Linear Characteristics)**.** An r-round *linear characteristic*[33] $(\alpha_0,\ldots,\alpha_r)$ is a vector of $r+1$ elements in $\mathbb{F}_2^n$. For a key-alternating cipher $\mathtt{Enc}_K = \mathtt{A}_{K_r} \circ \mathtt{R}^*_{r-1} \circ \mathtt{A}_{K_{r-1}} \circ \ldots \circ \mathtt{R}^*_0 \circ \mathtt{A}_{K_0}$ with $(K_0,\ldots,K_r) = \mathtt{KeySch}(K)$, the correlation of such a linear characteristic is defined by

$$\mathrm{Cor}(\alpha_0 \xrightarrow{\mathtt{R}_0} \alpha_1 \to \ldots \to \alpha_{r-1} \xrightarrow{\mathtt{R}_{r-1}} \alpha_r) = (-1)^{\bigoplus_{i=0}^{r} \langle K_i, \alpha_i\rangle} \cdot \prod_{i=0}^{r-1} \mathrm{Cor}(\alpha_i \xrightarrow{\mathtt{R}^*_i} \alpha_{i+1})$$

This implies that the absolute value for correlation of a linear characteristic is independent of the actual round keys and the master key. It follows that for a key-alternating cipher, correlation of a linear approximation over an encryption is the sum of the correlation of all containing characteristics whose plaintext and ciphertext linear masks are the same as the ones for encryption, i. e.,

$$\mathrm{Cor}(\alpha_0 \xrightarrow{\mathtt{Enc}_K} \alpha_r) = \sum_{\alpha_1,\ldots,\alpha_{r-1}} \mathrm{Cor}(\alpha_0 \xrightarrow{\mathtt{R}_0} \alpha_1 \to \ldots \to \alpha_{r-1} \xrightarrow{\mathtt{R}_{r-1}} \alpha_r).$$

This is what is usually referred to as the *linear hulls* introduced by Nyberg [Nyb95] and it was proven by Daemen, Govaerts, and Vandewalle [DGV95] for the first time. In this way, ELC of a linear approximation is equal to

[Nyb95] Nyberg "Linear Approximation of Block Ciphers (Rump Session)"

[DGV95] Daemen, Govaerts, and Vandewalle "Correlation Matrices"

$$\mathrm{ELC}(\alpha_0 \xrightarrow{\mathtt{Enc}} \alpha_r) = \sum_{\alpha_1,\ldots,\alpha_{r-1}} \Big(\sum_{K\in\mathbb{F}_2^\kappa} (-1)^{\oplus_{i=0}^{r}\langle K_i,\alpha_i\rangle} \Big) \cdot \prod_{i=0}^{r-1} \mathrm{Cor}(\alpha_i \xrightarrow{\mathtt{R}_i^*} \alpha_{i+1}).$$

Similar to computing the probability of a differential, computing this correlation is often expensive, usually infeasible. Also, considering independent round keys as in Assumption 16 does not help here, and that is because, for encryption with independent round keys and a non-zero linear characteristic (i. e., at least one of α_is is non-zero), we have

$$\sum_{K\in\mathbb{F}_2^{n\cdot(r+1)}} (-1)^{\oplus_{i=0}^{r}\langle K_i,\alpha_i\rangle} = \sum_{K_0,\ldots,K_r\in\mathbb{F}_2^n} \prod_{i=0}^{r} (-1)^{\langle K_i,\alpha_i\rangle} = \prod_{i=0}^{r} \sum_{K_i\in\mathbb{F}_2^n} (-1)^{\langle K_i,\alpha_i\rangle} = 0.$$

In the last equality, a well-known equality is used that for a non-zero α, we have $\sum_{X\in\mathbb{F}_2^n}(-1)^{\langle X,\alpha\rangle} = 0$.[34]

[34] This equality is proven in [Car07, Lemma 1].

Nyberg [Nyb95] for the first time introduced the term *Linear Hull Potential* as the square of linear approximation correlation and also *Expected Linear Hull Potential* as in the following to compare the strength of the linear approximations over an encryption.

$$\mathrm{ELP}(\alpha_0 \xrightarrow{\mathtt{Enc}} \alpha_r) = 2^{-\kappa} \cdot \sum_{K\in\mathbb{F}_2^\kappa} \mathrm{Cor}(\alpha_0 \xrightarrow{\mathtt{Enc}_K} \alpha_r)^2$$

While computing Expected Linear Potential (ELP) still needs to go through all the key space, Daemen and Rijmen [DR02, Theorem 7.9.1] gave an expression to compute ELP of a key-alternating cipher with independent round keys.

[DR02] Daemen and Rijmen *The Design of Rijndael: AES - The Advanced Encryption Standard*

Theorem 31. *In a key-alternating cipher* $\mathtt{Enc}_K = \mathtt{A}_{K_r} \circ \mathtt{R}^*_{r-1} \circ \mathtt{A}_{K_{r-1}} \circ \ldots \circ \mathtt{R}^*_0 \circ \mathtt{A}_{K_0}$ *with* $r+1$ *independent round keys* $K_0, \ldots, K_r$*:*

$$\mathrm{ELP}(\alpha_0 \xrightarrow{\mathtt{Enc}} \alpha_r) = \sum_{\alpha_1,\ldots,\alpha_{r-1}} \prod_{i=0}^{r-1} \mathrm{Cor}(\alpha_i \xrightarrow{\mathtt{R}_i^*} \alpha_{i+1})^2. \tag{2.7}$$

Since this theorem has an important role in providing arguments about the resistance of key-alternating ciphers against the linear attack, and it is one of the concepts used in Chapter 5, we are going to prove it.

Proof. In a key-alternating cipher with independent round keys, $2^{n\cdot(r+1)} \cdot \mathrm{ELP}(\alpha_0 \xrightarrow{\mathtt{Enc}} \alpha_r)$ is equal to

$$
\begin{aligned}
&= \sum_{K_0,\ldots,K_r} \mathrm{Cor}(\alpha_0 \xrightarrow{\mathtt{Enc}_K} \alpha_r)^2 \\
&= \sum_{K_0,\ldots,K_r} \Big(\sum_{\alpha_1,\ldots,\alpha_{r-1}} \mathrm{Cor}(\alpha_0 \xrightarrow{\mathtt{R}_0} \alpha_1 \to \ldots \to \alpha_{r-1} \xrightarrow{\mathtt{R}_{r-1}} \alpha_r)\Big)^2 \\
&= \sum_{K_0,\ldots,K_r} \Big(\sum_{\alpha_1,\ldots,\alpha_{r-1}} \mathrm{Cor}(\alpha_0 \xrightarrow{\mathtt{R}_0} \alpha_1 \to \ldots \to \alpha_{r-1} \xrightarrow{\mathtt{R}_{r-1}} \alpha_r)\Big) \cdot \\
&\qquad \Big(\sum_{\beta_1,\ldots,\beta_{r-1}} \mathrm{Cor}(\alpha_0 \xrightarrow{\mathtt{R}_0} \beta_1 \to \ldots \to \beta_{r-1} \xrightarrow{\mathtt{R}_{r-1}} \alpha_r)\Big) \\
&= \sum_{K_0,\ldots,K_r} \sum_{\substack{\alpha_1,\ldots,\alpha_{r-1} \\ \beta_1,\ldots,\beta_{r-1}}} \mathrm{Cor}(\alpha_0 \xrightarrow{\mathtt{R}_0} \alpha_1 \to \ldots \to \alpha_{r-1} \xrightarrow{\mathtt{R}_{r-1}} \alpha_r) \cdot \\
&\qquad \mathrm{Cor}(\alpha_0 \xrightarrow{\mathtt{R}_0} \beta_1 \to \ldots \to \beta_{r-1} \xrightarrow{\mathtt{R}_{r-1}} \alpha_r)
\end{aligned}
$$

Considering $\beta_0 = \alpha_0$ and $\beta_r = \alpha_r$, and by replacing correlation of a linear characteristic, we have

$$
\begin{aligned}
&= \sum_{K_0,\ldots,K_r} \sum_{\substack{\alpha_1,\ldots,\alpha_{r-1} \\ \beta_1,\ldots,\beta_{r-1}}} (-1)^{\bigoplus_{i=0}^{r}\langle K_i,\alpha_i\rangle} \cdot \prod_{i=0}^{r-1} \mathrm{Cor}(\alpha_i \xrightarrow{\mathtt{R}_i^*} \alpha_{i+1}) \cdot \\
&\qquad (-1)^{\bigoplus_{i=0}^{r}\langle K_i,\beta_i\rangle} \cdot \prod_{i=0}^{r-1} \mathrm{Cor}(\beta_i \xrightarrow{\mathtt{R}_i^*} \beta_{i+1}) \\
&= \sum_{K_0,\ldots,K_r} \sum_{\substack{\alpha_1,\ldots,\alpha_{r-1} \\ \beta_1,\ldots,\beta_{r-1}}} (-1)^{\bigoplus_{i=0}^{r}\langle K_i,\alpha_i\oplus\beta_i\rangle} \cdot \prod_{i=0}^{r-1} \Big(\mathrm{Cor}(\alpha_i \xrightarrow{\mathtt{R}_i^*} \alpha_{i+1}) \cdot \mathrm{Cor}(\beta_i \xrightarrow{\mathtt{R}_i^*} \beta_{i+1})\Big)
\end{aligned}
$$

By changing the order of the two sums, we have

$$
\begin{aligned}
&= \sum_{\substack{\alpha_1,\ldots,\alpha_{r-1} \\ \beta_1,\ldots,\beta_{r-1}}} \prod_{i=0}^{r-1} \Big(\mathrm{Cor}(\alpha_i \xrightarrow{\mathtt{R}_i^*} \alpha_{i+1}) \cdot \mathrm{Cor}(\beta_i \xrightarrow{\mathtt{R}_i^*} \beta_{i+1})\Big) \cdot \sum_{K_0,\ldots,K_r} (-1)^{\bigoplus_{i=0}^{r}\langle K_i,\alpha_i\oplus\beta_i\rangle} \\
&= \sum_{\substack{\alpha_1,\ldots,\alpha_{r-1} \\ \beta_1,\ldots,\beta_{r-1}}} \prod_{i=0}^{r-1} \Big(\mathrm{Cor}(\alpha_i \xrightarrow{\mathtt{R}_i^*} \alpha_{i+1}) \cdot \mathrm{Cor}(\beta_i \xrightarrow{\mathtt{R}_i^*} \beta_{i+1})\Big) \cdot \prod_{i=0}^{r} \sum_{K_i} (-1)^{\langle K_i,\alpha_i\oplus\beta_i\rangle}
\end{aligned}
$$

Since $\sum_{K\in\mathbb{F}_2^n}(-1)^{\langle K,\alpha\oplus\beta\rangle}$ is equal to zero if $\alpha \neq \beta$ and it is equal to 2^n only if $\alpha = \beta$,[35] the above equation will be simplified to

$$
\begin{aligned}
&= \sum_{\alpha_1,\ldots,\alpha_{r-1}} \prod_{i=0}^{r-1} \Big(\mathrm{Cor}(\alpha_i \xrightarrow{\mathtt{R}_i^*} \alpha_{i+1}) \cdot \mathrm{Cor}(\alpha_i \xrightarrow{\mathtt{R}_i^*} \alpha_{i+1})\Big) \cdot \prod_{i=0}^{r} 2^n \\
&= 2^{n\cdot(r+1)} \sum_{\alpha_1,\ldots,\alpha_{r-1}} \prod_{i=0}^{r-1} \mathrm{Cor}(\alpha_i \xrightarrow{\mathtt{R}_i^*} \alpha_{i+1})^2
\end{aligned}
$$

which proves the theorem. □

Theorem 31 shows that the ELP of a linear hull is equal to the sum of square correlations of all characteristics whose plaintext and ciphertext masks are the same as in the linear hull.[36]

[35] $\sum_{X\in\mathbb{F}_2^n}(-1)^{\langle X,\alpha\rangle} = \begin{cases} 2^n & \text{if } \alpha = 0, \\ 0 & \text{otherwise.} \end{cases}$

[36] Note that using *correlation matrices* as presented in [DGV95] is essentially an equivalent approach to the above.

The independent round keys assumption makes the computation much easier. In linear cryptanalysis of key-alternating ciphers, similar to the case of differential one, it is common to assume that all the round keys are independent. Then, ELP of a linear hull can be computed as given in Eq. (2.7) without considering the value of the master key, but still requires considering all linear characteristics with different $\alpha_1, \ldots, \alpha_{r-1}$.

Linear Approximation Correlation of an SPN Cipher

Similar to computing EDP of a differential in an SPN cipher, it is also possible to simplify computing ELP of the linear hull under Assumption 16.

Lemma 32. *Let* $\mathrm{L} : \mathbb{F}_2^{n_1} \to \mathbb{F}_2^{n_2}$ *be a linear mapping.*[37] *Then for any* $\alpha \in \mathbb{F}_2^{n_1}$ *and* $\beta \in \mathbb{F}_2^{n_2}$*, we have*

$$\mathrm{Cor}(\alpha \xrightarrow{\mathrm{L}} \beta) = \begin{cases} 1 & \text{if } \alpha = \mathrm{L}^T(\beta), \\ 0 & \text{otherwise,} \end{cases}$$

recalling that L^T *denotes the transposed linear mapping of* L*.*

[37] Note that this is the general case for all linear mappings, while the linear mappings in SPN all are bijections.

Lemma 33. *Let* $\mathrm{S} : \mathbb{F}_2^{s \cdot m} \to \mathbb{F}_2^{s \cdot m}$ *be the parallel application of* m *bijective S-boxes* $S_j : \mathbb{F}_2^s \to \mathbb{F}_2^s$ *with* $j \in \mathbb{Z}_m$ *such as defined in Definition 6. Then for any* $\alpha, \beta \in (\mathbb{F}_2^s)^m$*, we have*

$$\mathrm{Cor}\left(\phi_{s,m}(\alpha) \xrightarrow{\mathrm{S}} \phi_{s,m}(\beta)\right) = \prod_{j=0}^{m-1} \mathrm{Cor}(\alpha[j] \xrightarrow{S_j} \beta[j]).$$

Proposition 34. *Let* Enc *be the* r*-round SPN block cipher defined in Definition 6 with* $r+1$ *independent round keys* $K_0, \ldots, K_r$*. Then the ELP of a linear hull is equal to*

$$\mathrm{ELP}(\alpha_0 \xrightarrow{\mathrm{Enc}} \alpha_r) = \sum_{\alpha_1, \ldots, \alpha_{r-1}} \prod_{i=0}^{r-1} \prod_{j=0}^{m-1} \mathrm{Cor}\left(\phi_{s,m}^{-1}(\alpha_i)[j] \xrightarrow{S_{ij}} \phi_{s,m}^{-1}(\beta_i)[j]\right)^2. \tag{2.8}$$

where $\beta_i := \mathrm{L}_i^T(\alpha_{i+1})$ *with* $i \in Z_r$*.*

Due to the straightforward proof of Eq. (2.8) using previous lemmas, the proof is not given here. In the following, the rest of the definitions regarding the linear approximation properties of SPN block ciphers that will be necessary for the rest of the thesis are explained.

Definition 35 (Linear Approximation Table (LAT))**.** Let $F : \mathbb{F}_2^{n_1} \to \mathbb{F}_2^{n_2}$ be a vectorial Boolean function. The LAT of F is a table whose (α, β)-th element, i. e., $\mathrm{LAT}[\alpha][\beta]$, with $\alpha \in \mathbb{F}_2^{n_1}$ and $\beta \in \mathbb{F}_2^{n_2}$ shows the value of $\left|\{X \mid \langle \alpha, X \rangle = \langle \beta, S(X) \rangle\}\right| - \left|\{X \mid \langle \alpha, X \rangle \neq \langle \beta, S(X) \rangle\}\right|$,[38] i. e., $2^{n_1} \cdot \mathrm{Cor}(\alpha \xrightarrow{S} \beta)$.

[38] In some literature, LAT is defined based on linear bias, i. e., $\mathrm{LAT}[\alpha][\beta]$ shows number of $X \in \mathbb{F}_2^{n_1}$ that $\langle \alpha, X \rangle = \langle \beta, S(X) \rangle$.

Definition 36 (Linearity)**.** For the vectorial Boolean function $F : \mathbb{F}_2^{n_1} \to \mathbb{F}_2^{n_2}$, the *linearity* is defined as

$$\mathrm{lin}(F) = \max_{\substack{\alpha \in \mathbb{F}_2^{n_1}, \\ \beta \in \mathbb{F}_2^{n_2} \setminus \{0\}}} \left| \left|\{X \mid \langle \alpha, X \rangle = \langle \beta, S(X) \rangle\}\right| - \left|\{X \mid \langle \alpha, X \rangle \neq \langle \beta, S(X) \rangle\}\right| \right|.$$

It is noteworthy to mention that in an s-bit S-box, it is always $\mathrm{LAT}[0][0] = 2^s$, and if it is a bijective S-box, then for any non-zero $\alpha \in \mathbb{F}_2^s$, $\mathrm{LAT}[0][\alpha] = \mathrm{LAT}[\alpha][0] = 0$. This means that input mask of a bijective S-box is zero if and only if its output mask is zero. Similar to differential characteristics, in a linear characteristic, an S-box is called active if its input (or output) mask is non-zero and $\delta(X)$ is used to denote the *activity pattern* in X which is defined as in Definition 23.

Similar to computing EDP of a differential, computing the ELP (Eq. (2.8)) requires considering the absolute correlation of all underlying linear characteristics with different $\alpha_1, \ldots, \alpha_{r-1}$.

Extensions and Generalizations

For linear cryptanalysis also several variants have been proposed that only two of the most known variants are explained here that will be used in this thesis: *multidimensional linear hull* and *zero-correlation linear hull*.

MULTIDIMENSIONAL LINEAR HULLS: Similar to extending differentials to truncated differentials, in multiple and multidimensional linear hull analysis, the plaintext mask can be chosen from a set $\mathcal{U}$ and the ciphertext one from a set $\mathcal{V}$. This extension was introduced by Kaliski Jr. and Robshaw [KR94] for the first time and analyzed in detail by Hermelin, Cho, and Nyberg [HCN19].

[KR94] Kaliski Jr. and Robshaw "Linear Cryptanalysis Using Multiple Approximations"

[HCN19] Hermelin, Cho, and Nyberg "Multidimensional Linear Cryptanalysis"

The ELC and ELP of a multidimensional linear hull is denoted by

$$\mathrm{ELC}(\mathcal{U} \xrightarrow{\mathtt{Enc}} \mathcal{V}) = \frac{1}{|\mathcal{U}|} \cdot \sum_{\substack{\alpha \in \mathcal{U} \\ \beta \in \mathcal{V}}} \mathrm{ELC}(\alpha_0 \xrightarrow{\mathtt{Enc}} \alpha_r),$$

$$\mathrm{ELP}(\mathcal{U} \xrightarrow{\mathtt{Enc}} \mathcal{V}) = \frac{1}{|\mathcal{U}|} \cdot \sum_{\substack{\alpha \in \mathcal{U} \\ \beta \in \mathcal{V}}} \mathrm{ELP}(\alpha_0 \xrightarrow{\mathtt{Enc}} \alpha_r),$$

which is based on the assumption that all the masks in set $\mathcal{U}$ occur uniformly.

In the classical multidimensional linear hulls, $\mathcal{U}$ and $\mathcal{V}$ are a special kind of linear subspaces. The more general case where $\mathcal{U}$ and $\mathcal{V}$ are arbitrary sets can be handled similarly, but some more care has to be taken about interpreting the results, due to additional dependencies. In this thesis, the general multidimensional linear hull definition is used, and it will be mentioned explicitly in case the sets are linear subspaces.

ZERO-CORRELATION LINEAR HULLS: In 2011, Bogdanov and Rijmen [BR14] introduced the idea of *zero-correlation linear hulls*. The linear hull (α, β) is called a zero-correlation linear hull for (reduced-round) encryption $\mathtt{Enc}$ if for all key values K, $\mathrm{Cor}(\alpha \xrightarrow{\mathtt{Enc}_K} \beta) = 0$. This extension of linear analysis is similar to the impossible differentials in the differential analysis.

[BR14] Bogdanov and Rijmen "Linear hulls with correlation zero and linear cryptanalysis of block ciphers"

2.3.7 *Integral Attacks*

The idea of *integral* attack was originally introduced by Daemen, Knudsen, and Rijmen [DKR97] to analyze the block cipher SQUARE.[39] Later, Knudsen and Wagner [KW02] generalized the idea which is likely to be efficient to analyze SPN block ciphers.

[DKR97] Daemen, Knudsen, and Rijmen "The Block Cipher Square"

[39] For this reason, it is also called the *Square attack*.

[KW02] Knudsen and Wagner "Integral Cryptanalysis"

The integral attack takes a multi-set of plaintexts, in which particular words are fixed to a constant value, and the other cells can contain all possible values. Considering a set of such plaintexts, each s-bit word of the cipher state belongs to one of the four cases below:

- All (**A**): all possible values in $\mathbb{F}_2^s$ appear the same number of times, i. e., with a uniform distribution.
- Balanced (**B**): the XOR sum of all values in the word is 0.
- Constant (**C**): the value of the cell is constant.
- Unknown (**U**): no particular property holds.

To find the longest integral distinguisher for an SPN cipher, we first, set one active word to the state, i. e., belonging to the **A** case, and the other words are fixed to a constant value, i. e., the **C** case. The cipher state is processed by the round functions until all nibbles become unknown, i. e., **U** case. Then, we extend it to a higher-order integral by propagating the active cell in the backward direction until all the nibbles become active.

DIVISION PROPERTY: A more recent technique to build integral distinguisher is to use the so-called *division property* introduced by Todo [Tod15]. This technique was later refined into *bit-based division property* by Todo and Morii [TM16]. While in the typical integral distinguisher, all values for an active word must be considered, in the division property this restriction is omitted and it is possible to consider more general cases.

[Tod15] Todo "Structural Evaluation by Generalized Integral Property"

[TM16] Todo and Morii "Bit-Based Division Property and Application to Simon Family"

2.4 HARDWARE CONCERNS IN BLOCK CIPHER DESIGN

Every cryptographic designer deals with the trade-off between security, costs, and performance of the scheme. In the case of block ciphers, the key and block length usually take the role for security-cost trade-off, while the number of rounds is the one for security-performance trade-off, and it is the hardware architecture for cost-performance trade-off. Optimizing over only two of these design goals is usually easily achievable, i. e., security and low costs, security and performance, or low costs and performance, but it is challenging to optimize all three design goals simultaneously.

In the following, starting with Section 2.4.1, the implementation metrics, architectures, and complexity notions are explained. Subsequently, we briefly talk about the matter of *lightweight cryptography* in Section 2.4.2. Finally in Section 2.4.4, we explain the concept of *physical attacks*, especially *fault analysis* and the state of the art for its countermeasures.

2.4.1 *Hardware Implementations*

Hardware Metrics

Usually, the following metrics are used to compare the efficiency of hardware implementations:

AREA: It is usually measured in μm^2, but since this value depends on the fabrication technology and the standard cell library, it is always stated as *gate equivalents* (GE) independent of the technology and the library. One GE is equivalent to the area required to implement the two-input NAND gate with the lowest driving strength of the appropriate technology. Thereby, the area in GE is derived by dividing the area in μm^2 by the area of a two-input NAND gate.

CYCLES: It is the number of clock cycles required to compute and output the results.

TIME: It is the required time for a certain operation, i. e., the division of the number of cycles by the operating frequency.

THROUGHPUT: It is the bit rate production of a new output with respect to time, i. e., the number of output bits divided by the time, and it is expressed in bits per second (bps).

POWER: It is usually the power consumption estimated on the gate level by the synthesizer tool and is typically expressed in µW. Note that power estimations on the transistor level are more accurate, but this requires further design steps in the design flow.

ENERGY: It is the power consumption over a certain time period, i. e., multiplicating the power consumption with the required time, and it is typically expressed in µJ.

Implementation Architectures

To reach the design goals of the application, the designer has four main choices for the hardware architecture: *Unrolled*, *Pipelined*, *Round-based*, and *Serialized*.

UNROLLED: In an unrolled implementation, the cipher's whole encryption or decryption process is computed within only one clock cycle without using any registers in the combinatorial circuit. The unrolled implementation is the appropriate architecture for *low-latency* designs that there designer wants to design encryption with the capability of implementing it in only a single clock cycle with high frequency.[40]

[40] Inverse of the maximum clock frequency is called *latency*.

PIPELINED: While in a pipelined implementation, the circuit for the whole encryption or decryption process is implemented (similar to the unrolled implementation), some registers are inserted in the critical path[41] to increase the maximum clock frequency. This architecture provides a higher throughput rate but with the cost of higher area and power consumption.

[41] That is the path with maximum delay.

ROUND-BASED: In a round-based implementation, each round function of the cipher is computed within one clock cycle. Compared to the unrolled or pipelined ones, this architecture reduces the area and power consumption at the cost of decreasing the throughput.

SERIALIZED: In a serialized implementation, each round function is computed in several clock cycles, and in each clock cycle, a small part of the round function is computed (e. g., only one S-box, or only one word of the linear layer). This architecture makes it possible to have a lower area and power consumption than that of the two other architectures. But, on the other hand, it has the lowest throughput. It is noteworthy that serializing can reduce the area and power consumption until one specific point. This is because the serializing technique requires a more complicated control logic, and after a point, implementing this control logic may require more overhead than before.

Implementation Complexity

To mathematically model the costs of the implementation of a Boolean circuit for some specific applications, some terms are defined that are known as the *complexity* of the circuit. In the following, we bring the general definition of these complexities. In all of these definitions, we consider that $\mathcal{G}$ is the set of all allowed gates to use,[42] e. g., all the gates with at most two inputs.

[42] The set $\mathcal{G}$ is sometimes called the *basis* of the implementation.

Definition 37 (Gate Count Complexity)**.** It is the smallest number of gates chosen from $\mathcal{G}$ required to compute the function.

Even though different types of gates have different implementation (area) costs, this definition is usually considered the minimum area cost for hardware implementation of a function. To have a reasonable estimation of the area cost, it is common practice to only consider the gates with a fan-in[43] of at most 2.

[43] Number of the inputs to the gate.

Definition 38 (Gate Depth Complexity)**.** It is the minimum length of the longest path (concerning the number of gates chosen from $\mathcal{G}$ used in the path) from an input to an output for implementing the function.

Note that if any gate with any fan-in number is allowed, then every function can be computed by a circuit with depth 2, e. g., by expressing the function in conjunctive or disjunctive normal form. However, this can lead to a $\mathcal{G}$ that is usually not desirable. Again, it is typical to use only gates with a fan-in of at most 2.

Even though different types of gates have different implementation (delay) costs, this definition is usually considered the minimum delay cost for hardware implementation of a function. This means that optimizing for this goal reduces the total delay and, therefore, increases the clock frequency.

Definition 39 (Multiplicative Complexity)**.** [BPP00] It is defined as the smallest number of nonlinear gates with fan-in 2 required to compute the function.

[BPP00] Boyar, Peralta, and Pochuev, "On the Multiplicative Complexity of Boolean Functions over the Basis ($\wedge$, $\oplus$, 1)"

For instance, if $\mathcal{G}$ is restricted to the AND, OR, XOR, and NOT operations, we only need to consider the number of ANDs and ORs. This optimization is useful for the applications used in multi-party computation, fully homomorphic encryption, and also in the side-channel countermeasures e. g., in masking schemes like threshold implementations, where the non-linear gates are typically more expensive to protect.

2.4.2 *Lightweight Cryptography*

The IoT is becoming more and more prevalent in our daily lives as it connects various kinds of everyday objects via networks and transfers (personal) data between them. Small embedded devices in IoT have a vital role in these systems, highlighting their security significance. Hence, their security is a non-deniable requirement in relevant services like secure communication or storage of private and sensitive data.

In symmetric cryptography, motivated by new application scenarios – in particular by the IoT – *lightweight* cryptography has been a very active research area in the last decade. While there is no strict definition of lightweight cryptography, it is usually understood as cryptography with a strong focus on efficiency. Here efficiency can be measured according to various criteria and their combination.

The first generation of lightweight ciphers, e. g., PRESENT [Bog+07] and KATAN [DDK09], focused on chip area only and used very simple round functions as the main building block. Later generations of lightweight ciphers broadened the scope significantly. By now, we have at hand dedicated ciphers optimized with respect to code-size (e. g., PRIDE [Alb+14] and SPECK [Bea+13; Bea+15]), latency (e. g., PRINCE [Bor+12], MANTIS [Bei+16], and QARMA [Ava17]), efficiency of adding countermeasures against passive SCA (e. g., the family of LS-Designs [Gro+15], PICARO [PRC12] and ZORRO [Gér+13] (software oriented), FIDES [Bil+13] and actually NOEKEON [Dae+00] (hardware oriented) even before the term lightweight cryptography was invented), efficient fault detection (e. g., FRIT [Sim+18]), and energy-efficient block ciphers (e. g., MIDORI [Ban+15] and GIFT [Ban+17]). Moreover, the overhead of implementing decryption on top of encryption has been subject to optimization (e. g., ICEBERG [Sta+04], MIDORI and NOEKEON where the components are involutions and PRINCE with its α-reflection property).

Besides focusing on various criteria, there has also been an advance in the general design philosophy. Indeed, many recent lightweight ciphers (e. g., LED [Guo+11], SKINNY [Bei+16] and MIDORI) use the general framework of the AES round function and fine-tune its components to achieve better performance while previous constructions were rather ad-hoc. Borrowing from AES in particular allows for simpler security analysis, following e. g., the wide-trail strategy (see [Dae95]). More recently, there are also attempts to design lightweight tweakable block ciphers which allows for novel encryption modes and efficient constructions of authenticated encryption schemes [LRW02]. Examples here include SKINNY, MANTIS, QARMA, and JOLTIK, and the TWEAKEY framework [JNP14] as a general design principle.

[Dae95] Daemen, *Cipher and Hash Function Design, Strategies Based on Linear and Differential Cryptanalysis, PhD Thesis*

In the following, we briefly talk about the state of the art in the design of low-latency block ciphers that is our focus in Chapter 4.

Low-Latency Ciphers

Various technology fields, including industrial automation, robotics, and the 5th generation mobile network, urge for real-time operation and require low-latency execution while preserving the highest security levels. Low-power and low-energy requirements are of equal importance, especially

when considering the IoT market, where the majority of devices are in a low-power mode during most of their lifetime. Those devices occasionally are called, carry out a quick computation, store or communicate securely, and then go back to standby mode.

Securing these devices is a complex problem and requires a multi-layered approach. Starting with trust provisioning and then enabling secure boot, secure debug access, secure firmware update, secure test, and life-cycle management are essential for creating a secure execution environment. One of the crucial aspects of a secure execution environment is secure storage, a security mechanism used for keeping confidentiality and integrity of valuable code and data stored on the device.

Memory encryption has been used in the PC world for a long time already. Some examples include IBM's SecureBlue++, Intel's SGX, and AMD's SEV encryption and integrity mechanisms. Those mechanisms are used to protect valuable assets against an attacker capable of monitoring and/or manipulating the memory content. The attacker is assumed to be employing various software tools running on the targeted platform and being able to manipulate the integrity of any underlying hardware by using invasive methods such as probing, voltage glitching, electromagnetic fault injection, etc. While the device must remain secure in such an adversary's hands, the level of protection differs depending on whether the memory is volatile or non-volatile or whether it is internal or external to the system on chip (SoC). Securing the SoC-external memory is especially challenging while, at the same time, the advances of CMOS technology lead to increased production of FLASH-less microcontrollers. Add to this the everlasting requirement to minimize power and energy consumption and the never-ending race for higher performance, and it will become clear why designing an efficient memory encryption scheme is a serious challenge.

PRINCE [Bor+12] is the first publicly known low-latency block cipher that got scrutinized by the cryptographic community.[44] As a result, PRINCE has been deployed in several products including LPC55S of NXP Semiconductors,[45] which is a family of highly constrained general-purpose IoT microcontrollers.

As pointed out in [Bor+12; KNR12], a low-latency block cipher design's ultimate goal is to encrypt a block of data in a single clock cycle. The best illustration of the importance of meeting this goal is to look at the comparison of PRINCE and AES in the low-latency setting. When implemented fully unrolled, PRINCE occupies 4 times less silicon area while, at the same time, reaching 8 times higher clock frequency than AES.

Although some design principles have been explored during the design of PRINCE, there has been little work going on to determine the design choices that lead to the lowest-latency and most energy-efficient cipher architecture. Several parameters contribute to the efficiency of a given cipher design: area, latency, throughput, power, and energy. Several other designs, including MIDORI [Ban+15], MANTIS [Bei+16] and QARMA [Ava17] have been particularly optimized for one or more of these parameters.

[44] https://www.emsec.ruhr-uni-bochum.de/research/research_startseite/prince-challenge/.

[45] See https://www.nxp.com/docs/en/application-note/AN12283.pdf.

[Bor+12] Borghoff et al., "PRINCE - A Low-Latency Block Cipher for Pervasive Computing Applications - Extended Abstract"

[KNR12] Knežević, Nikov, and Rombouts, "Low-Latency Encryption - Is "Lightweight = Light + Wait"?"

2.4.3 *Physical Attacks and Countermeasures*

[BDL97] Boneh, DeMillo, and Lipton "On the Importance of Checking Cryptographic Protocols for Faults (Extended Abstract)"

[BS97] Biham and Shamir "Differential Fault Analysis of Secret Key Cryptosystems"

The application of *physical attacks* against cryptographic primitives started with introducing *fault injection attacks* by Boneh, DeMillo, and Lipton [BDL97], and by Biham and Shamir [BS97]. In a physical attack, we face a situation where the cryptographic devices are in the adversary's hands. This enables the attacker to apply all kinds of additional attacks on implementing the cryptographic scheme, even though it is mathematically secure. The reason behind this is that secret intermediate values of the execution of cryptographic algorithms can be recovered. Thus, not only the underlying cryptographic scheme has to fulfill mathematical security requirements, the device that implements the scheme should also be physically secure.

As a result, integrating countermeasures to prevent such physical attacks is mandatory for widespread products that deal with security and privacy concepts. However, those countermeasures usually come at a significant cost. Since the cryptographic algorithms are usually designed considering their robustness against cryptanalytic attacks, the integration of countermeasures against physical attacks into their implementation becomes – most of the time – challenging and not necessarily efficient.

Physical attacks differ significantly in terms of cost, time, equipment, and expertise needed. The main categorization of physical attacks is based on two criteria. The first criterion is based on if the attack is active or passive.

In an *active attack*, the device, its inputs, and/or its environment are manipulated to make the device behave differently. Then the secret data is extracted by comparing this different behavior with its genuine behavior. Active attacks are also known as *fault-injection attacks* [BDL97; BS97], and probably the most powerful class of physical attacks.

While active attacks aim to disturb the devices' regular execution so that a fault occurs, in passive attacks, the attacker tries to extract secret data based on observing the device's physical properties (e. g., the power consumption, the execution time).

The second criterion of categorization is based on the attack's invasiveness: non-invasive, semi-invasive, and invasive attacks. A non-invasive attack does not require any initial preparations of the device under test and will not physically harm the device during the attack. The attacker can either tap the wires to the device or plug it into a test circuit for the analysis. Invasive attacks require direct access to the device's internal components possible by depacking the device's chip and usually need a well-equipped and knowledgeable attacker to succeed. Semi-invasive attacks fill the gap between non-invasive and invasive attacks. They are not as expensive as invasive attacks but are not as easily repeatable as non-invasive attacks.

Side Channel Attacks (SCAs) are the non-invasive passive attacks that are important topics in the implementation of cryptographic devices.

In this thesis, the main focus is on fault injection attacks, especially Differential Fault Analysis (DFA) attacks, and for this reason, other kinds of physical attacks are not introduced.

2.4.4 *Fault Injection Attacks*

Fault attacks are the first instantiation of physical attacks in literature and were introduced by Boneh, DeMillo, and Lipton [BDL97] and by Biham and Shamir [BS97] in the same year. In this attack, the attacker tries to tamper with the device such that the output is computed differently than specified. The way of tampering with the device by the attacker is called *fault injection*, and the corresponding output is called the *faulty output*. Then, based on the faulty output's mathematical analysis with the genuine output, the attacker can recover the secret data.

Fault injections, which are usually transient faults, can be made using a clock glitch [Ago+10b] (which violates the delay of the circuit's critical path), under-powering [SGD08] (which, in addition to a setup-time violation, may modify the circuit's execution flow), an EM glitch [Deh+12] (which can change the transistors' state), or a laser beam [Ago+10a] (which as the most precise tool, it can change the state of particular transistors). Recent works have shown that it is even possible to target multiple transistors independently with advanced optical setups [SHS16].

[Ago+10b] Agoyan, Dutertre, Naccache, Robisson, and Tria, "When Clocks Fail: On Critical Paths and Clock Faults"

[SGD08] Selmane, Guilley, and Danger, "Practical Setup Time Violation Attacks on AES"

[Deh+12] Dehbaoui, Dutertre, Robisson, and Tria, "Electromagnetic Transient Faults Injection on a Hardware and a Software Implementations of AES"

[Ago+10a] Agoyan, Dutertre, Mirbaha, Naccache, Ribotta, and Tria, "How to flip a bit?"

[SHS16] Selmke, Heyszl, and Sigl, "Attack on a DFA Protected AES by Simultaneous Laser Fault Injections"

Most of the fault injection attacks applied on symmetric ciphers are based on mathematical comparing of a single or multiple faulty outputs to the genuine ones resp., called DFA introduced in [BS97]. DFA is closely related to reduced-round differential cryptanalysis that the attacker injects the fault in an intermediate state of the encryption where the injected fault is modeled as the difference between two states (the genuine one and the faulty output) and observes the output difference by comparing the genuine output and the faulty one. However, later, several extensions of fault analysis have been introduced, which are different in certain aspects or requirements: Fault Sensitivity Analysis (FSA) [Li+10], Statistical Fault Attack (SFA) [Fuh+13], Differential Fault Intensity Analysis [Gha+14], etc.

[Li+10] Li, Sakiyama, Gomisawa, Fukunaga, Takahashi, and Ohta, "Fault Sensitivity Analysis"

[Fuh+13] Fuhr, Jaulmes, Lomné, and Thillard, "Fault Attacks on AES with Faulty Ciphertexts Only"

[Gha+14] Ghalaty, Yuce, Taha, and Schaumont, "Differential Fault Intensity Analysis"

SFA improves the performance of DFA by statistically analyzing the faulty outputs. While in DFA and SFA, the attacker uses the faulty output, in FSA, the attacker exploits the dependency of secret data to the delay of the circuit. Furthermore, Ineffective Fault Attack (IFA) [Cla07] and Statistical Ineffective Fault Attack (SIFA) [Dob+18] utilizes the fault-free outputs even though the fault is injected, i. e., ineffective fault injections.

[Cla07] Clavier, "Secret External Encodings Do Not Prevent Transient Fault Analysis"

[Dob+18] Dobraunig, Eichlseder, Korak, Mangard, Mendel, and Primas, "SIFA: Exploiting Ineffective Fault Inductions on Symmetric Cryptography"

Unlike the others, SIFA does not rely on strong assumptions and fault models. It is applicable in a wide range of models, even in the presence of countermeasures. The only requirement of SIFA is some dependency between the output and the intermediate value on which the ineffective fault appeared. However, the attacker does not need to know this dependency. Indeed, the probability of changing an intermediate value, x, by a fault injection (like stuck-at-`0`/`1`) is not the same for all x values. This bias is the only requirement of SIFA.

The concept of counteracting fault injection attacks has sometimes been confounded with the fault tolerance design methodology. As a result, the dependability community has introduced several fault-detection schemes for cryptographic hardware with extremely marginal capability to counteract fault-injection attacks [Ber+03]. The main reason behind such shortcomings is the difference between the faults' nature in these two concepts. The

dependability community considers the faults as single-event-upsets (SEUs) that may rarely happen during the system's operation and are of limited weight, e. g., one or two bits in the entire circuit. In case of fault-injection attacks, depending on the underlying fault model, the adversary tries to hit several locations (several bits) of the circuit as many times as he needs to recover the secret.

Applying redundancy to the circuit was the first reasonable choice to counteract the faults attacks. This can be done by calculating twice, either in parallel (area duplication) or consecutively (time duplication) [Bar+06; Guo+15]. By comparing the outputs from both computations, it is possible to detect some of the faults. Duplication can ensure detection of all asymmetric faults – those faults the first half is not the same as the second half – at the cost of about twice the area or twice the computation time, resp.

[Bar+06] Bar-El, Choukri, Naccache, Tunstall, and Whelan, "The Sorcerer's Apprentice Guide to Fault Attacks"

[Guo+15] Guo, Mukhopadhyay, Jin, and Karri, "Security analysis of concurrent error detection against differential fault analysis"

Duplication does not provide security against the attacker who can inject the same fault in both of the computations. Applying such symmetric faults is often easily possible, e. g., by a clock glitch.

Applying Error Detection Codes (EDCs) seems to be a promising approach to counter fault attacks. Depending on the underlying code, it is possible to increase the maximum number of faults that the code can detect. Concurrent Error Detections (CEDs) are the countermeasures that check the information's consistency simultaneously with the computation. There is an extensive body of work related to the design and implementation of EDC-based CEDs [Ber+03; Och+05; NFR07; Bri+14; Ana+16; CG16].

It is noteworthy to mention that none of the CED countermeasures can protect against SIFA. A trivial solution to protect against SIFA is to correct faults to weaken the bias in the distribution of the intermediate values, as mentioned earlier. Therefore, the attacker cannot distinguish between the corrected faults and ineffective faults; hence the attack would be more challenging. Majority voting and Error Correction Code (ECC) are among such techniques. The majority voting needs the circuit to be instantiated an odd number of times; hence, the area overhead is expected to be significant. Besides, it is stated in [Dob+18] that faults can be injected on multiple instances of majority voting with fewer complications, leading to successful attacks.

CED Schemes

As stated before, CED schemes are a common countermeasure against fault attacks. Since CED is currently the most secure approach to protect cryptographic systems on an algorithmic level, it is the focus of this thesis. In the following, the basic structure and corresponding notations are briefly introduced, which will be used in the rest of the thesis.

Although a CED scheme can be realized in various ways with different redundancy types, its basic structure is mostly the same. These schemes usually include the original target algorithm A and its designated *predictor* A'. Such a redundancy can be categorized into different classes [Guo+15] listed below. Note that in Figures 2.6 to 2.8, where block diagrams of the corresponding schemes are shown, the underlying cryptographic algorithm is considered as an encryption in a product cipher that is the iteration of a

round function R_{K_i}. Besides, for distinguishing between the encryption part A, and the predictor part A′, different colors in all figures are used where the first one is in black and the latter is in gray.

- In *timing redundancy*, the computation is repeated one (or more) time(s) by the same piece of hardware, and the results are compared [MSY06]. As the name says, it has a timing overhead, which linearly increases by the number of times a computation is repeated. Despite its simplicity, it cannot detect permanent faults.

- In *area redundancy*, the target module is instantiated more than once so that they all operate in parallel with the same input, and their result can be compared. Although it has no timing overhead, it leads to a factor of two (or more) area overhead, but it can detect both transient and permanent faults. The simplest version of such a scheme is *duplication*, where the entire circuit is instantiated two times [MSY06].

- In *information redundancy*, the goal is to keep the null timing overhead and to reduce the area overhead compared to duplication. Instead of fully instantiating the same module two times, it calculates a signature (e. g., parity) to enable the correctness check [Ber+03]. Of course, there is a trade-off between the area overhead and the efficiency of a fault-detection scheme. In the most extreme case, when the redundancy is a single parity bit, the area overhead is minimized, but only certain faults can be detected.

- *Hybrid redundancy* is a customized type of either one or a combination of the above-explained schemes. For instance, instead of duplication, one can compute the inverse of the operations by the second module [Kar+02; Sat+08], leading to a limited area and limited timing redundancy.

Note that depending on A and the underlying code, predictors may not be able to compute the redundant mapping of the output of A. Hence, they may require intermediate results from A during the computation. When both A and A′ are finished, their results are checked in a *Check Point* to detect possible errors before transmitting the output.

Due to their limited area overhead, the schemes based on information redundancy have been investigated more widely than the others. However, they usually come with severe shortcomings due to the non-transparency of the signature (e. g., parity) to the underlying function. For clarification, consider Figure 2.8; depending on the underlying signature scheme, a certain type of faults can be detected during the computation of R_{K_i}. However, it cannot detect any fault injected in the register cells. The reason behind such a split – as stated above – is the non-transparency between the signatures and the operations. Most of the signatures, e. g., parities, are based on linear functions. Hence, the signature of input and output of a non-linear operation, e. g., an S-box, cannot be easily used to detect a fault occurrence. In such scenarios the faults injected right after a signature control and the next signature generation (red arrow in Figure 2.8) cannot be detected.

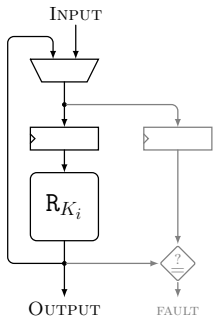


FIGURE 2.6: CED with Timing Redundancy.

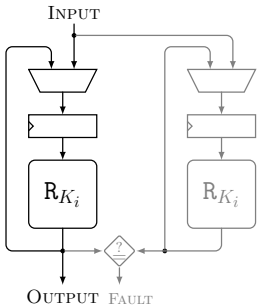


FIGURE 2.7: CED with Area Redundancy.

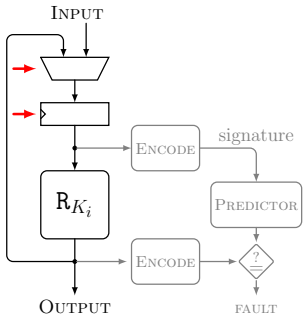


FIGURE 2.8: CED with Information Redundancy.

[Kar+02] Karri, Wu, Mishra, and Kim, "Concurrent error detection schemes for fault-based side-channel cryptanalysis of symmetric block ciphers"

[Sat+08] Satoh, Sugawara, Homma, and Aoki, "High-Performance Concurrent Error Detection Scheme for AES Hardware"

To avoid such issues, duplication is a natural and straightforward solution. It can be indeed seen as an information redundancy scheme where the signature is the same as the original data. Despite its simplicity, duplication has a pitfall of not detecting symmetric faults, i. e., those faults which are similarly injected into both original and redundant modules (usually possible by employing two precisely-localized laser beams).

Fault coverage is a metric to evaluate CED schemes and has commonly been used in several related works, e. g., [Ber+03]. While a higher fault coverage theoretically indicates a higher level of protection, the practical security strongly depends on the chosen fault model and its closeness to reality. This model should be carefully adapted based on the assumed adversary to avoid under- or overestimating the coverage. While underestimation reduces the security, overestimation results in inefficient implementations due to large overheads.

Definition 40 (Fault Coverage)**.** The fault coverage of a given CED scheme **C** in a specific fault model **M** is defined as the ratio

$$Cov(\mathbf{C}, \mathbf{M}) = \frac{N_{cov}(\mathbf{C}, \mathbf{M})}{N_{tot}(\mathbf{C}, \mathbf{M})},$$

where $N_{tot}(\mathbf{C}, \mathbf{M})$ stands for the number of possible faults of **C** and $N_{cov}(\mathbf{C}, \mathbf{M})$ for the number of covered (detectable) faults, both adjusted to the distribution of **M**.

Error Detecting and Correcting Codes

EDCs and ECCs are essential aspects of information theory and are often used in CED schemes to counter fault attacks. In the following, some notions borrowed from [MS77] related to linear codes are recalled, which will be used in this thesis.

[MS77] MacWilliams and Sloane, *The Theory of Error Correcting Codes*

Definition 41 (Binary Code)**.** A binary $[n, k]$-code C with $n > k$, is defined as a bijective mapping from the space of messages $\mathcal{M} = \mathbb{F}_2^k$ to the space of codewords $\mathcal{C} \subset \mathbb{F}_2^n$. That means each message $x \in \mathcal{M}$ is mapped to a unique codeword $c \in \mathcal{C}$ with $c = C(x)$.

The operation to compute the codeword from a message is called *encoding*, while the reverse operation to compute the message from the codeword is called *decoding*. The code parameters n and k are referred to as the *length* and *rank* of the $[n, k]$-code C, resp. Besides, the difference between length and rank values is referred to as the *parity size*, i. e., $n-k$.

Since symmetric cryptographic primitives rely on the binary field, the codes used in CEDs of these primitives are binary codes for the matter of performance. Thus, this thesis's focus is only on binary codes, and for simplicity, whenever writing "code" in this thesis, we mean a binary code.

Definition 42 (Systematic Code)**.** A code in which the message x is embedded in the codeword c is called a systematic code. This means that the codeword c is the concatenation of x with a parity (redundancy) x', i. e., $c = (x \| x')$, while the parity bits are generated from x.

This special structure of the codewords in a systematic code enables a simple split of the data paths between message and check bits. Therefore, the original implementation of the target operation A can stay as it is, while it is extended with the predictors A' for the check bits. Furthermore, systematic codes do not require extra logic to recover the message from the codeword. The decoder can take the first k bits of a codeword to get the message for zero implementation cost. This implementation advantage justifies why most CEDs make use of systematic codes.

Definition 43 (Linear Code)**.** The $[n,k]$-code C for which the codeword space is a vector subspace over $\mathbb{F}_2^n$ is called *linear code*.

Definition 44 (Generator Matrix)**.** For a linear $[n,k]$-code, the $k \times n$ matrix G that maps a message into the corresponding codeword, is called the *generator matrix*, i. e., $C(x) = x \cdot G$.

Definition 45 (Parity Check Matrix and Syndrome)**.** For a linear $[n,k]$-code, since the rank of the generator a matrix is k, there are $n-k$ linear equations between the codeword bits to be satisfied. These equations can be shown as matrix multiplication. The $n \times (n-k)$ matrix H that checks if an element of $\mathbb{F}_2^n$ is a possible codeword is called the *parity check matrix*. Besides, for any $x \in \mathbb{F}_2^n$, the output of $x \cdot H$ is called *syndrome*. Hence for any $c \in \mathcal{C}$, the corresponding syndrome $c \cdot H$ is equal to the zero vector of $n-k$ bits.

The generator matrix G of a linear systematic $[n,k]$-code is of the form $G = [I_k|P]$ with I_k being the identity matrix of size k, while the parity bits are generated using a $k \times (n-k)$ matrix P as $x' = x \cdot P$. Then for any $(x\|x') \in \mathbb{F}_2^n$, the syndrome can be computed by $x \cdot P \oplus x'$. Using the value of the syndrome, it is possible to detect some of the erroneous codewords, in a way that if it is a zero vector then the input is a possible codeword, otherwise, it is not. Indeed, the error detection and correction capability of any code depend on its minimum distance, i. e., larger d allows more errors to be detected or corrected.

Definition 46 (Minimum Distance)**.** The minimum distance d of a $[n,k]$-code C is defined as

$$d = \min_{\substack{c,c' \in C \\ c \neq c'}} \mathrm{hw}(c \oplus c'),$$

recalling that hw denotes the Hamming weight.

An $[n,k]$-code with minimum distance d is denoted as an $[n,k,d]$-code.

Lemma 47. *For a linear $[n,k]$-code, we have $d = \min_{e \in \mathcal{C} \setminus \{0\}} \mathrm{hw}(e)$.*

In a setting in which faults are modeled by an n-bit additive error vector e (i. e., the faulty codeword $\tilde{c}$ can be written as $c \oplus e$), a fault is definitely detectable if $\mathrm{hw}(e) < d$, or it is correctable if $\mathrm{hw}(e) < d/2$. It is important to mention that this does not mean both correcting the faults with $\mathrm{hw}(e) < d/2$ and detecting the other faults with $\mathrm{hw}(e) < d$ is possible. It can only detect faults with $\mathrm{hw}(e) < d$ or only correct the faults with faults with $\mathrm{hw}(e) < d/2$. Besides, this capability of detecting and correcting holds for any $[n,k,d]$-codes and it is not necessary to be a linear code. We formalize this for linear codes in the following lemma.

Lemma 48. *A linear $[n, k, d]$-code C can detect erroneous codewords $\tilde{c} = c \oplus e$ if and only if $e \notin C$. In particular, all non-zero error vectors e with $wt(e) < d$ are detectable. Besides, this code can correct all erroneous codewords $\tilde{c} = c \oplus e$ with $wt(e) < d/2$.*

The proof of the above lemma can be found in most of the literature on coding theory (e. g., [MS77]), and they are not recalled here again.

[Bla03] Blahut, *Algebraic Codes for Data Transmission*

As noted in [Bla03], the focus on systematic linear codes does not bring any restriction, since any linear non-systematic code can be transformed into a systematic code with the same minimum distance.

Syndrome Decoding is an efficient method of error correction in linear codes. The underlying principle idea is the linearity of the code. Assuming a systematic linear code, for a codeword $(x \| x')$, we have $x' \oplus x \cdot P = 0$. Then, for an erroneous codeword $(x \oplus e \| x' \oplus e')$, the syndrome is equal to $e'' = (x' \oplus e') \oplus (x \oplus e) \cdot P = e' \oplus e \cdot P$. Using a proper look-up table, one can map all syndrome values to an error vector $(\tilde{e} | \tilde{e}')$ with $\tilde{e}' \oplus \tilde{e} \cdot P = e''$. Such a look-up table $e'' \mapsto (\tilde{e} | \tilde{e}')$ is called a *syndrome decoder*.

By adding the syndrome's error $(\tilde{e} | \tilde{e}')$ to the erroneous codeword, it is possible to correct the codeword. In this way, output of addition is $(x \oplus e \oplus \tilde{e} \| x' \oplus e' \oplus \tilde{e}')$ and if the occurred error $(e \| e')$ had the same value as the syndrome's error $(\tilde{e} | \tilde{e}')$, the output is the same as correct codeword $(x \| x')$. Since, in coding channels, occurrence probability of errors with smaller Hamming weight value is higher, for a given syndrome value, the syndrome's error is chosen from the ones that has the minimum possible Hamming weight, i. e., $\mathrm{hw}(\tilde{e}) + \mathrm{hw}(\tilde{e}') < d/2$.

It is noteworthy that the mapping in the syndrome decoder can be chosen differently in other applications than in the coding, and it is much related to the distribution of the errors.

Example 49 (Duplication)**.** Duplication is indeed a systematic linear code with the generator matrix being the identity matrix, $G = [I_k | I_k]$, and representing the codewords as $(x \| x)$. Such a code cannot detect the symmetric faults $(\delta \| \delta)$ and has a minimum distance of $d = 2$.

Example 50 (Parity Code)**.** In a parity code, x' is only one bit and is the Hamming parity of the message x. Since the redundancy is only one bit, the required extra logic is relatively small. This leads to the capability of detecting any faults with an odd Hamming weight.

Example 51 (Multiple Executions)**.** Another common approach to CED schemes is implementing the target algorithm multiple times (by either time or area redundancy) [Kar+02; Guo+15]. It corresponds to a $[\lambda \cdot k, k, \lambda]$-code where λ also denotes the number of executions of the target algorithm (e. g., when $\lambda = 2$ the target algorithm runs twice, i. e., duplication). The error-detecting capability can be straightforwardly improved by increasing λ at the cost of multiplying the overhead by λ.

Using linear codes has another advantage; the algebraic degree of functions in the redundant part of the scheme A' stays the same as that of A. This is beneficial when the implementation should also be protected against

side-channel analysis attacks by a masking scheme such as a threshold implementation. On the other hand, in some articles, it is proposed to use non-linear codes to improve the fault coverage [KKT04; GSK06; KKT07; KWK08; Akd+12] with the disadvantage of higher area overhead. However, their benefits over linear codes are questionable in some scenarios depending on the assumed fault model and overhead [MSY06]. Therefore, the focus of this thesis is on linear codes.

[KKT04] Karpovsky, Kulikowski, and Taubin, "Robust Protection against Fault-Injection Attacks on Smart Cards Implementing the Advanced Encryption Standard"

[GSK06] Gaubatz, Sunar, and Karpovsky, "Non-linear Residue Codes for Robust Public-Key Arithmetic"

[KKT07] Kulikowski, Karpovsky, and Taubin, "Robust codes and robust, fault-tolerant architectures of the Advanced Encryption Standard"

[KWK08] Kulikowski, Wang, and Karpovsky, "Comparative Analysis of Robust Fault Attack Resistant Architectures for Public and Private Cryptosystems"

[Akd+12] Akdemir, Wang, Karpovsky, and Sunar, "Design of Cryptographic Devices Resilient to Fault Injection Attacks Using Nonlinear Robust Codes"

Part II

HARDWARE-ORIENTED DESIGNS

3

Fault Detection and Correction Countermeasures

"You cannot pass." —Gandalf saying to Balrog in "The Lord of the Rings: The Fellowship of the Ring" by J. R. R. Tolkien.

FAULT ANALYSIS poses a serious threat to cryptographic hardware implementations. As it is discussed in Section 2.4.4, the means to inject faults have improved over the last years and are becoming increasingly powerful. A promising countermeasure to protect against these attacks is employing EDC-based CED schemes.

In Section 3.1, we present a methodology that explains how EDC-based CED schemes must be implemented. Precisely, it is explained how to prevent the matter of *fault propagation*. Besides, an adversary model is defined, which can inject faults at a bounded number of cells at any circuit location. Later, in Section 3.2, we present an implementation strategy based on the underlying EDC, which guarantees detection of up to a certain number of faults.

Sections 3.1 and 3.2 are mainly based on article [Agh+20]. The extended version of this article can be found in [Agh+18] that the author of the thesis wrote together with Anita Aghaie, Amir Moradi, Aein Rezaei Shahmirzadi, Falk Schellenberg, and Tobias Schneider. Note that except Section 3.1, the parts of these articles that the author of the thesis has not contributed, are not included in this thesis. The author has not contributed to Section 3.1, but since it is the basic and concept of the following sections, it is added to this thesis by permission of co-authors. Besides, as mentioned in Sections 3.2.3 and 3.2.4, the author has not contributed to the protection of multiplexers, checkpoints, and registers and extending to a multivariate model in a CED scheme. It is important to mention that the author has not contributed to any of the hardware implementations, but to illustrate the results of the theoretical contributions, they are borrowed from this article.

[Agh+20] Aghaie, Moradi, Rasoolzadeh, Shahmirzadi, Schellenberg, and Schneider, "Impeccable Circuits"

[Agh+18] Aghaie, Moradi, Rasoolzadeh, Shahmirzadi, Schellenberg, and Schneider, *Impeccable Circuits*

In Section 3.3, the tweakable block cipher CRAFT is presented. The efficient protection of its implementations against DFA has been one of the main design criteria. This section is based on article [Bei+19] that the author of the thesis wrote together with Christof Beierle, Gregor Leander, and Amir Moradi.

[Bei+19] Beierle, Leander, Moradi, and Rasoolzadeh, "CRAFT: Lightweight Tweakable Block Cipher with Efficient Protection Against DFA Attacks"

Since most of the previously introduced countermeasures against fault analysis do not provide protection against SIFA, in Section 3.4, we introduce a methodology which shows how to apply ECCs to protect against this attack.

The proposed Concurrent Error Correction (CEC) scheme guarantees the correction of faults up to a number of faults, in any location of the circuit, and at any clock cycle, as long as they fit into the underlying adversary model. That means the CEC scheme protects SIFA as long as the number of injected faults does not exceed the implementation's bounded model. Section 3.4 is based on article [SRM20] that the author of the thesis wrote together with Amir Moradi and Aein Rezaei Shahmirzadi.

[SRM20] Shahmirzadi, Rasoolzadeh, and Moradi, "Impeccable Circuits II"

Again, we emphasize that the author of the thesis has not contributed to any of the hardware implementations, but to illustrate the results of the theoretical contributions, they are borrowed from the corresponding articles.

3.1 CONCEPT AND METHODOLOGY

As stated before in Section 2.4.4, most of the CED schemes rely on systematic linear codes. The effectiveness of EDCs relies on the specific parameters of the underlying code, i. e., length, rank, and distance.

Previously, most of the proposed CED schemes are evaluated theoretically in the *uniform fault model* [Ber+03] or by practical experiments with limited capabilities of the evaluator [Pat+15]. In the uniform model $\mathbf{M}_{Uni}$, it is assumed that the non-zero error vector $e \in \mathbb{F}_2^n$ follows a uniform distribution, i. e., one specific fault occurs with probability $(2^n-1)^{-1}$,[1] based on which the fault coverage is evaluated. Considering such an adversary model, the fault detectable/correctable coverage depends only on the code length and the rank.

[Pat+15] Patranabis, Chakraborty, Nguyen, and Mukhopadhyay, "A Biased Fault Attack on the Time Redundancy Countermeasure for AES"

[1] Note that only non-zero faults are considered.

Lemma 52 (Fault Coverage in the Uniform Model)**.** *For an error detection scheme* $\mathbf{C}$ *using a linear* $[n,k]$*-code and assuming uniform distribution through all faults,* detectable *fault coverage is always*

$$Cov_{Det.}(\mathbf{C}, \mathbf{M}_{Uni}) = \frac{2^n - 2^k}{2^n - 1}. \tag{3.1}$$

While for an error correction scheme using the same linear $[n,k]$*-code and uniform distribution through all faults,* correctable *fault coverage is always*

$$Cov_{Cor.}(\mathbf{C}, \mathbf{M}_{Uni}) = \frac{2^{n-k} - 1}{2^n - 1}. \tag{3.2}$$

Proof. Using the explanations in Section 2.4.4, the proof is quite simple. Through all 2^n-1 possible faults, the corresponding syndrome of 2^k-1 faults will map to zero vector and will not be detectable. But the other 2^n-2^k faults will be detected.

The error correction scheme, since the syndrome decoder receives $n-k$ bits, can correct at most 2^{n-k} faults. But it must map the zero input to zero output; hence it can correct only $2^{n-k}-1$ non-zero faults. □

Since both detectable and correctable fault coverage of a linear code is independent of the code minimum distance d, every code with a constant length and rank provides the same fault coverage under the uniform model $\mathbf{M}_{Uni}$, e. g., an $[8,4,2]$-code and an $[8,4,4]$-code both have a detectable fault coverage of $(2^8-2^4)/(2^8-1) \approx 0.94$ and correctable fault coverage of $(2^4-1)/(2^8-1) \approx 0.06$.

Therefore, the uniform fault model assumes a relatively weak adversary without much control over the fault injection process. In practice, most of the fault distributions contain a specific bias that can increase a potential attack's success rate.

In the following, first, we introduce an adversary model with control over the fault injection process. Note that this adversary model is used in the whole of the current chapter. Then, we explain the fault propagation problem in the code-based CED schemes, and later we discuss the choices for preventing this problem.

3.1.1 *Adversary Models*

We assume a similar adversary model to [Ish+06], i. e., the computation of the circuit is partitioned in clock cycles, and the adversary can adaptively make t wires faulty (toggle) per clock cycle. Each wire is the output of a cell (either a combinatorial gate or a register cell). Hence, we model every fault on the outputting wire on the corresponding cell.

[Ish+06] Ishai, Prabhakaran, Sahai, and Wagner, "Private Circuits II: Keeping Secrets in Tamperable Circuits"

Definition 53 (Univariate Adversary Model $\mathbf{M}_t$)**.** In a given sub-circuit, the adversary can make up to t cells faulty in the algorithm's entire operation, e. g., full encryption. t can be split into various clock cycles.

Definition 54 (Multivariate Adversary Model $\mathbf{M}_t^*$)**.** Here, the $\mathbf{M}_t$ adversary model is extended to allow the attacker to inject such bounded faults at every clock cycle, i. e., the attacker can target up to t cells at every clock cycle.

Definition 55 (Check Point)**.** A check point monitors the correctness of the state of the circuit at a specific point in the computation. In fact, for any $x \in \mathbb{F}_2^k$ and $x' \in \mathbb{F}_2^{n-k}$, it computes the corresponding syndrome for $(x \| x')$ (i. e., $x' \oplus x \cdot P$) and checks if it is equal to zero vector.[2]

[2] Recalling that P is the parity matrix of the underlying code.

Furthermore, the checkpoints' frequency and position inside the circuit strongly affect its security and efficiency. While too frequent checks will increase the area overhead and the design's critical path, a missing check point at an important position could significantly reduce the fault coverage. Moreover, the module responsible for the consistency check is an attractive target for an adversary and needs to be implemented with care.

It is noteworthy to mention that considering this kind of adversary model, similar attacks to the Safe Error ones are excluded. Such attacks are based on forcing a particular internal state to a known value, e. g., `0`. If no fault is observed, the adversary concludes that the value was already `0`.

In this context, it is important to mention that the proposed methodology provides a sound basis not only for error detection but also to incorporate error correction in the circuit while still providing robustness against fault propagation. Since the approach already allows us to compute encoded values correctly, it would require only an additional correction module which needs to be implemented robustly.

In the following, the main focus is on the univariate model, and the techniques to construct a circuit providing full fault coverage are presented.

Afterward, a solution to turn a $\mathbf{M}_t$-secure circuit to its $\mathbf{M}_t^*$-secure variant is explained.

3.1.2 *Fault Propagation*

If an input of a gate in the circuit is faulty, depending on the gate's type and the value of the other inputs to the gate, its output might be faulty. This phenomenon is propagated through the circuit, and as a result, a $\mathbf{M}_t$-bounded adversary can achieve t' faulty cells, which $t' \geq t$. A formal definition of fault propagation is given in the following assumption.

Assumption 56 (Fault Propagation)**.** *We assume the worst case for fault propagation, i. e., one faulty input of a gate results in a faulty output. Hence, in the presence of fault propagation, a $\mathbf{M}_t$-bounded adversary can achieve t' faulty gates with $t' \geq t$.*

It is noteworthy to mention that the value of $t'-t$ is much related to the number of output bits of the fault-injected sub-circuits and the number of gates infected by fault propagation.

For CED schemes, fault propagation can result in lower fault coverage. This has serious implications for the security of code-based CED schemes as the underlying code's distance does not provide a reliable bound for t anymore.

Example 57 (Parity Code with Fault Propagation)**.** To illustrate the threat of fault propagation for CED schemes, we examine a basic parity scheme's susceptibility. Assume a CED scheme $\mathbf{C}_{Parity}$ using $[4,3,2]$ parity code, with a generator matrix

$$G_{Parity} = (I_3|P) = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix}.$$

Consider a circuit realizing the function $T : \mathbb{F}_2^3 \mapsto \mathbb{F}_2^3$, which is supposed to be protected by such a CED; as an example, a linear function with a corresponding matrix of L given below.

$$L = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

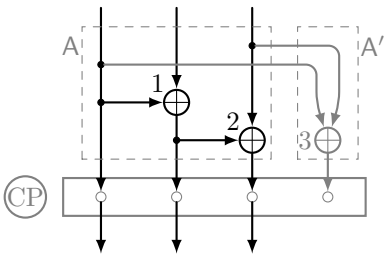


FIGURE 3.1: Example for Fault Propagation.

Such a circuit protected by the $[4,3,2]$-code (including A and A′) is depicted in Figure 3.1 consisting of three XOR gates. A realizes $T(x) = x \cdot L$ and A′ is formed by $x \cdot L \cdot P$. The check point is also placed at the output of the circuit that is denoted by (CP). In the uniform fault model, based on Eq. (3.1), its fault coverage is

$$Cov_{Det.}(\mathbf{C}_{Parity}, \mathbf{M}_{Uni}) = \frac{2^4 - 2^3}{2^4 - 1} = 0.53,$$

meaning that the countermeasure can detect around 53% of all injected faults. As stated before, the uniform fault model has a relatively low relevance in practice. Therefore, we compute the fault coverage under a $\mathbf{M}_t$

adversary model. Since the code has a distance of $d = 2$, we restrict the adversary to maximum $t = d-1 = 1$ fault injections, i. e., $\mathbf{M}_{t=1}$. Without fault propagation, the code is indeed able to detect all possible faults injected at the gates whose output is exclusively connected to the check points (e. g., XOR gates 2 and 3 in Figure 3.1). However, in the presence of fault propagation, there is a possibility that the adversary can increase the number of faulty gates and thus create a faulty state that is not detectable at the check points. In Figure 3.1, if the adversary makes the XOR gate 1 faulty, then it propagates through the XOR gate 2, and two faulty values arrive at the following check point, creating a state that is not detectable by parity, i. e., the weight of the error vector is 2. This results in

$$Cov_{Det.}(\mathbf{C}_{Parity}, \mathbf{M}_{t=1}) = \frac{2}{3}.$$

In the following, several solutions are presented to provide full fault coverage under a $\mathbf{M}_t$ adversary model.

3.1.3 *Extra Check Points*

One strategy to restrict the fault propagation is including extra check points in the circuit. The intuition behind this is very basic. By splitting the circuit into smaller sub-circuits divided by check points, this effect can be damped assuming each sub-circuit contains fewer gates than the whole design. Thereby, the effect of fault propagation is limited, since a detectable faulty state cannot traverse over a check point.

An interesting question is at which points in a circuit the extra check points need to be inserted to achieve the desired level of security. An approach to reduce the effect of fault propagation for a given function T is a decomposition into multiple sub-functions as $T = T_l \circ \ldots \circ T_1$ with a check point between every T_i and T_{i+1}. This approach limits the fault propagation if each of the sub-functions is less sensitive to fault propagation.

To measure the sensitivity to fault propagation, it can be trivially seen that the fault cannot be propagated if the circuit realizing a sub-function T_i has a depth of 1, i. e., there is no gate in the underlying sub-circuit whose input is derived from another gate of the same sub-circuit. Therefore, to completely remove fault propagation independent of the target function, it is necessary to include a check point after every circuit depth in the combinatorial logic as noted in Lemma 58.

Lemma 58 (Preventing Fault Propagation with Extra Check Points)**.** *Assume that the function $T : \mathbb{F}_2^k \mapsto \mathbb{F}_2^k$ is decomposable into multiple sub-functions as $T = T_l \circ \ldots \circ T_1$ in such a way that each T_i is a mapping from $\mathbb{F}_2^k$ to itself and its gate depth is 1. Also gate depth of each function T_i', the function to compute parity of the output for T_i (if y and y' are the output for functions T_i and T_i', resp., then $y' = y \cdot P$), is 1. Then fault propagation can be completely prevented by inserting a check point at the output wires of the corresponding sub-circuits for T_i and T_i'.*

Proof. The proof is straightforward. Since each T_i and T_i' have a gate depth of 1, there will be no fault propagation inside their sub-circuits. And since

the state at each check point has to be a valid codeword under the employed code, all detectable faults by the code will be detected. And propagation from one detectable fault to an undetectable fault will be prevented. Therefore, one faulty gate can only result in at most one faulty wire at the following check point. □

Note that this strategy is very generic and can be applied independently of the target algorithm. It is only necessary to ensure that the state at every check point is verifiable, i. e., a valid codeword.

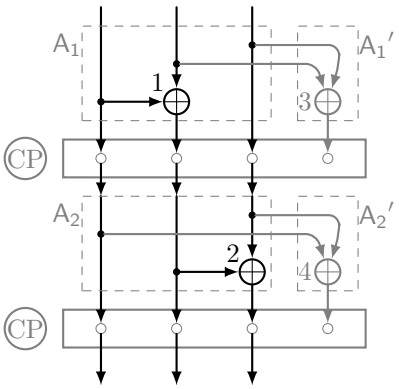


FIGURE 3.2: Example for Using Extra Check Points to Prevent Fault Propagation.

Example 59 (Parity Code with Extra Check Points)**.** To prevent fault propagation in the function T from Example 57 using extra check points, one solution is shown in Figure 3.2. The inclusion of an extra check point in the middle of the circuit successfully prevents harmful fault propagation and ensures the error-detection capability for $t = 1$. The linear function T is decomposed to two linear functions T_1 and T_2, with the following corresponding L_1 and L_2 matrices, resp., i. e., $T = T_2 \circ T_1$ and $L = L_1 \cdot L_2$.

$$L_1 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix} \qquad L_2 = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

For $i \in \{1, 2\}$, A_i realizes $T_i(x) = x \cdot L_i$ and $\mathsf{A'}_i$ is formed by $x \cdot L_i \cdot P$.[3] Excluding the extra check point, this comes at the cost of one additional XOR. Nevertheless, the new design provides the desired fault coverage of 1 for the $\mathbf{M}_{t=1}$ adversary model.

[3] Note that P is the corresponding parity matrix for the parity code.

3.1.4 *Independence Property*

Another possibility to achieve security against a $\mathbf{M}_t$-bounded adversary is based on the independence property of the implementation. For this, the circuit needs to be split up into independent sub-circuits that each computes exactly *one* output bit.

Definition 60 (Independence Property of Implementation)**.** Let us assume the target function for implementing is $T : \mathbb{F}_2^k \mapsto \mathbb{F}_2^{k'}$ which maps k-bit input x to a k'-bit output y. The function T is physically realized by k' sub-circuits that each of them realizes a coordinate function $T[i]$ with $i \in \mathbb{Z}_{k'}$.[4] Such a set of sub-circuits follows the *Independent Property* if no gate is shared between every two sub-circuits. In more detail, if $\mathcal{G}_i$ denotes the sub-circuit for $T[i]$, then for any unequal i and j, there is no gate that is shared between $\mathcal{G}_i$ and $\mathcal{G}_j$.

[4] Recalling that for any i-th bit of output y we have $y[i] = T[i](x)$.

The concept of preventing fault propagation in an implementation with independence property is formalized in the following lemma.

Lemma 61 (Preventing Fault Propagation with Independence Property)**.** *For a function $T : \mathbb{F}_2^k \mapsto \mathbb{F}_2^{k'}$, an implementation realized by independence property defined as in Definition 60 does not suffer from fault propagation.*

Proof. Based on the assumption that the sub-circuits (each of which realizing a coordinate function $T[i] : \mathbb{F}_2^k \mapsto \mathbb{F}_2$) are distinct and do not share any

gates, it is not possible for a faulty wire in $T[i]$ to traverse to $T[j]$ that $j \neq i$. Therefore, one faulty gate can *maximally* affect one output wire of T since each $T[i]$ computes only one unique output bit of T. This trivially implies that any t faulty gates in the entire set of sub-circuits can make *at most* t output wires of T faulty. □

This strategy to thwart fault propagation is very simple to implement for a given function T by instantiating completely independent sub-circuits for every output bit. Since the hardware synthesizers usually optimize the given design (e. g., by sharing the identical coordinates to achieve a lower area footprint), particular attention should be paid to avoid such optimizations.[5] Otherwise, the resulting circuit may merge the sub-circuits, which are supposed to be independent.

[5] It can be done by instantiating a unique sub-circuit for each coordinate function, and forcing the synthesizer to keep the hierarchy.

Example 62 (Parity Code with Independence Property)**.** For the T function defined in Example 57, the strategy of independence property can be applied. Considering A and A′ as one function $T^* : \mathbb{F}_2^4 \mapsto \mathbb{F}_2^4$, based on Lemma 61, it is required to implement T^* as four distinct sub-circuits which do not share any gates. The resulting design is depicted in Figure 3.3. Similar to the previous approach with an extra check point, the design requires one more XOR gate. However, forced independence suffices with only one check point, which makes it more efficient than implementing it as in Example 59, while providing the same full fault coverage for the $\mathbf{M}_{t=1}$ adversary model.

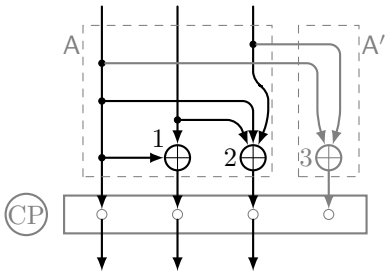


FIGURE 3.3: Example for Using Indepndence Property to Prevent Fault Propagation.

3.1.5 *Combination*

Each aforementioned solution comes at the cost of a higher area. Frequent check points require additional combinatorial logic to ensure that the state is always a valid codeword. Depending on the underlying function, implementing with independence property may require even more logic since ordinary optimizations (i. e., reuse of the common gates) are not allowed. Therefore, for a given target function $T : \mathbb{F}_2^k \mapsto \mathbb{F}_2^k$, a hybrid approach in three steps is proposed in the following.

1. *Decompose* $T : \mathbb{F}_2^k \mapsto \mathbb{F}_2^k$ *into multiple sub-functions* $T_i : \mathbb{F}_2^k \mapsto \mathbb{F}_2^k$ *of a lower complexity in a way that is explained in Lemma 58.*

Such a specific decomposition depends on the target function T. Considering an SPN block cipher, each sub-function T_i reflects the operations within a round of the cipher (e. g., S-box, linear layer, and key addition).

2. *Split each* T_i *into all its coordinate functions* $T_i[j] : \mathbb{F}_2^k \mapsto \mathbb{F}_2$ *in a way that is explained in Lemma 61.*

That means each sub-function T_i is split into coordinate functions with the same (or probably even more) number of input and only one output bits. A basic split in most block ciphers follows the cipher's structure where a certain function is applied multiple times in parallel, e. g., the S-box on each word, and MixColumns on only some of the words. Step 1 and 2 can be repeated until the desired level of granularity is achieved.

3. *Implement each $T_i[j]$ function fulfilling the independence property and place a check point at their output.*

This step benefits from the small input size of $T_i[j]$ since it allows to reduce the area overhead when the independence property is applied. However, a decomposition (step 1) that is too fine would suffer from the basic problem of frequent check points. Therefore, it is imperative to find a balance between the two strategies adjusted to the target function.

3.2 FAULT DETECTION STRUCTURE

To clarify the application of strategies explained in Section 3.1 for a code-based CED scheme, consider an exemplary algorithm realized by the sequential circuit depicted in Figure 3.4 consisting of a register that loads the INPUT at the start of the operation (triggered by rst signal) and performs the function T repeatedly till the OUTPUT is taken from the register.[6] Note that any sequential circuit can be represented by such a construction. For the sake of simplicity, suppose that the bit-length of T input and output (therefore length for INPUT and register) is k bits. The application of a systematic linear $[n,k,d]$-code would lead to transforming the k-bit x to an n-bit codeword $c=(x\|x')$. Hereafter, we refer to the application of matrix P on x to derive the redundant part x' by function $F:\mathbb{F}_2^k \mapsto \mathbb{F}_2^p$ as $x'=F(x)=x\cdot P$, where $p=n-k$ denotes the bit-length of the redundancy. In the following, the function F is distinguished into two different cases, and it is explained how to apply the underlying EDC.

[6] Note that the Finite State Machine (FSM) is not shown.

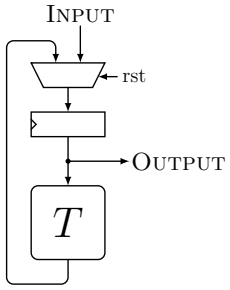


FIGURE 3.4: Original Structure as a Target for CED.

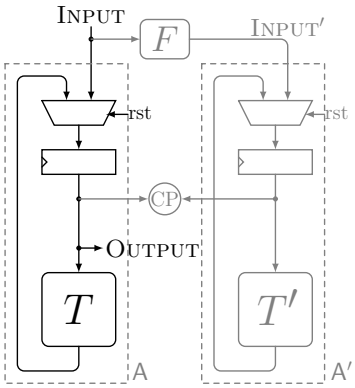


FIGURE 3.5: Proposed EDC-based CED scheme using an Injective F.

3.2.1 *The Parity Function and the General Structure*

Depending on if function F is injective or not, there are two possibilities to implement CED structure.

Injective F

If the parity function is injective, then the redundancy size is at least as large as the message size, i. e., $p \geq k$. Then, the redundancy part of the circuit (noted beforehand by A′) can operate on x' independent of x, as it is shown in Figure 3.5. For any arbitrary function T, the redundant function T' can be achieved by fulfilling $T' \circ F = F \circ T$. Both T and T' are implemented following the independence property. Note that T' cannot be implemented as separate decomposed functions F^{-1}, T, and F. Instead, its description should be first derived (as given above) and then implemented by independent coordinate functions. As the last step, a single check point is placed at the input of the T function (marked by (CP) in Figure 3.5).

An arising question is whether it is essential to place the check point at the input of T. Otherwise, if the check points are moved to the output of T, the faults injected at the register cells would potentially propagate to multiple output bits of T. Hence, to maintain full fault coverage against a $\mathbf{M}_{t=d-1}$-bounded adversary, the check point should be placed right at the input of the function implemented by the independence property. It should be noted that since the multiplexer and the register (see Figure 3.5) independently operate

on each bit of T output, they do not affect the independence property of the entire circuit (i. e., from the T input forward to the register output). Therefore, any fault injected at T fitting to $\mathbf{M}_{t=d-1}$, is detected at the check point in the next clock cycle.

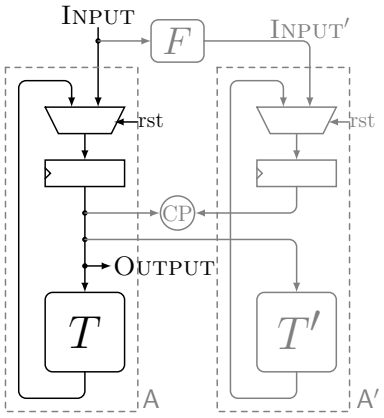


FIGURE 3.6: Proposed EDC-based CED scheme using a Non-injective F.

Non-injective F

For the cases that smaller redundancy is desired, $p < k$, (for lower area overheads), the function F cannot be injective. Generally, when F is not injective, the construction shown in Figure 3.5 is not necessarily applicable. It indeed depends on the specification of the underlying function T. For an arbitrary T, it is not always possible for T' to solely operate on x'. In such cases, T' needs to receive the original data x to compute $T'(x) = F \circ T(x)$. The corresponding construction is shown in Figure 3.6, where the only difference to the former case is how the T' is realized.

For some particular functions T, it is still possible to realize the circuit similar to the one shown in Figure 3.5, i. e., T' can operate only on x' while F is not injective. Any intermediate value of the circuit should be a valid codeword. That means if $x \in \mathbb{F}_2^k$ and $x' \in \mathbb{F}_2^p$ are the input of T and T', resp., with $x' = F(x)$, their output $\big(T(x) \| T'(x')\big)$ should also form a valid codeword. This implies that $T'(x') = F \circ T(x)$ which equally means

$$T' \circ F = F \circ T.$$

Given T and F, it can be examined if there exists such a function T' fulfilling the above condition. In many cases, especially in SPN block ciphers, the linear layer uses multiplication in $\mathbb{F}_{2^k}$ with a *primitive* element $a \in \mathbb{F}_2^k$, i. e., $T : \mathbb{F}_2^k \mapsto \mathbb{F}_2^k$ in such a way that $T(x) = \psi_k^{-1}\big(a \times \psi_k(x)\big)$ where that ψ_k is the bijective mapping from $\mathbb{F}_2^k$ to $\mathbb{F}_{2^k}$. The following lemma shows that for such primitive elements a, there exists no such a function T' fitting to $T' \circ F = F \circ T$.

Lemma 63. *Let $T : \mathbb{F}_2^k \mapsto \mathbb{F}_2^k$ be the corresponding Boolean function for multiplication in $\mathbb{F}_{2^k}$ with a primitive element a of this field. And let $F : \mathbb{F}_2^k \mapsto \mathbb{F}_2^p$ be any linear function with $p < k$. Then there is no $T' : \mathbb{F}_2^p \mapsto \mathbb{F}_2^p$ such that $F \circ T = T' \circ F$.*

Proof. Since F is a linear function, among its inputs, there exist 2^c (with $c \geq k - p$) values that are mapped to zero. This means that there are $2^c - 1 \geq 1$ non-zero roots for F. Let u_0 be one of these non-zero roots. Now assume that there exists a T' function which $F \circ T = T' \circ F$. It is clear that as both F and T are linear functions, T' is also a linear function. This results to $T'(0) = 0$. Hence, for such u_0, we have

$$F \circ T(u_0) = T' \circ F(u_0) = T'(0) = 0.$$

This means $u_1 := T(u_0)$ is another non-zero root of F. By repeating above equation, we find out that for any $i > 0$, $u_i := T^i(u_0)$ is a non-zero root for F. Since T is the corresponding Boolean function for multiplication in $\mathbb{F}_{2^k}$ and a is a primitive element, all u_i and u_j with $0 \leq i < j < 2^k - 1$ are different. Hence, we have $2^k - 1$ non-zero roots for F which means that any

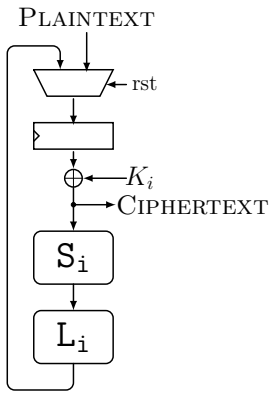


FIGURE 3.7: Round-Based Implementation of an SPN Block Cipher.

[7]Note that this does not mean the proposed CED scheme is only applicable to the round-based implementation. As explained in [Agh+20; Agh+18], it is also applied to the nibble-serialized implementation of some ciphers.

[8]Note that here, n is the message length, not the length for the codeword.

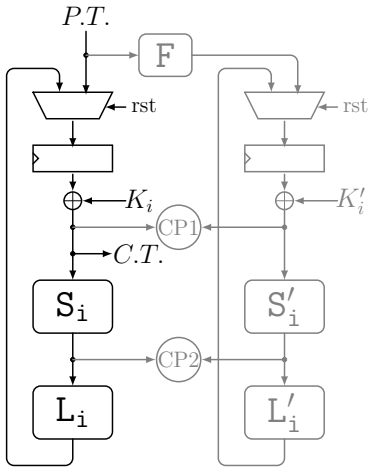


FIGURE 3.8: Proposed CED Scheme of an SPN Block Cipher using Injective F.

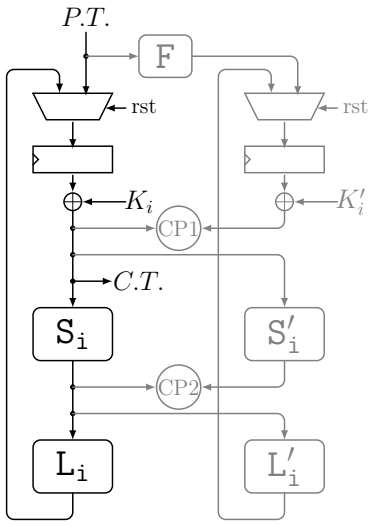


FIGURE 3.9: Proposed CED Scheme of an SPN Block Cipher using Non-injective F.

[9]Even more beneficial, if both of the input bits are faulty, then the output bit is fault-free.

$x \in \mathbb{F}_2^k$ is a root for F, i. e., for any $\forall x$, $F(x) = 0$. And this is in contrast with the assumption that F is an arbitrary linear function. □

Although there is no solution with a being a primitive element, it has trivial solutions for $a = 0$ or $a = 1$. It means that if T is the corresponding Boolean function for multiplication with $a \in \{0, 1\}$ in $\mathbb{F}_{2^k}$, the redundant counterpart T' is also the corresponding Boolean function for multiplication with the same constant in $\mathbb{F}_{2^p}$.

3.2.2 *Application for an SPN cipher*

Consider an n-bit SPN block cipher as defined in Definition 6 and its round-based implementation[7] shown in Figure 3.7. The goal is to protect its implementation against a $\mathbf{M}_{t=d-1}$ adversary model using an $[n+p, n, d]$-code.[8] Decomposing the round functions $\mathtt{R}_i$ to its operations (i. e., $\mathtt{A}_{K_i}$, $\mathtt{S}_i$ and $\mathtt{L}_i$), it is possible to apply the proposed CED scheme to whole encryption implementation by using the CED scheme for each of the operations ($\mathtt{A}_{K'_i}$, $\mathtt{S}'_i$ and $\mathtt{L}'_i$, resp.). In this case, generally, after each operation, we need to insert a check point. Some possibilities to reduce the number of check points and reduce the implementation overhead are discussed in the following.

Key Addition

$\mathtt{A}_{K'_i}$ computes the parity of the output for $\mathtt{A}_{K_i}$ operation that the only condition to follow the CED scheme is that $K'_i = F(K_i)$. Note that independent of the injective-ness of F, this operation can be independently operated on x' and does not require access to the whole x. Key addition, by its nature, follows the independence property and does not propagate the fault to more output bits.[9] So there is no need to insert one check point before and one after the key addition. Without any loss in the fault coverage of the scheme, remove the one before $\mathtt{A}_{K_i}$.

Depending on if F is injective or not, Figures 3.8 and 3.9 depict the proposed CED scheme to protect the target SPN block cipher. Note that a similar CED scheme also protects the key (or tweak or tweakey) schedule like the one for round operations, but for simplicity, together with the FSM and its protection, are not shown in the figures.

S-box Layer

Since the S-box layer $\mathtt{S}_i$ is a non-linear function, using codes with a corresponding dense P matrix (e. g., the codes with minimum possible code or parity length for given message length and distance) ends up with an $\mathtt{S}'_i$ function such that each output bit is a (non-linear) function of (usually) all the n input bits. And since the implementation must follow the independence property, this means comparing to $\mathtt{S}_i$, implementation of $\mathtt{S}'_i$ will be slower and probably larger.

Since $\mathtt{S}_i$ is split to m number of s-bit S_{ij} S-boxes, it comes to mind to use a code that is also divided into m words and applies a smaller code to each word. This means the $[n+p, n, d]$-code for the whole n-bit state is applying m times smaller code of $[s+q, s, d]$ for each of the S-boxes in parallel where $m \cdot q = p$. Such a code is denoted by $[s+q, s, d]^m$. Considering that F is

the function to compute parity of each $[s+q,s,d]$-code, the one to compute parity of the $[s+q,s,d]^m$-code, is denoted by F that for any $x \in \mathbb{F}_2^{s\cdot m}$ we have

$$\mathrm{F}(x) = \phi_{s,m}\Big(F\big(\phi_{s,m}^{-1}[0](x)\big),\ldots,F\big(\phi_{s,m}^{-1}[m-1](x)\big)\Big).$$

Using an $[s+q,s,d]^m$-code costs an $m\cdot q$-bit parity and this can be comparably larger with respect to the parity size of the code with minimum possible parity size for given message length and distance. Even though $[s+q,s,d]^m$-code has a bigger parity size, using this code helps to improve delay of S'_i (and even probably its implementation area, too). And the reason for this is that now each output bit of S'_i is only function of at most s bits (the same value as the one for S_i).

Note that the code's split, as mentioned above, is one solution to apply CED schemes for the ciphers' non-linear operations. Depending on the desired goals (small implementation area or shorter delays) and the adversary model (that affects the distance of the underlying EDC), and the cipher type, there could be other underlying code choices. But generally, for the SPN block ciphers, the split as mentioned above seems to be a good trade-off.

Linear Layer

Due to the linearity of the underlying code, compared to the S-box layer's case, for the linear layer, it is much easier to compute or find the corresponding possible L'_i linear operation.

Furthermore, as explained in the following theorem, if the linear layer of an SPN block cipher is realized by binary multiplication in $\mathbb{F}_{2^s}$ such as defined in Definition 7, then it is possible to remove one of the check points before or after the linear layer. Note that this kind of linear layer is used in several block ciphers including, MIDORI, SKINNY, and MANTIS.

Theorem 64. *If the linear layer of an SPN block cipher is realized by binary multiplication in $\mathbb{F}_{2^s}$ with representative matrices B_i such as defined in Definition 7, then using an $[s+q,s,d]^m$-code, there is no need to have both of the check points before or after the linear layer (i. e., one of them can be removed).*

Proof. Since each L_i is a binary multiplication in $\mathbb{F}_{2^s}$, we have $\mathrm{F}\circ\mathrm{L}_i = \mathrm{L}'_i\circ\mathrm{F}$ and also L'_i is a binary multiplication in $\mathbb{F}_{2^q}$. Besides, the representative matrix for both L_i and L'_i are the same.

Let $x \in (\mathbb{F}_2^s)^m$ and $x' \in (\mathbb{F}_2^q)^m$ represent the fault-free input state of the L_i and L'_i linear layers, resp., i. e., $x' = F(x)$. Due to the independence property of each function's implementation, any fault injected at t cells of a function can be modeled by a binary additive error vector at its output. Let $e_1 \in (\mathbb{F}_2^s)^m$, $e'_1 \in (\mathbb{F}_2^q)^m$, $e_2 \in (\mathbb{F}_2^s)^m$ and $e'_2 \in (\mathbb{F}_2^q)^m$ denote the corresponding error vectors of injected faults in S_i, S'_i, L_i and L'_i, resp.

Here, it is considered that the check point before the linear layer is removed, and only the one after the linear layer is kept (the one that is after the key addition in the next round).[10] The check point after the next round's key addition at the input state of the linear layer (see Figure 3.10 for the case that F is non-injective) examines indeed the equality of the below equation:[11]

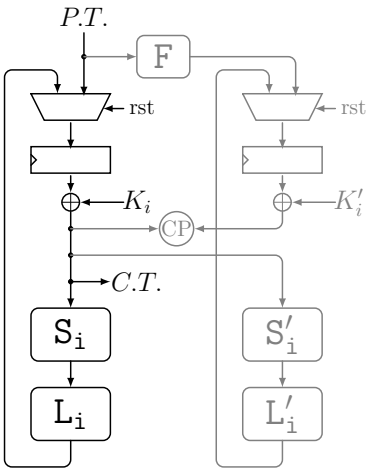


FIGURE 3.10: Proposed CED Scheme of an SPN Block Cipher with Binary Multiplication Linear Layer using Non-injective F.

[10] Note that the proof for the other way around is similar to this case.

[11] Note that as explained in Section 3.2.2, it is considered that the key addition is a fault-free operation.

$$\mathrm{F}\big(\mathrm{L}_i(x \oplus e_1) \oplus e_2\big) \stackrel{?}{=} \mathrm{L}'_i(x' \oplus e'_1) \oplus e'_2,$$

that can be simplified to

$$\mathrm{F} \circ \mathrm{L}_i(x) \oplus \mathrm{F} \circ \mathrm{L}_i(e_1) \oplus \mathrm{F}(e_2) \stackrel{?}{=} \mathrm{L}'_i(x') \oplus \mathrm{L}'_i(e'_1) \oplus e'_2 \quad \Rightarrow$$
$$\mathrm{F} \circ \mathrm{L}_i(e_1) \oplus \mathrm{F}(e_2) \stackrel{?}{=} \mathrm{L}'_i(e'_1) \oplus e'_2$$

By their definition, for each output s-bit and q-bit word of L_i and L'_i, resp., we have

$$\mathrm{L}_i[k](e_1) = \bigoplus_{j=0}^{m-1} b_{i,j,k} \cdot e_1[j], \qquad \mathrm{L}'_i[k](e'_1) = \bigoplus_{j=0}^{m-1} b_{i,j,k} \cdot e'_1[j].$$

Besides, since F is realized by m parallel F functions operating on each s-bit word, the equation for the check point simplifies to checking m times

$$F\Big(\bigoplus_{j=0}^{m-1} b_{i,j,k} \cdot e_1[j]\Big) \oplus F(e_2[k]) \stackrel{?}{=} \bigoplus_{j=0}^{m-1} b_{i,j,k} \cdot e'_1[j] \oplus e'_2[k] \quad \Rightarrow$$
$$F\Big(\bigoplus_{j=0}^{m-1} b_{i,j,k} \cdot e_1[j] \oplus e_2[k]\Big) \stackrel{?}{=} \bigoplus_{j=0}^{m-1} b_{i,j,k} \cdot e'_1[j] \oplus e'_2[k] \tag{3.3}$$

for each k. Considering a $\mathbf{M}_{t=d-1}$ adversary model, suppose that the attacker injects such bounded faults, i. e.,

$$\sum_{j=0}^{m-1} e_1[j] + e'_1[j] + e_2[j] + e'_2[j] < d.$$

To detect such a fault at the check point, the relation in Eq. (3.3) must be unequal for at least one k. That means

$$\sum_{j=0}^{m-1} e_1[j] + e'_1[j] + e_2[j] + e'_2[j] < d \implies$$
$$\exists k \quad F\Big(\bigoplus_{j=0}^{m-1} b_{i,j,k} \cdot e_1[j] \oplus e_2[k]\Big) \neq \bigoplus_{j=0}^{m-1} b_{i,j,k} \cdot e'_1[j] \oplus e'_2[k].$$

This equally means that we need to prove

$$\forall k \quad F\Big(\bigoplus_{j=0}^{m-1} b_{i,j,k} \cdot e_1[j] \oplus e_2[k]\Big) = \bigoplus_{j=0}^{m-1} b_{i,j,k} \cdot e'_1[j] \oplus e'_2[k] \implies$$
$$\sum_{j=0}^{m-1} e_1[j] + e'_1[j] + e_2[j] + e'_2[j] \geq d.$$

To do so, consider $\alpha_k = \bigoplus_{j=0}^{m-1} b_{i,j,k} \cdot e_1[j] \oplus e_2[k]$ and $\beta_k = \bigoplus_{j=0}^{m-1} b_{i,j,k} \cdot e'_1[j] \oplus e'_2[k]$. If for all k values, $\alpha_k = 0$, this implies that for all ks $\beta_k = 0$, then $\mathrm{L}_i(e_1) \oplus e_2 = 0$ and $\mathrm{L}'_i(e'_1) \oplus e'_2 = 0$. It means that the outputs of L_i and L'_i are fault free, hence not useful for the adversary. So without loss of generality, we consider that there exists a k with $\alpha_k \neq 0$ and $\beta_i = F(\alpha_i)$. Since F is the corresponding function to produce parity of an $[s+q, s, d]$-code, we already know that $\alpha_k \in \mathbb{F}_2^s$, $\mathrm{hw}(\alpha_k) + \mathrm{hw}(\beta_k) \geq d$, which implies

that

$$
\begin{aligned}
d &\leq \mathrm{hw}(\alpha_k) + \mathrm{hw}(\beta_k) \\
&= \mathrm{hw}\Big(\bigoplus_{j=0}^{m-1} b_{i,j,k} \cdot e_1[j] \oplus e_2[k]\Big) + \mathrm{hw}\Big(\bigoplus_{j=0}^{m-1} b_{i,j,k} \cdot e_1'[j] \oplus e_2'[k]\Big) \\
&\leq \mathrm{hw}\Big(\bigoplus_{j=0}^{m-1} b_{i,j,k} \cdot e_1[j]\Big) + \mathrm{hw}\Big(\bigoplus_{j=0}^{m-1} b_{i,j,k} \cdot e_1'[j]\Big) + \mathrm{hw}(e_2[k]) + \mathrm{hw}(e_2'[k]) \\
&\leq \sum_{j=0}^{m-1} \mathrm{hw}(b_{i,j,k} \cdot e_1[j]) + \sum_{j=0}^{m-1} \mathrm{hw}(b_{i,j,k} \cdot e_1'[j]) + \mathrm{hw}(e_2[k]) + \mathrm{hw}(e_2'[k]) \\
&= \sum_{j=0}^{m-1} b_{i,j,k} \cdot \mathrm{hw}(e_1[j]) + \sum_{j=0}^{m-1} b_{i,j,k} \cdot \mathrm{hw}(e_1'[j]) + \mathrm{hw}(e_2[k]) + \mathrm{hw}(e_2'[k]) \\
&\leq \sum_{j=0}^{m-1} \mathrm{hw}(e_1[j]) + \sum_{j=0}^{m-1} \mathrm{hw}(e_1'[j]) + \mathrm{hw}(e_2[k]) + \mathrm{hw}(e_2'[k]) \\
&= \mathrm{hw}(e_1[k]) + \mathrm{hw}(e_1'[k]) + \mathrm{hw}(e_2[k]) + \mathrm{hw}(e_2'[k]) \\
&\leq \mathrm{hw}(e_1) + \mathrm{hw}(e_1') + \mathrm{hw}(e_2) + \mathrm{hw}(e_2').
\end{aligned}
$$

And this proves the theorem. □

Since the corresponding linear layer in MIDORI, SKINNY, and MANTIS block ciphers fulfill this condition, it makes it possible to use only one check point for each round and detect all the useful fault injections up to $t = d-1$ points.

3.2.3 *Control Signals, Multiplexers, Check Points and Registers*

In the examples shown above, the FSM is not considered. If the FSM is not protected against fault injections, the adversary can change the control flow and obtains faulty results exploiting the secrets. As a trivial example, independent of the employed EDC scheme on the data-processing part of the circuit, the attacker can force to terminate an encryption process at the first rounds. This leads to having access to the intermediate values in the encryption, and therefore recovering the key.

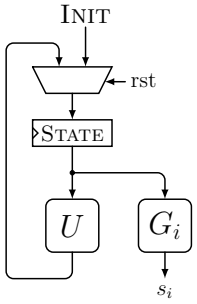


FIGURE 3.11: Structure of an FSM.

The FSM can also be seen as a set of register cells loaded by a particular initialized value INIT, and updated at every clock cycle through an update function U. The content of the register is referred to by STATE. A dedicated function derives each control signal $s_i \in \mathbb{F}_2$ over the FSM register, marked by G_i in Figure 3.11. The application of an $[s+q, s, d]$-code on such a controlling circuit is the same as shown above. The only difference is how the control signals are encoded. Each control signal s_i and its redundant counterpart s_i' are related in the form of $s_i' = F\big(s_i \| (0, \ldots, 0)\big)$, i. e., s_i is padded with zeros to form an s-bit word.[12] That means the redundancy of every control signal has a size of q bits.

[12] Note that the bit size of the control signal can be any value larger than or equal to 1.

In the following, considering that the state bit-size of the FSM is *at most* s bits, it is explained how to protect it by using an $[s+q, s, d]$-code. For state size larger than s bits, using an $[s+q, s, d]^{m'}$, the approach is quite similar.

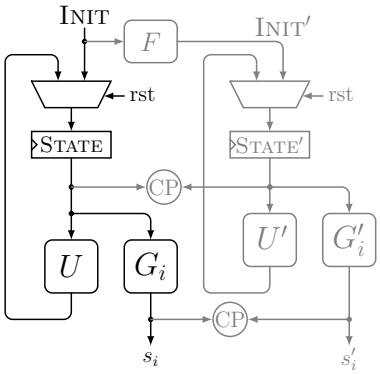


FIGURE 3.12: Application of an EDC on an FSM using Injective F.

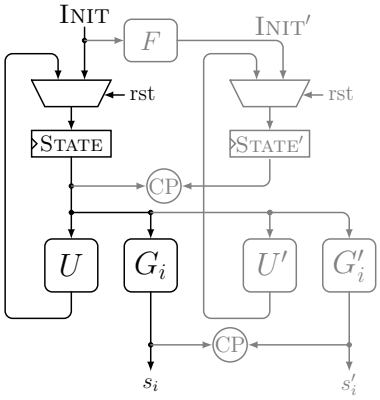


FIGURE 3.13: Application of an EDC on an FSM using Non-injective F.

INJECTIVE F: For an injective F function, the redundant part of the update function realizes $U' \circ F = F \circ U$ and each control signal s_i is mapped to $s'_i = G'_i(\text{STATE}')$ which $G'_i \circ F = F \circ G_i$. Figure 3.12 shows the construction.

NON-INJECTIVE F: In this case, the redundancy update function would operate on STATE as $U' = F \circ U$. The same holds for the control signals as $s'_i = G'_i(\text{STATE})$ while $G'_i = F \circ G_i$ (see Figure 3.13). Note that, depending on G_i it might be possible to generate s'_i over STATE′. To this end, there should exist a G'_i satisfying $G'_i \circ F = F \circ G_i$.

It is important to emphasize that all above-given functions U, U', G_i, and G'_i should be implemented with independence property. As shown in Figures 3.12 and 3.13, the check points are placed at the register output as well as at the output of every function generating a control signal s_i.

As a side note, it is possible to merge a couple of control signals before encoding them, but it usually leads to a more complicated multiplexer controlled by such a redundant control signal. The details about redundant multiplexers are discussed in the following.

MULTIPLEXERS, CHECK POINTS AND REGISTERS: Similar to the control signals, all the multiplexers, registers with enables, and check points must be implemented in a way that protects the CED scheme against a $\mathbf{M}_t$ adversary model. A solution for protecting these elements of the scheme is given in [Agh+20; Agh+18], and since the author of the thesis has not contributed to these parts of the papers, those are not detailed here. As an important result, the multiplexers in the A′ part of the scheme, working with control signal s'_i, will be larger and slower than its corresponding multiplexer in A if parity size q is larger than the length of s_i.

3.2.4 *Extending to Multivariate Model*

Suppose the circuit shown in Figure 3.5 with a $[k+m, k, d]$-code that should detect all single-cell faults. Suppose also that at one clock cycle before the last, the adversary injects a fault in T, which results in a value with a single-bit fault stored in the register. If at the next clock (which is the last one), the adversary injects a single-gate fault on the corresponding F function of the consistency check process, the faulty output can pass the multiplexer and be stored in the final register.

To extend the adversary model to multivariate version and to keep the full fault coverage, it suffices to introduce extra check points right at the registers' input in the design. Independent of the redundancy size, this includes the register of the data processing module and that of the FSM. This makes sure that the consistency of the values stored in the registers is examined. Hence, in the example explained above, the fault is detected at the first check point. Introducing more check points increases the area requirements; furthermore, since they are placed right at the registers' input, the critical path delay of the circuit is increased.

To support the multivariate $\mathbf{M}_t^*$ model, it is also necessary to adjust the consistency check points. In [Agh+20; Agh+18], a solution based on increasing the size of the error registers is given, and since the author of

the thesis has not contributed to this part of the papers, that concept is not detailed here.

3.2.5 *Case Studies*

To assess the new proposed methodology's overhead, in [Agh+20; Agh+18], several cases based on AES, GIFT, LED, MIDORI-64, PRESENT, SIMON-64, and SKINNY-64 block ciphers are examined. All these ciphers, except AES, have a state size of 64 bits. In the following, some details of the implementation are explained. For more information and implementation results, we ask readers to refer to [Agh+20; Agh+18].

To equip the implementation with an EDC, we first need to specify the underlying $[k+p, k, d]$-code. Below the implementations are categorized into three groups where the case for AES is excluded:

- $[q+4, 4, d]^{16}$ with $q < 4$ that implies the case for non-injective F. The $[7, 4, 3]$-code is the well-known Hamming code, and $[5, 4, 2]$-code computes 1-bit parity for each 4-bit word. The remaining code $[6, 4, 2]$ has one extra bit of redundancy compared to the parity code, but its distance $d = 2$ indicates that its full fault coverage is the same as that of the parity. The applied parity matrix of these codes are

$$P_1 = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}, \quad P_2 = \begin{bmatrix} 0\,1 \\ 1\,1 \\ 1\,0 \\ 1\,1 \end{bmatrix}, \quad P_3 = \begin{bmatrix} 0\,1\,1 \\ 1\,0\,1 \\ 1\,1\,1 \\ 1\,1\,0 \end{bmatrix}.$$

  In total, there are 6 of $[6, 4, 2]$ and 4 of $[7, 4, 3]$-codes up to bit permutations in the input and output of the generator matrix. It is noteworthy to mention, depending on the type of functions, choice of the generator matrix can lead to different costs of implementation.

- $[8, 4, 4]^{16}$. The common code for this case is the *Extended Hamming*-code $[8, 4, 4]$ that implies the case for injective F. The parity matrix of the Extended Hamming code is

$$P_{EH} = \begin{bmatrix} 1\,1\,1\,0 \\ 1\,1\,0\,1 \\ 1\,0\,1\,1 \\ 0\,1\,1\,1 \end{bmatrix}.$$

- $[d, 1, d]^{64}$, i. e., duplication $[2, 1, 2]^{64}$, triplication $[3, 1, 3]^{64}$, and quadruplication $[4, 1, 4]^{64}$. These cases are included in the investigation to compare the proposed methodology and typical and straightforward duplication schemes that provide full fault coverage considering the same adversary model.

AES-128

Similar to previously mentioned 64-bit block ciphers, the implementations are categorized into three groups for AES-128:

- Non-injective F: $[9,8,2]^{16}$-, $[12,8,3]^{16}$-, $[13,8,4]^{16}$-, and $[16,8,5]^{16}$-codes that implies the case for non-injective F. Note that they are one of the smallest codes with rank 8 and the considered distances 2, 3, 4, and 5, resp. The applied parity matrix of these codes are

$$P_1 = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}, P_2 = \begin{bmatrix} 1\,0\,1\,1 \\ 0\,1\,1\,1 \\ 1\,1\,0\,0 \\ 1\,0\,1\,0 \\ 1\,0\,0\,1 \\ 0\,1\,1\,0 \\ 0\,1\,0\,1 \\ 0\,0\,1\,1 \end{bmatrix}, P_3 = \begin{bmatrix} 1\,1\,1\,0\,0 \\ 1\,1\,0\,1\,0 \\ 1\,0\,1\,0\,1 \\ 1\,0\,0\,1\,1 \\ 0\,1\,1\,1\,0 \\ 0\,1\,1\,0\,1 \\ 0\,1\,0\,1\,1 \\ 0\,0\,1\,1\,1 \end{bmatrix}, P_4 = \begin{bmatrix} 1\,1\,1\,0\,0\,0\,0\,1 \\ 1\,1\,0\,1\,1\,0\,0\,0 \\ 1\,0\,1\,0\,1\,1\,0\,0 \\ 1\,0\,0\,1\,0\,1\,1\,0 \\ 0\,1\,1\,0\,1\,0\,1\,0 \\ 0\,1\,0\,1\,0\,1\,0\,1 \\ 0\,0\,1\,1\,0\,0\,1\,1 \\ 0\,0\,0\,0\,1\,1\,1\,1 \end{bmatrix}.$$

- Injective F: $[17,8,6]^{16}$-, $[19,8,7]^{16}$-, and $[20,8,8]^{16}$-code which reflect 9-, 11-, and 12-bit redundancy with a distance of 6, 7, and 8, resp. The P matrix of these codes are as follows.

$$P_5 = \begin{bmatrix} 1\,1\,1\,0\,1\,0\,0\,1\,0 \\ 1\,1\,0\,1\,1\,1\,0\,0\,0 \\ 1\,0\,1\,0\,1\,1\,1\,0\,0 \\ 1\,0\,0\,1\,0\,1\,1\,1\,0 \\ 0\,1\,1\,0\,1\,0\,1\,0\,1 \\ 0\,1\,0\,1\,0\,1\,0\,1\,1 \\ 0\,0\,1\,1\,0\,0\,1\,1\,1 \\ 0\,0\,0\,0\,1\,1\,1\,1\,1 \end{bmatrix}, \quad P_6 = \begin{bmatrix} 1\,0\,1\,1\,0\,0\,1\,1\,0\,1\,0 \\ 1\,0\,1\,0\,1\,1\,1\,0\,0\,0\,1 \\ 1\,0\,0\,1\,1\,1\,0\,1\,1\,0\,0 \\ 0\,1\,1\,1\,0\,1\,1\,0\,1\,0\,0 \\ 0\,1\,1\,0\,1\,1\,0\,1\,0\,1\,0 \\ 0\,1\,0\,1\,1\,0\,1\,1\,0\,0\,1 \\ 0\,0\,1\,1\,1\,0\,0\,0\,1\,1\,1 \\ 0\,0\,0\,0\,0\,1\,1\,1\,1\,1\,1 \end{bmatrix},$$

$$P_7 = \begin{bmatrix} 1\,1\,1\,1\,1\,0\,1\,0\,0\,1\,0\,0 \\ 1\,1\,1\,0\,0\,1\,1\,1\,0\,0\,1\,0 \\ 1\,0\,1\,1\,0\,1\,0\,1\,1\,1\,0\,0 \\ 1\,0\,0\,1\,1\,1\,1\,0\,1\,0\,1\,0 \\ 0\,1\,1\,0\,1\,1\,0\,1\,0\,1\,0\,1 \\ 0\,1\,0\,1\,1\,0\,1\,1\,0\,0\,1\,1 \\ 0\,0\,1\,1\,1\,0\,0\,0\,1\,1\,1\,1 \\ 0\,0\,0\,0\,0\,1\,1\,1\,1\,1\,1\,1 \end{bmatrix}.$$

- $[d,1,d]^{128}$ i. e., duplication $[2,1,2]^{128}$, triplication $[3,1,3]^{128}$, and quadruplication $[4,1,4]^{128}$.

3.3 CRAFT: LIGHTWEIGHT TWEAKABLE BLOCK CIPHER

In this section, a new block cipher is presented, which considers protection against DFA attacks by design. To this end, we show that any cipher that allows the fault-detection unit to solely operate on the redundant part of information with a strictly shorter size than that of the plaintext/ciphertext has a critical cryptographic weakness. In the second part, we present the tweakable block cipher CRAFT (with effiCient pRotection Against differential Fault analysis aTtacks) with a 64-bit block, 128-bit key, and 64-bit tweak length. In short, its properties are listed below.

- In addition to having strong cryptographic properties, CRAFT has been particularly designed to ease the integration of code-based fault-detection schemes following the concept presented in Sections 3.1 and 3.2. This allows the application of any arbitrary EDC in the implementation; four of such codes are considered in the case studies.

  At the same time as designing CRAFT, the permutation FRIT by Simon, Batina, Daemen, Grosso, Massolino, Papagiannopoulos, Regazzoni, and Samwel [Sim+18] was introduced in which efficient implementation of a fault-detection technique has been considered as a design criterion. Although it got broken by Dobraunig, Eichlseder, Mendel, and Schofnegger [Dob+19], FRIT uses *interleaved parity* for fault detection, which – based on the definition of the underlying code – can guarantee the detection of only single-bit faults.

- Due to its fundamental building blocks' involutory property, the CRAFT encryption function can easily be turned into decryption, supporting both encryption and decryption with minimal area cost.

- CRAFT supports a 64-bit tweak, which adds a very little area to the corresponding implementation.

- Considering the area footprint of a round-based architecture, CRAFT outperforms – to the best of our knowledge – all lightweight block ciphers with the same state and key size. As a remarkable outcome, its encryption-only core needs 949 GE, which is way lower than any reported round-based implementation of a lightweight cipher.[13] It indeed competes with a bit-serial implementation of SIMON with 958 GE requiring 64×44 clock cycles [Bea+15]. Among the key features that enable these achievements is the construction of the key schedule in CRAFT, where – similar to MIDORI [Ban+15], PICCOLO [Shi+11], and KTANTAN [DDK09] – the key bits are alternated. This allows avoiding instantiating extra registers to process and generate the round keys in round-based architectures.

- Focusing on fault-protected implementations, under all settings with respect to the employed EDC, its area overhead (even with decryption and tweak support) is smaller than all block ciphers considered in Section 3.2 with compatible state and key size.

[Sim+18] Simon, Batina, Daemen, Grosso, Massolino, Papagiannopoulos, Regazzoni, and Samwel *Towards Lightweight Cryptographic Primitives with Built-in Fault-Detection*

[Dob+19] Dobraunig, Eichlseder, Mendel, and Schofnegger "Algebraic Cryptanalysis of Variants of Frit"

[13] Note that KTANTAN, due to its 80-bit key size and the high number of clock cycles for encryption, is excluded.

CRAFT is built upon the knowledge and experience that has been established for building lightweight (tweakable) ciphers by now. This is reflected in the fact that CRAFT, on a high-level view, is quite similar to, e. g., SKINNY [Bei+16], which itself borrows the general structure of the AES. As mentioned above, this, in particular, allows us to base the security analyses on well-known principles.

3.3.1 *Lower Bounds on the Redundancy*

Consider the construction shown in Figure 3.5. F does not need to be linear; indeed, fulfilling the goal of separation between A and A′ can be done by replacing F with any injective function. The selection of a linear function

helps to stay with characteristics of systematic binary linear codes, which, compared to non-linear functions, have better fault detection properties.[14] It has another advantage that the algebraic degree of the sub-functions T' stays the same as that of T. This is beneficial when the implementation should also be protected against SCA attacks.

[14] For instance, parity code as a linear code can detect all of the single-bit faults while there is no non-linear code with parity size one that can detect all such faults.

It is explained in Section 3.2.1 that if $p < k$, the construction shown in Figure 3.5 is not necessarily feasible. It depends on the employed function F and the underlying computing function T. As a temporary goal, let's aim at examining whether it is possible to design a block cipher, with its round function denoted by T, in such a way that its fault-protected implementation (using F) can be realized following the construction in Figure 3.5. That means it should be $p < k$, and A' should solely operate on redundant information (i. e., signature marked as INPUT′ in Figure 3.5).

The following theorem shows that in the construction of Figure 3.5, if $p < k$, for any function $F : \mathbb{F}_2^k \to \mathbb{F}_2^p$, there is a crucial structural weakness of the underlying block cipher.

Theorem 65. *If a cipher can have an error detection structure using area redundancy, where the redundancy part can be processed independently (the structure in Figure 3.5 but with any arbitrary applied F function) with a redundancy size smaller than the block size, then the cipher is not cryptographically secure.*

Proof. Let us assume $\mathtt{Enc}_K$, the encryption with key K, formed by repeating the round function T for a certain number of rounds. Also (just for now), suppose that the bit length of the original data (plaintext/ciphertext) is denoted by k, and the bit length of the redundant information is p.

By construction, there is a function F such that for each instance of a cipher $\mathtt{Enc}_K$, there exists an encryption $\mathtt{Enc}'_K$ such that $\mathtt{Enc}'_K \circ F = F \circ \mathtt{Enc}_K$. Thus, for every two plaintexts P_1 and P_2 for which $F(P_1) = F(P_2)$, one obtains $F \circ \mathtt{Enc}_K(P_1) = F \circ \mathtt{Enc}_K(P_2)$. Thus, regardless of the key, whenever two plaintexts have the same image under F, the corresponding ciphertexts will also have the same image under F. For instance, if F is a balanced function mapping k to p bits, each of the 2^p different preimages $F^{-1}(x), x \in \mathbb{F}_2^p$ is a set containing 2^{k-p} elements. Further, every instance $\mathtt{Enc}_K$ operates as a permutation over these preimages, i. e.,

$$\text{if } \mathtt{Enc}_K : X \mapsto Y, \text{ then } \mathtt{Enc}_K : F^{-1}(X) \mapsto F^{-1}(Y).$$

Thus, under a chosen-plaintext attack, the underlying cipher $\mathtt{Enc}_K$ can easily be distinguished from a random permutation. The adversary chooses plaintexts P_1 and P_2 with $F(P_1) = F(P_2)$. If $C_i = \mathtt{Enc}_K(P_i)$ with $i \in \{1, 2\}$, then the property $F(C_1) = F(C_2)$ holds with a probability of 1. Note that this property should hold only with probability 2^{-p} for a uniformly chosen random permutation. □

As a conclusion, whenever $p < k$, *it is not possible to design a fully secure cipher* if the construction in Figure 3.5 is desired. Therefore, in the rest of the current section, only the construction in Figure 3.5 for $p \geq k$[15] and the one in Figure 3.6 for $p < k$ are considered.

[15] Note that F must be an injective function.

3.3.2 *Specification*

CRAFT is a lightweight tweakable block cipher made out of involutory building blocks. It consists of a 64-bit block, a 128-bit key, and a 64-bit tweak. The state is viewed as a 4×4 square array of nibbles. The notation $I[i, j]$ denotes the nibble located at row i and column j of the state. One can also view this 4×4 square array as a vector by concatenating the rows. Thus, $I[i]$ denotes the nibble in the i-th position of this vector, i. e., $I[i, j] = I[4i + j]$.

The 128-bit key K is split into two 64-bit keys K_0 and K_1. Together with the 64-bit tweak input T, four 64-bit tweakeys TK_0, TK_1, TK_2 and TK_3 are derived. Each of those 64-bit tweakeys is also considered as a 4×4 square array of nibbles, and similar indexing as the one for the cipher state is used. By initializing the state with the plaintext, the cipher iterates 31 round functions ($\mathtt{R}_i$, $i \in \mathbb{Z}_{31}$) and appends one more linear round $\mathtt{R}'_{31}$ to compute the ciphertext. Figure 3.14 depicts the structure of CRAFT. Each round function $\mathtt{R}_i$ applies the following five involutory round operations: SubBox SB, MixColumn MC, PermuteNibbles PN, AddConstant $\mathtt{A}_{RC_i}$ and AddTweakey $\mathtt{A}_{TK_i}$, while $\mathtt{R}'_{31}$ only applies the MixColumn, AddConstant and AddTweakey operations. The round operations are defined as follows, and one full-round function is depicted in Figure 3.15.

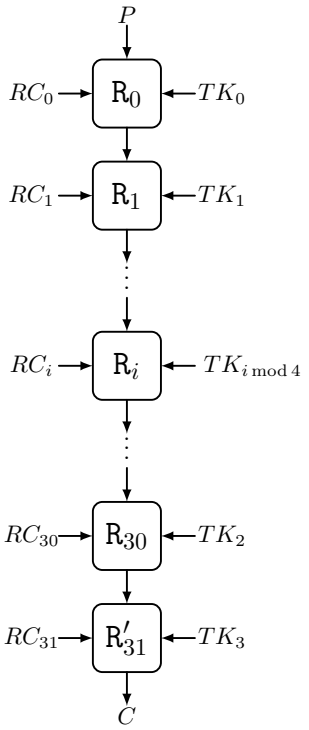


FIGURE 3.14: Structure of CRAFT.

SUBBOX SB: The 4-bit involutory S-box S is applied 16 times in parallel, i. e., to each nibble of the state. This S-box is the same as the S-box used in the block cipher MIDORI [Ban+15]. The table for the S-box (in hexadecimal notation) is given in Table 3.1.

TABLE 3.1: The S-box of MIDORI and CRAFT.

x	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
$S(x)$	c	a	d	3	e	b	f	7	8	9	1	5	0	2	4	6

MIXCOLUMN MC: The following involutory binary matrix M is multiplied to each column of nibbles in the state:[16]

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

That is, for each column index $j \in \mathbb{Z}_4$,

[16] Note that, by Definition 7, the linear layer of CRAFT is a binary multiplication in $\mathbb{F}_{2^4}$.

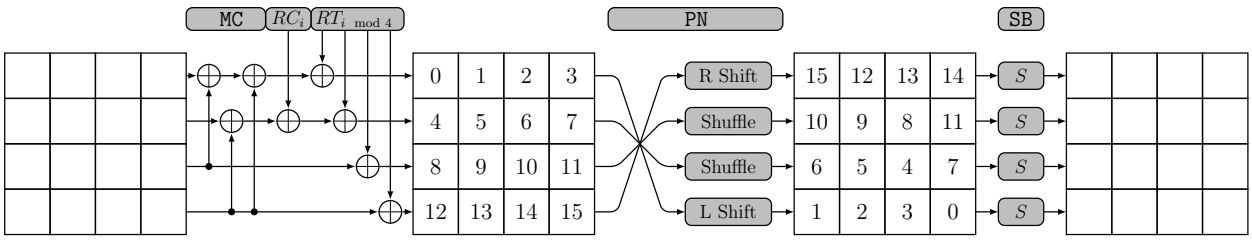


FIGURE 3.15: One Full Round of CRAFT.

$$\begin{bmatrix} I[0,j] \\ I[1,j] \\ I[2,j] \\ I[3,j] \end{bmatrix} \mapsto \begin{bmatrix} I[0,j] \oplus I[2,j] \oplus I[3,j] \\ I[1,j] \oplus I[3,j] \\ I[2,j] \\ I[3,j] \end{bmatrix} .$$

PERMUTENIBBLES PN: An involutory permutation P is applied on the nibble positions of the state. In particular, for all $i \in \mathbb{Z}_{16}$, $I[i]$ is replaced by $I[P(i)]$, where

$$P = [15, 12, 13, 14,\ 10, 9, 8, 11,\ 6, 5, 4, 7,\ 1, 2, 3, 0] .$$

ADDCONSTANTS $\mathtt{A}_{RC_i}$: One 4-bit and one 3-bit LFSR, whose states are denoted by $a = (a_0, a_1, a_2, a_3)$ and $b = (b_0, b_1, b_2)$ (with a_0 and b_0 being the most significant bits), resp., are used to generate round constants. The LFSRs are initialized by the values $(\mathtt{0}, \mathtt{0}, \mathtt{0}, \mathtt{1})$ and $(\mathtt{0}, \mathtt{0}, \mathtt{1})$ and their update functions are

$$(a_0, a_1, a_2, a_3) \to (a_3 \oplus a_2, a_0, a_1, a_2) \quad , \quad (b_0, b_1, b_2) \to (b_2 \oplus b_1, b_0, b_1) .$$

In every round, (a_0, a_1, a_2, a_3) and $(\mathtt{0}, b_0, b_1, b_2)$ are XOR-ed with the state nibbles $I[4]$ and $I[5]$, resp., and then both LFSRs get updated. Table 3.2 shows the hexadecimal values of all round constants, i. e., the integer value for $(a_0, a_1, a_2, a_3, \mathtt{0}, b_0, b_1, b_2)$.

TABLE 3.2: Round Constants of CRAFT.

Round i	$(a_0, a_1, a_2, a_3, \mathtt{0}, b_0, b_1, b_2)$
0 - 3	11, 84, 42, 25
4 - 7	96, c7, 63, b1
8 - 11	54, a2, d5, e6
12 - 15	f7, 73, 31, 14
16 - 19	82, 45, 26, 97
20 - 23	c3, 61, b4, 52
24 - 27	a5, d6, e7, f3
28 - 31	71, 34, 12, 85

ADDTWEAKEY $\mathtt{A}_{TK_i}$: Using a permutation Q on the nibbles of the given tweak, the cipher derives four 64-bit tweakeys TK_0, TK_1, TK_2 and TK_3 from the tweak T and the key $(K_0||K_1)$ as

$$TK_0 = K_0 \oplus T \ , \ TK_1 = K_1 \oplus T \ , \ TK_2 = K_0 \oplus \mathtt{QN}(T) \ , \ TK_3 = K_1 \oplus \mathtt{QN}(T) .$$

Thereby, $\mathtt{QN}(T)$ applies the permutation

$$Q = [12, 10, 15, 5,\ 14, 8, 9, 2,\ 11, 3, 7, 4,\ 6, 0, 1, 13]$$

on the position of tweak nibbles T where for all $i \in \mathbb{Z}_{16}$, $T[i]$ is replaced by $T[Q(i)]$. Then in each round i, without any key update, the tweakey $TK_{i \bmod 4}$ is XOR-ed to the cipher state.

ROUND FUNCTION: To conclude, using the above explained operations, the round functions $\mathtt{R}_i$, $i \in \mathbb{Z}_{31}$, are defined as

$$\mathtt{R}_i = \mathtt{SB} \circ \mathtt{PN} \circ \mathtt{A}_{TK_i} \circ \mathtt{A}_{RC_i} \circ \mathtt{MC}$$

and the last round $\mathtt{R}'_{31}$ as

$$\mathtt{R}_{31} = \mathtt{A}_{TK_{31}} \circ \mathtt{A}_{RC_{31}} \circ \mathtt{MC}.$$

Security Claim 1 (CRAFT). *Overall, we claim 124-bit security of* CRAFT *in the related-tweak model. Note that the claim does not consider the security in the chosen-key, known-key, or related-key model.*

From the provided analyses in Section 6.1, the most promising cryptanalysis on CRAFT is an accelerated exhaustive search with a time complexity of 2^{124} cipher encryptions and data and memory complexity of 16 (see Section 6.1.1).

It is expected that a 30-round version of CRAFT attains 128-bit security against (impossible) differential and (zero-correlation) linear attacks, integral attacks, and invariant attacks, as well as MitM attack. It is noteworthy that it does not mean that there exists a 30-round attack on CRAFT. It is only an upper bound on the number of rounds that can be attacked. Hence, considering two extra rounds as a security margin, it is claimed that 32-round CRAFT has 124-bit security in the related-tweak model.

3.3.3 *Rationale*

When designing CRAFT, the main criterion was to use components best suited for the fault-detection constructions following the structure introduced in Section 3.2 while provide the necessary cryptographic security. The second design criterion was to build a construction for which, with the least possible changes, both encryption and decryption use a similar structure. All details of the design choices are explained in the following.

Involutory Building Blocks

To design a cipher with a similar structure for both encryption and decryption, let's restrict the choices for the components of *substitution* and *permutation* to involutory ones. From the fact that all of the round operations are involutions and by applying the last linear round $\mathtt{R}'_{31}$, the CRAFT decryption is a parametrized CRAFT encryption.

Lemma 66. *Decryption with* CRAFT *with tweakeys* (TK_0, TK_1, TK_2, TK_3) *and round constants* $(RC_0, \ldots, RC_{31})$ *is the same as the* CRAFT *encryption with tweakeys* $(TK'_3, TK'_2, TK'_1, TK'_0)$ *and round constants* $(RC_{31}, \ldots, RC_0)$*, where* $TK'_i = \mathtt{MC}(TK_i)$.

Proof. Let $\mathtt{Enc}^{RC_0,\ldots,RC_{31}}_{TK_0,\ldots,TK_3}$ and $\mathtt{Dec}^{RC_0,\ldots,RC_{31}}_{TK_0,\ldots,TK_3}$ denote the CRAFT encryption and decryption with tweakeys (TK_0, TK_1, TK_2, TK_3) and round constants $(RC_0, RC_1, \cdots, RC_{31})$, resp. We will make use of the facts that $\mathtt{PN} \circ \mathtt{SB} = \mathtt{SB} \circ \mathtt{PN}$ and $\mathtt{MC} \circ \mathtt{A}_{RC} \circ \mathtt{A}_{TK} = \mathtt{A}_{TK'} \circ \mathtt{A}_{RC} \circ \mathtt{MC}$. We further make use of the observation that $\mathtt{MC} \circ \mathtt{A}_{RC} = \mathtt{A}_{RC}$ since the round constants are applied on the second nibble of each column, while for any $x, y \in \mathbb{F}_2^4$

$$M \cdot \begin{bmatrix} x \\ y \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} x \\ y \\ 0 \\ 0 \end{bmatrix}.$$

$$\begin{aligned}
\mathtt{Dec}^{RC_0,\ldots,RC_{31}}_{TK_0,\ldots,TK_3} &= (\mathtt{A}_{TK_3} \circ \mathtt{A}_{RC_{31}} \circ \mathtt{MC} \circ \mathtt{SB} \circ \mathtt{PN} \circ \mathtt{A}_{TK_2} \circ \mathtt{A}_{RC_{30}} \circ \mathtt{MC} \circ \cdots \circ \\
&\quad \mathtt{SB} \circ \mathtt{PN} \circ \mathtt{A}_{TK_1} \circ \mathtt{A}_{RC_1} \circ \mathtt{MC} \circ \mathtt{SB} \circ \mathtt{PN} \circ \mathtt{A}_{TK_0} \circ \mathtt{A}_{RC_0} \circ \mathtt{MC})^{-1} \\
&= \mathtt{MC} \circ \mathtt{A}_{RC_0} \circ \mathtt{A}_{TK_0} \circ \mathtt{PN} \circ \mathtt{SB} \circ \mathtt{MC} \circ \mathtt{A}_{RC_1} \circ \mathtt{A}_{TK_1} \circ \mathtt{PN} \circ \mathtt{SB} \\
&\quad \circ \cdots \circ \mathtt{MC} \circ \mathtt{A}_{RC_{30}} \circ \mathtt{A}_{TK_2} \circ \mathtt{PN} \circ \mathtt{SB} \circ \mathtt{MC} \circ \mathtt{A}_{RC_{31}} \circ \mathtt{A}_{TK_3} \\
&= \mathtt{A}_{TK'_0} \circ \mathtt{A}_{RC_0} \circ \mathtt{MC} \circ \mathtt{SB} \circ \mathtt{PN} \circ \mathtt{A}_{TK'_1} \circ \mathtt{A}_{RC_1} \circ \mathtt{MC} \circ \mathtt{SB} \circ \mathtt{PN} \\
&\quad \circ \cdots \circ \mathtt{A}_{TK'_2} \circ \mathtt{A}_{RC_{30}} \circ \mathtt{MC} \circ \mathtt{SB} \circ \mathtt{PN} \circ \mathtt{A}_{TK'_3} \circ \mathtt{A}_{RC_{31}} \circ \mathtt{MC} \\
&= \mathtt{Enc}^{RC_{31},\ldots,RC_0}_{TK'_3,\ldots,TK'_0}.
\end{aligned}$$

□

Algorithm 1 CRAFT Encryption.

function CRAFT ENCRYPTION(X, T, K_0, K_1)
  $TK_0 \leftarrow K_0 \oplus T$
  $TK_1 \leftarrow K_1 \oplus T$
  $TK_0 \leftarrow K_2 \oplus \mathtt{QN}(T)$
  $TK_1 \leftarrow K_3 \oplus \mathtt{QN}(T)$
  $(a_0, a_1, a_2, a_3, b_0, b_1, b_2) \leftarrow (\mathtt{1}, \mathtt{0}, \mathtt{0}, \mathtt{0}, \mathtt{1}, \mathtt{0}, \mathtt{0})$
  for $i \leftarrow 0 : 31$ **do**
    $X \leftarrow \mathtt{MC}(X)$
    $X[4] \leftarrow X[4] \oplus (a_0, a_1, a_2, a_3)$
    $X[5] \leftarrow X[5] \oplus (b_0, b_1, b_2, \mathtt{0})$
    $X \leftarrow X \oplus TK_{i \bmod 4}$
    if $i = 31$ **then return** X
    $X \leftarrow \mathtt{PN}(X)$
    $X \leftarrow \mathtt{SB}(X)$
    $(a_0, a_1, a_2, a_3) \leftarrow (a_1, a_2, a_3, a_1 \oplus a_0)$
    $(b_0, b_1, b_2) \leftarrow (b_1, b_2, b_1 \oplus b_0)$

Algorithm 2 CRAFT Decryption.

function CRAFT ENCRYPTION(X, T, K_0, K_1)
  $TK_0 \leftarrow \mathtt{MC}\big(K_0 \oplus T\big)$
  $TK_1 \leftarrow \mathtt{MC}\big(K_1 \oplus T\big)$
  $TK_0 \leftarrow \mathtt{MC}\big(K_2 \oplus \mathtt{QN}(T)\big)$
  $TK_1 \leftarrow \mathtt{MC}\big(K_3 \oplus \mathtt{QN}(T)\big)$
  $(a_0, a_1, a_2, a_3, b_0, b_1, b_2) \leftarrow (\mathtt{0}, \mathtt{0}, \mathtt{0}, \mathtt{1}, \mathtt{1}, \mathtt{0}, \mathtt{1})$
  for $i \leftarrow 31 : 0$ **do**
    $X \leftarrow \mathtt{MC}(X)$
    $X[4] \leftarrow X[4] \oplus (a_0, a_1, a_2, a_3)$
    $X[5] \leftarrow X[5] \oplus (b_0, b_1, b_2, \mathtt{0})$
    $X \leftarrow X \oplus TK_{i \bmod 4}$
    if $i = 0$ **then return** X
    $X \leftarrow \mathtt{PN}(X)$
    $X \leftarrow \mathtt{SB}(X)$
    $(a_0, a_1, a_2, a_3) \leftarrow (a_3 \oplus a_0, a_0, a_1, a_2)$
    $(b_0, b_1, b_2) \leftarrow (b_2 \oplus b_0, b_0, b_1)$

Algorithms 1 and 2 show pseudo-codes for the encryption and decryption functions resp. Lemma 66 shows that the two algorithms can be efficiently merged. This feature of CRAFT is discussed in detail with respect to hardware implementations in Section 3.3.4.

S-box

To choose the S-box which best suits our structure while at the same time have good cryptographic properties, the following approach is done. For all 46 206 736 involutory 4-bit S-boxes, first, their uniformity u and linearity l are considered. Then, all S-boxes with trivial differential or linear characteristics are discarded. Second, since a bit permutation at the input or the

output of an S-box does not change its implementation area, those S-boxes from the candidate list, which are bit permutation equivalent of each other, are omitted. This means if two S-boxes are different only with respect to a bit permutation at the input/output, one of them is kept. Then, the implementation area cost concerning the independence property introduced in Section 3.1.4 for the remaining S-box candidates are evaluated. Therefore, the circuit implementing an S-box needs to be split up into independent component circuits, each computing exactly *one* output bit. In the implementation of the fault-detection structure – as explained in Section 3.2.2 – in addition to the S-box S, depending on the size of the redundancy, either $F \circ S \circ F^{-1}$ or $F \circ S$ needs to be implemented (see Figures 3.8 and 3.9), with $F : x \mapsto x \cdot P_q$ and P_q being the parity matrix of the underlying binary linear systematic code to compute the q-bit parity. Considering all four cases for the redundancy size $q \in \{1, 2, 3, 4\}$, four redundant S-boxes are defined: $S_1 = F_1 \circ S$, $S_2 = F_2 \circ S$, $S_3 = F_3 \circ S$ and $S_4 = F_4 \circ S \circ F_4^{-1}$. The corresponding parity matrices can be given as follows:

$$P_1 = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}, \quad P_2 = \begin{bmatrix} 1 & 1 \\ 1 & 1 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad P_3 = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix}, \quad P_4 = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{bmatrix}.$$

It is noteworthy that any two rows of P_4 result is a valid choice for P_2, and the same holds for P_3 made of any three rows of P_4. Therefore, in the comparisons it is considered that $S'_4 = F_4 \circ S$, and then, the two cheapest output bits (with respect to the necessary area) are chosen as S_2 and the three cheapest bits as S_3 where by cheap, the smaller area size in the hardware implementation is meant.

Since it is necessary to fulfill the independence property, the size of the implementation for a vectorial Boolean function is equal to the sum of the area for implementing each of its Boolean coordinate functions. Hence, to evaluate the size of the implementation of S, S_1, ..., and S_4, it is necessary to know the size of the implementation of S, S_1, S'_4 and S_4, which include 13 Boolean coordinate functions. This means that it is not necessary to implement and synthesize all S-box candidates. Instead, we only need to evaluate the size of the implementation of all 12 870 four-bit Boolean coordinate functions. Actually, by omitting the Boolean coordinate functions, which are bit permutation equivalent of the others, we end up with only 730 Boolean coordinate functions which need to be implemented and synthesized to evaluate their implementation size. To this end, using Synopsys Design Compiler with the publicly available IBM 130 nm ASIC library, this task is accomplished in a fraction of one day. Since we have evaluated all 4-bit Boolean coordinate functions' area requirement, we can easily calculate the implementation size of any given S, S_1, S'_4 and S_4.

In order to classify the constructed S-boxes, the implementation size of five combinations are considered: S, (S, S_1), (S, S_2), (S, S_3), (S, S_4) corresponding to the S-box itself (without redundancy), and four other cases of the redundancy size $q \in \{1, 2, 3, 4\}$. The results are given in Table 3.3 that are sorted by the first being the best choice. The index of the affine equivalent class of 4-bit S-boxes introduced in [Bil+12], is referred to by

TABLE 3.3: Result of the S-box Search, Area Size (GE) using the IBM 130 nm ASIC Library.

Class	(u,l)	(n_D,n_L)	S-box	Size				
				S	$+S_1$	$+S_2$	$+S_3$	$+S_4$
266	(4,4)	(32,32)	9BDFAE678041C253	11	16	22.5	26.5	26.25
262	(8,4)	(64,32)	018945EA237DFB6C	12.25	19.5	20.75	26	29
45	(6,6)	(46,78)	016E4F2798ACBD35	12	16	20	25.25	30.5
51	(6,6)	(46,78)	B7A39F816420DCE5	12.5	18.5	19.75	25.25	34.75
76	(6,6)	(46,78)	E6AFD5178C2B9403	12.75	18.75	20.5	25	34.75
208	(6,6)	(46,78)	413205B7CEA68F9D	12.25	19	24.25	30.25	25.75
			D16EF52798ACB034	13	19.25	23.75	29.75	25.75
24	(8,6)	(64,78)	6C284E0F39BA1D57	13	16.5	21	25.75	32.5
32	(8,6)	(64,78)	3210654D89CBA7EF	14	20	21	26.25	42.25
46	(8,6)	(64,78)	B7A395816420DCEF	12.5	17.5	19	24.25	33
57	(8,6)	(64,78)	DF574263B9A8E0C1	11.25	15.25	22	28	28.25
101	(8,6)	(64,78)	21034D6CAF8E75B9	14	17.75	21.25	25.5	32.25
102	(8,6)	(64,78)	290D6B4C81A573FE	13	19.75	25	31.5	25
			DF32CE6A987B4051	12	19.25	20.25	26.5	25
137	(10,6)	(95,78)	01CD6F4E89AB2375	19.25	21.25	21.25	25.75	32
			98DCF65710BA32E4	19.5	20	21.25	25.75	33.25
141	(10,6)	(95,78)	98DCF76510BA32E4	12.75	19	21.5	26	32.25
149	(10,6)	(95,78)	465E021FCDAB8937	12	17.75	20.5	26	26.75
217	(10,6)	(95,78)	6523410BEDF7C98A	11	16.25	20.25	22.25	31.25
			6DF74903E5ABC182	14	21.5	20.25	20.5	35.75
216	(12,6)	(155,78)	2301ABF7CD4589E6	10.75	15.5	21	27.5	25.75
			A68437152B09FDEC	11	15.25	18.5	23.5	29.75
			E6C43715B9A82F0D	11.25	16.5	20.75	26.25	27.75
			6789AB012345FDEC	11.75	16.5	19.25	23.5	27
			84A617350B29FDEC	12.5	15.75	20	25.25	29.25
			210BFD8967A3E5C4	12.5	20.75	21	27.25	25.5
			EAC8654F3B192D07	13	18.75	23	29	25.5

the class number. Besides, (u,l) denotes the uniformity and linearity of the corresponding class, and (n_D,n_L) denotes the minimum number of active S-boxes in the differential and linear attack to reach a differential trail probability of $\leq 2^{-64}$ and a linear trail correlation of $\leq 2^{-32}$, resp.

To obtain the list in Table 3.3, all cases are searched, and no other choice is found for which all five size values are smaller than that of the candidate S-box class 266, the so-called reference S-box. Also, the list includes the other cases that have at least one size (among five) smaller than the reference S-box.

Notably, the reference S-box has minimum uniformity and linearity. Although other candidates lead to low area requirements, they have larger uniformity or linearity. Hence, their usage would imply that we should use more rounds to protect the cipher from differential or linear attacks. Therefore, it is decided to choose the reference S-box as the best suited choice for our structure for fault detection. The reference S-box is a bit

permuted version of the MIDORI S-box. It is a coincidence since the MIDORI S-box has been selected based on its predicted low energy consumption. Hence, to avoid introducing a new S-box, it is decided to use the MIDORI S-box in CRAFT.

Linear Layer

For making CRAFT efficient to be implemented in fault-detection structures with redundancy size smaller than the block size, it is decided to use a binary matrix for the MixColumn operation. According to Theorem 64, it allows the MixColumn of the redundant part of the circuit $\mathtt{A}'$ to solely operate on the redundant part of the information (see Figure 3.10). Among all 20 160 bijective 4×4 binary matrices, only 316 are involutions. On the other hand, since CRAFT includes PermuteNibbles applied right after MixColumn, and because all the involutory permutations are checked for each matrix, this reduces the set of 316 matrices to 30 candidates, which are equivalent up to a permutation of the rows.

Lemma 67. *Let $\mathtt{PN}_r$ be the operation that permutes the position of the rows of the state. The encryption with round operations $\mathtt{SB}$, $\mathtt{MC}$ and $\mathtt{PN}$ is the same as the encryption with round operations $\mathtt{SB}$, $\mathtt{MC}'$ and $\mathtt{PN}'$ with modified tweakey and round constants and up to a nibble-wise permutation of the plaintext and ciphertext, where $\mathtt{MC}' = \mathtt{PN}_r^{-1} \circ \mathtt{MC} \circ \mathtt{PN}_r$ and $\mathtt{PN}' = \mathtt{PN}_r^{-1} \circ \mathtt{PN} \circ \mathtt{PN}_r$.*

Proof. By simplifying the round function of the second cipher, we have

$$\begin{aligned}
\mathtt{R}' &= \mathtt{SB} \circ \mathtt{PN}' \circ \mathtt{A}_{TK} \circ \mathtt{A}_{RC} \circ \mathtt{MC}' \\
&= \mathtt{SB} \circ \mathtt{PN}_r^{-1} \circ \mathtt{PN} \circ \mathtt{PN}_r \circ \mathtt{A}_{TK} \circ \mathtt{A}_{RC} \circ \mathtt{PN}_r^{-1} \circ \mathtt{MC} \circ \mathtt{PN}_r \\
&= \mathtt{SB} \circ \mathtt{PN}_r^{-1} \circ \mathtt{PN} \circ \mathtt{A}'_{TK} \circ \mathtt{A}'_{RC} \circ \mathtt{PN}_r \circ \mathtt{PN}_r^{-1} \circ \mathtt{MC} \circ \mathtt{PN}_r \\
&= \mathtt{PN}_r^{-1} \circ \mathtt{SB} \circ \mathtt{PN} \circ \mathtt{A}'_{TK} \circ \mathtt{A}'_{RC} \circ \mathtt{MC} \circ \mathtt{PN}_r,
\end{aligned}$$

where $RC' = \mathtt{PN}_r(RC)$ and $TK'_i = \mathtt{PN}_r(TK_i)$. By iterating the round function of the second cipher we have:

$$\begin{aligned}
\mathtt{Enc}'^{RC_0,\ldots,RC_{31}}_{TK_0,\ldots,TK_3} &= \mathtt{PN}_r^{-1} \circ \mathtt{A}_{TK'_3} \circ \mathtt{A}_{RC'_{31}} \circ \mathtt{MC} \circ \cdots \circ \mathtt{SB} \circ \mathtt{PN} \circ \mathtt{A}_{TK'_0} \circ \mathtt{A}_{RC'_0} \circ \mathtt{MC} \circ \mathtt{PN}_r \\
&= \mathtt{PN}_r^{-1} \circ \mathtt{Enc}^{RC'_0,\ldots,RC'_{31}}_{TK'_0,\ldots,TK'_3} \circ \mathtt{PN}_r.
\end{aligned}$$

□

The 30 candidates for the matrix M are given below.

$$M_0 : \begin{bmatrix} 1&0&0&0 \\ 0&1&0&0 \\ 0&0&1&0 \\ 0&0&0&1 \end{bmatrix} \quad M_1 : \begin{bmatrix} 0&1&0&0 \\ 1&0&0&0 \\ 0&0&1&0 \\ 0&0&0&1 \end{bmatrix} \quad M_2 : \begin{bmatrix} 0&1&0&0 \\ 1&0&0&0 \\ 0&0&0&1 \\ 0&0&1&0 \end{bmatrix} \quad M_3 : \begin{bmatrix} 1&0&0&1 \\ 0&1&0&0 \\ 0&0&1&0 \\ 0&0&0&1 \end{bmatrix}$$

$$M_4 : \begin{bmatrix} 0&1&0&0 \\ 1&0&0&0 \\ 0&0&1&1 \\ 0&0&0&1 \end{bmatrix} \quad M_5 : \begin{bmatrix} 1&0&1&1 \\ 0&1&0&0 \\ 0&0&1&0 \\ 0&0&0&1 \end{bmatrix} \quad M_6 : \begin{bmatrix} 1&0&0&1 \\ 0&1&0&1 \\ 0&0&1&0 \\ 0&0&0&1 \end{bmatrix} \quad M_7 : \begin{bmatrix} 1&0&1&0 \\ 0&1&0&1 \\ 0&0&1&0 \\ 0&0&0&1 \end{bmatrix}$$

$$M_8 : \begin{bmatrix} 0&1&0&1 \\ 1&0&0&1 \\ 0&0&1&0 \\ 0&0&0&1 \end{bmatrix} \quad M_9 : \begin{bmatrix} 1&1&1&0 \\ 0&0&1&0 \\ 0&1&0&0 \\ 0&0&0&1 \end{bmatrix} \quad M_{10} : \begin{bmatrix} 1&1&1&1 \\ 0&1&0&0 \\ 0&0&1&0 \\ 0&0&0&1 \end{bmatrix} \quad M_{11} : \begin{bmatrix} 1&0&0&1 \\ 0&1&0&1 \\ 0&0&1&1 \\ 0&0&0&1 \end{bmatrix}$$

$$M_{12}: \begin{bmatrix} 0&1&1&0\\ 1&0&0&1\\ 0&0&0&1\\ 0&0&1&0 \end{bmatrix} \quad M_{13}: \begin{bmatrix} 1&0&1&1\\ 0&1&0&1\\ 0&0&1&0\\ 0&0&0&1 \end{bmatrix} \quad M_{14}: \begin{bmatrix} 0&1&0&1\\ 1&0&0&1\\ 0&0&1&1\\ 0&0&0&1 \end{bmatrix} \quad M_{15}: \begin{bmatrix} 1&1&1&1\\ 0&0&1&0\\ 0&1&0&0\\ 0&0&0&1 \end{bmatrix}$$

$$M_{16}: \begin{bmatrix} 1&0&1&1\\ 0&1&1&1\\ 0&0&1&0\\ 0&0&0&1 \end{bmatrix} \quad M_{17}: \begin{bmatrix} 0&1&1&1\\ 1&0&1&1\\ 0&0&1&0\\ 0&0&0&1 \end{bmatrix} \quad M_{18}: \begin{bmatrix} 1&1&1&0\\ 0&1&0&1\\ 0&0&1&1\\ 0&0&0&1 \end{bmatrix} \quad M_{19}: \begin{bmatrix} 0&1&1&1\\ 1&0&1&0\\ 0&0&1&1\\ 0&0&0&1 \end{bmatrix}$$

$$M_{20}: \begin{bmatrix} 1&1&1&0\\ 0&0&1&1\\ 0&1&0&1\\ 0&0&0&1 \end{bmatrix} \quad M_{21}: \begin{bmatrix} 1&0&1&1\\ 0&1&1&1\\ 0&0&0&1\\ 0&0&1&0 \end{bmatrix} \quad M_{22}: \begin{bmatrix} 1&1&0&1\\ 0&1&1&1\\ 0&0&0&1\\ 0&0&1&0 \end{bmatrix} \quad M_{23}: \begin{bmatrix} 0&1&1&1\\ 1&0&1&1\\ 0&0&0&1\\ 0&0&1&0 \end{bmatrix}$$

$$M_{24}: \begin{bmatrix} 1&1&1&1\\ 0&1&0&1\\ 0&0&1&1\\ 0&0&0&1 \end{bmatrix} \quad M_{25}: \begin{bmatrix} 1&1&1&1\\ 0&0&1&1\\ 0&1&0&1\\ 0&0&0&1 \end{bmatrix} \quad M_{26}: \begin{bmatrix} 1&1&1&1\\ 1&0&1&0\\ 1&0&1&1\\ 0&1&1&0 \end{bmatrix} \quad M_{27}: \begin{bmatrix} 1&1&1&0\\ 1&1&0&1\\ 1&0&1&1\\ 0&1&1&1 \end{bmatrix}$$

$$M_{28}: \begin{bmatrix} 1&1&1&0\\ 1&0&1&1\\ 1&1&0&1\\ 0&1&1&1 \end{bmatrix} \quad M_{29}: \begin{bmatrix} 0&1&1&1\\ 1&0&1&1\\ 1&1&0&1\\ 1&1&1&0 \end{bmatrix}$$

For each of these 30 candidate matrices, in principle, it is necessary to search through all 46 206 736 involutory permutations for $\mathtt{PN}$. The lemma given below explains that it is not necessary to consider *all* such permutations, but only those up to a permutation over the state's columns.

Lemma 68. *Let $\mathtt{PN}_c$ be a permutation over the columns of the state. The encryption with round operations $\mathtt{SB}$, $\mathtt{MC}$ and $\mathtt{PN}$ is the same as the encryption with round operations $\mathtt{SB}$, $\mathtt{MC}$ and $\mathtt{PN}'$ with modified tweakey and round constants and up to a nibble-wise permutation of the plaintext and ciphertext, where $\mathtt{PN}' = \mathtt{PN}_c^{-1} \circ \mathtt{PN} \circ \mathtt{PN}_c$.*

Proof. The proof is quite similar to the one given for Lemma 68. The round function of the second cipher will be simplified to:

$$\begin{aligned} \mathtt{R}' &= \mathtt{SB} \circ \mathtt{PN}' \circ \mathtt{A}_{TK} \circ \mathtt{A}_{RC} \circ \mathtt{MC} \\ &= \mathtt{SB} \circ \mathtt{PN}_c^{-1} \circ \mathtt{PN} \circ \mathtt{PN}_c \circ \mathtt{A}_{TK} \circ \mathtt{A}_{RC} \circ \mathtt{MC} \\ &= \mathtt{PN}_c^{-1} \circ \mathtt{SB} \circ \mathtt{PN} \circ \mathtt{A}_{TK'} \circ \mathtt{A}_{RC'} \circ \mathtt{MC} \circ \mathtt{PN}_c, \end{aligned}$$

with $RC' = \mathtt{PN}_c(RC)$ and $TK'_i = \mathtt{PN}_c(TK_i)$. Hence for the second cipher we have:

$$\begin{aligned} \mathtt{Enc}'^{\,RC_0,\ldots,RC_{31}}_{TK_0,\ldots,TK_3} &= \mathtt{PN}_c^{-1} \circ \mathtt{A}_{TK'_3} \circ \mathtt{A}_{RC'_{31}} \circ \mathtt{MC} \circ \cdots \circ \mathtt{SB} \circ \mathtt{PN} \circ \mathtt{A}_{TK_0} \circ \mathtt{A}_{RC_0} \circ \mathtt{MC} \circ \mathtt{PN}_c \\ &= \mathtt{PN}_c^{-1} \circ \mathtt{Enc}^{RC'_0,\ldots,RC'_{31}}_{TK'_0,\ldots,TK'_3} \circ \mathtt{PN}_c. \end{aligned}$$

□

Therefore, we can reduce the search space of all involutory permutations P by the above equivalency. In combination with 30 candidate matrices M_0 to M_{29}, we need to search through such permutations to find those that provide the highest possible security level. To this end, we evaluated the

minimum number of rounds such that the linear layer attains full diffusion in both forward and backward directions (r_1) together with the minimum number of rounds to guarantee at least 32 active S-boxes in both differential and linear attacks (r_2).

Regardless of the choice for involutory P, the matrices $M_0, \ldots, M_{11}$ do not provide full diffusion. For the remaining 18 matrices, their minimum possible r_1 and r_2 are listed together with their hardware implementation cost in Table 3.4. By the implementation cost, we refer to the number of 2-input XOR gates ($\#xor$) and the number of the 2-to-1 multiplexers ($\#mux$) needed to implement the tweakey schedule of both encryption and decryption together. Note that for encryption, the tweakey TK_i is added to the state, while in decryption, $\texttt{MC}(TK_i)$. Hence, $\#mux$ is only considered when the implementation should support both encryption and decryption. It is noteworthy to emphasize that the numbers are given for $\#xor$ and $\#mux$ are those required for one column of bits. Hence for one complete block, we need to multiply these values by 16.

TABLE 3.4: Implementation Cost and Security Properties of the Candidate Matrices for CRAFT MC.

i	$\#xor$	$\#mux$	r_1	r_2
12	2	4	8	10
13	3	2	7	9
14/15	3	3	7	11
16/17	4	2	5	16
18	4	3	6	9
19/20	4	3	6	8
21	4	4	5	16
22	4	4	6	8
23	4	4	5	8
24	5	3	4	8
25	5	3	4	11
26	7	4	4	8
27/29	8	4	4	8
28	8	4	4	7

From the remaining matrices, M_{12} is the smallest one with respect to its implementation cost in the encryption-only case, while M_{13} is the smallest if both encryption and decryption should be supported. Since M_{13} can provide better security properties, it is chosen as the MixColumn matrix of CRAFT.

Over all involutory permutations, only the following candidate for P provides the reported r_1 and r_2. More precisely, it attains full diffusion after 7 rounds and assures at least 32 active S-boxes after 9 rounds in both differential and linear attacks.

$$P = [15, 12, 13, 14,\ 10, 9, 8, 11,\ 6, 5, 4, 7,\ 1, 2, 3, 0].$$

This permutation replaces the nibbles in the first row with the nibbles in the last row and also the nibbles in the second row with those in the third row. Then, it does a right (resp. left) shift in the first (resp. fourth) row and a shuffle in the second and third rows (see Figure 3.15).

Round Constants

We decided to use LFSRs to generate the round constants since compared to a randomly chosen set of round constants, an LFSR usually leads to lower implementation cost. Further, to make it efficient for the considered fault-detection mechanism, the LFSR size is restricted to 4 bits, as in the underlying code $[4+q, 4, d]$ (due to the S-box size). The LFSR can also be considered as the round counter. This implies that the period of the LFSR should be larger than the number of rounds. As an n-bit LFSR has a maximum period of $2^n - 1$, one 4-bit LFSR does not suffice. Hence, it is decided to use two LFSRs, one with a 4-bit and one with a 3-bit state. Using primitive polynomials for their feedback functions, the joint period of the LFSRs can reach $15 \cdot 7 = 105$, which is more than the need. While there is only one primitive polynomial for a 3-bit LFSR ($x^3 + x + 1$), there are two choices for the 4-bit one ($x^4 + x + 1$ and $x^4 + x^3 + 1$). Concerning the size of their hardware implementation, there is no preference between the 4-bit polynomials. We have just chosen $x^4 + x + 1$ as the polynomial for the 4-bit LFSR.

In every round, the states of 4-bit and 3-bit LFSRs are added to the fourth and fifth nibbles of the state of the cipher. These two positions are chosen since – considering our selected linear layer – nibbles in the first/second row of the state have full diffusion after 5/6 rounds; they get involved in the entire state nibbles after 5/6 rounds, while this happens after 7 rounds if the round constants are added to the third row. Although the first row has the fastest diffusion, adding round constants to this row causes larger latency in each clock cycle than adding them to the second row. Hence, it is decided to add them in the fourth and fifth nibble of the cipher.

Key and Tweak Schedule

To make the key schedule of the cipher small and lightweight, it is decided to use the same round keys in an alternating way. In particular, by separating the 128-bit master key into two 64-bit halves K_0 and K_1, using 64 multiplexers K_0 is selected in the even rounds and K_1 in the odd rounds. This is beneficial in a round-based implementation (as it is our target design architecture) since no extra register is required to process and generate the round keys. The same technique has been used in PICCOLO and MIDORI. As a side note, to make sure that the first and the last round keys are different, the total number of rounds has to be even.

To use the same concept for the tweak schedule, it is decided not to use an arbitrary update function for the tweak. Instead, using a permutation Q on the nibbles position of the tweak T, $\mathtt{QN}(T)$ is calculated, and iteratively they are added to the round key; T is used in the first two rounds, and $\mathtt{QN}(T)$ for the next two rounds and iterate this order. In a round-based implementation, this needs only 64 XOR and 64 MUX extra logic.

To find a permutation Q that provides the highest security, we evaluated the necessary number of rounds to ensure 32 active S-boxes concerning the related-tweak differential attack similar to the one by Kelsey, Schneier, and Wagner [KSW96]. On the other hand, to make Time-Data-Memory trade-off attacks less powerful, it is decided to use a circular permutation. The reason for this restriction is explained in Section 6.1.2 in more detail.

[KSW96] Kelsey, Schneier, and Wagner "Key-Schedule Cryptanalysis of IDEA, G-DES, GOST, SAFER, and Triple-DES"

Since the search space for a circular permutation is $15! \approx 2^{40}$, it is not possible to check all permutations. Hence, it is decided to choose about one thousand randomly generated permutations and examine their security. We found that the permutation given in Section 3.3.2 guarantees 32 active S-boxes in 13 rounds, which is less than the one for other permutations (See Section 6.1.3 for more detail).

3.3.4 *Hardware Implementation*

As stated before, the target is a round-based implementation, where one round of the cipher is completed at every clock cycle. This leads to the design architecture shown in Figure 3.16, which supports both encryption and decryption functionalities. To this end, it is just necessary to add a MixColumn module through the round tweakey path and a multiplexer to decide whether the selected round tweakey TK_i or $\mathtt{MC}(TK_i)$ should be given to the round function. It is noteworthy that since MixColumn does not change the third and fourth rows (see Section 3.3.2), 32 bits of TK_i and $\mathtt{MC}(TK_i)$ are

the same. This saves 32 multiplexers. In total, supporting decryption (only for the round tweakeys) costs $3 \times 16 = 48$ XOR gates and 32 multiplexers. To provide the round constant for decryption, it is necessary to implement each LFSR update function with both forward and backward functionalities, selected by the encryption signal bit $\mathtt{E}$.[17] Hence, it is necessary to initialize the LFSRs[18] with either $(0,0,0,1,0,0,1)$ or $(1,0,0,0,1,0,1)$ for encryption or decryption, resp. (see Table 3.2). As given in Section 3.3.3, each LFSR for either forward or backward needs only one XOR gate for the feedback function. This means that supporting decryption needs 2 extra XOR gates and 7 multiplexers. Note that different initial values for encryption and decryption can be generated by the $\mathtt{E}$ control signal and only one NOT gate. More precisely, $(\bar{\mathtt{E}}, 0, 0, \mathtt{E}, \bar{\mathtt{E}}, 0, 1)$ generates $(0,0,0,1,0,0,1)$ for encryption $(\mathtt{E} = 1)$ and $(1,0,0,0,1,0,1)$ for decryption $(\mathtt{E} = 0)$. In total, turning an encryption-only implementation of CRAFT into encryption-and-decryption costs at most 50 XOR gates, 39 multiplexers, and one inverter. The hardware synthesizers usually merge the logic and even achieve a smaller overhead. Independent of whether decryption is supported by an implementation, adding the tweak support needs 64 multiplexers to choose T or $\mathtt{QN}(T)$ and 64 XOR gates to add the tweak to the K_i.

For the implementations, the Synopsys Design Compiler with the IBM 130 nm ASIC standard cell library is used. The result of pure implementations (not protected against either SCA or DFA attacks) is shown in the first column of Table 3.5. Surprisingly, the only-encryption CRAFT without tweak needs less than 1000 GE, which – to the best of our knowledge – is a record for a round-based implementation with a 64-bit state and 128-bit key. It is important to highlight that due to the key-alternating fashion of the key schedule, registers are not dedicated to the key state. Instead, large multiplexers are used (see Figure 3.16) to select the corresponding 64-bit round key. The same approach has been used in the design and implementation of MIDORI [Ban+15], PICCOLO [Shi+11], and KTANTAN [DDK09]. Also, for the key state, registers are not used in the implementations of these three ciphers, but under the same condition CRAFT outperforms MIDORI and PICCOLO with a considerable distance (see Table 3.5). It is noteworthy that for comparison, all considered ciphers are implemented following a unique design architecture and implementation fashion that allows us 1) to synthesize all of them under the same ASIC library,[19] and 2) to apply the underlying countermeasure to DFA attacks enabling a fair comparison. Furthermore, the extraordinary scan flip-flops – an optimized combination of a flip-flop and a multiplexer – are not used in the implementations. Therefore, the area footprints given in Table 3.5 do not necessarily fit the numbers reported in original documents, each of which is synthesized by a different library.

Serialized architectures (e. g., nibble-serial) have been used in several lightweight ciphers to achieve a low area footprint but with high latency. For example, a bit-serial implementation of SIMON with an area footprint of 958 GE needs 2816 clock cycles [Bea+15]. Serializing CRAFT would need extra registers for the key state to provide the key bits/nibbles/bytes per clock cycle. This implies that a serialized CRAFT with high latency needs more area than its round-based variant accomplishing the encryption-and-decryption

[17] If $\mathtt{E} = 1$, then it does the encryption; otherwise, it does the decryption.

[18] Note that state of the LFSRs is $(a_0, a_1, a_2, a_3, b_0, b_1, b_2)$.

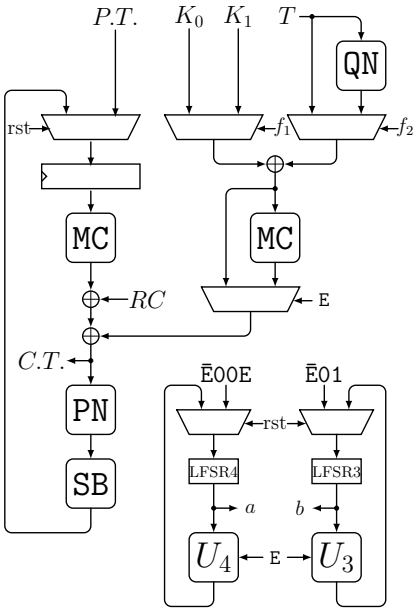


FIGURE 3.16: Round-based Design Architecture of CRAFT Supporting Tweak, Encryption and Decryption. Note that RC, f_1, and f_2 are functions of outputs of the LFSRs, i. e., a and b.

[19] Synthesizing with different ASIC libraries can lead to different results [Jea+17].

in 32 clock cycles. Notably, the critical path delay of CRAFT is higher than that of PRESENT, GIFT, and SIMON. This is because, in such ciphers, the diffusion layer is realized by a bit permutation, which induces no delay. In the comparisons, KATAN and KTANTAN with 64-bit state are included, although their 80-bit key size and the high number of 762 clock cycles per encryption do not match the other considered ciphers. Furthermore, SKINNY with a 192-bit tweakey is included, which is compatible with CRAFT supporting a 64-bit tweak.

As a side note, it can be seen that CRAFT is smaller than MIDORI while they share some components. At the same time, CRAFT needs a higher number of clock cycles (32 versus 17). However, the whole latency (of the entire encryption) of the slowest CRAFT (with tweak and decryption support) is smaller than that of MIDORI (127.68 ns versus 128.69 ns). Of course, due to the higher number of clock cycles, CRAFT cannot outperform MIDORI with respect to energy consumption per encryption.

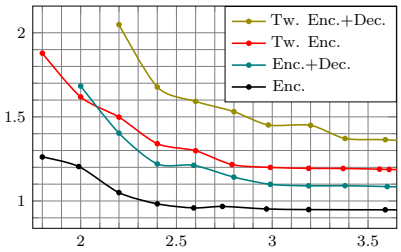


FIGURE 3.17: Latency vs. Area of Round-based CRAFT, using the IBM 130 nm ASIC Library. X axis is the latency in nano-seconds and Y axis is the area in 1000 GE.

[20]Due to an NDA, the full name of the used library is omitted.

Note that the clock period is not restricted in the syntheses allowing the synthesizer to achieve the smallest possible area. However, it is possible to force the synthesizer to reach a certain maximum latency, leading to higher area requirements. As a reference, Figure 3.17 shows such results for round-based implementations of CRAFT. Again, the IBM 130 nm ASIC standard cell library is used due to its public availability and the fact that it is used to benchmark SIMON area footprints in [Bea+15].

To give an overview of the performance comparisons under a more modern ASIC library, all the above syntheses are repeated using a commercial 40 nm standard cell library.[20] The corresponding results are shown in Table 3.6.

3.3.5 *Protection against DFA Attacks*

As stated before, the focus is on the fault-detection technique proposed in Section 3.1.1 where two adversary models are defined: the univariate adversary model $\mathbf{M}_t$, where at each encryption process the adversary can make at most t cells of the entire circuit faulty, and the multivariate adversary model $\mathbf{M}_t^*$, which is bounded to t faulty cells in the entire circuit at every clock cycle.

It is important to emphasize that this implies that the safe-error [YJ00] and stuck-at-0/1 [Cla07] models are not covered. Further, the fault-protected implementations do not necessarily provide security against FSA [Li+10] and SIFA [Dob+18].

RESULTS: Four cases for the redundancy size of each S-box is considered: $q \in \{1,2,3,4\}$ where $[5,4,2]^{16}$, $[6,4,2]^{16}$, $[7,4,3]^{16}$, and $[8,4,4]^{16}$-codes are applied, resp. The corresponding design architectures for $q < 4$ and $q \geq 4$ are shown in Figures 3.18 and 3.19, resp. Generator matrices of these codes have been given in Section 3.3.3. This implies that with $q = 1$ and $q = 2$ (both leading to $d = 2$) the circuit is able to detect at most $t = 1$ faulty cell, i. e., protection against a $\mathbf{M}_{t=1}$ adversary. This is improved by larger $q = 3$ and $q = 4$ to protect against a $\mathbf{M}_{t=2}$ and $\mathbf{M}_{t=3}$ adversary, resp.

TABLE 3.5: Area (GE) and Latency (ns) Comparison of Round-based Implementations with a $[4+q,4,d]^{16}$-Code, using the IBM 130 nm ASIC Library.

Block Cipher	**Key Length**	**Clock Cycles**	**Plain**	$\mathbf{M}_t$				$\mathbf{M}_t^*$			
				$[\mathbf{5,4,2}]^{\mathbf{16}}$	$[\mathbf{6,4,2}]^{\mathbf{16}}$	$[\mathbf{7,4,3}]^{\mathbf{16}}$	$[\mathbf{8,4,4}]^{\mathbf{16}}$	$[\mathbf{5,4,2}]^{\mathbf{16}}$	$[\mathbf{6,4,2}]^{\mathbf{16}}$	$[\mathbf{7,4,3}]^{\mathbf{16}}$	$[\mathbf{8,4,4}]^{\mathbf{16}}$
			area latency	area latency	area latency	area latency	area latency	area latency	area latency	area latency	area latency
SKINNY	128	37	1738 3.66	3640 5.16	4494 5.24	5636 6.09	6804 6.37	4236 7.84	5320 8.73	6879 8.85	8477 9.60
LED	128	49	1664 9.15	4499 9.57	5264 9.17	6699 10.04	8718 12.80	4813 13.11	5729 12.46	7359 12.96	9637 15.89
MIDORI	128	17	1372 7.57	3282 8.25	3942 8.16	5262 8.87	6840 10.40	3615 11.18	4358 11.42	5891 11.52	7693 14.30
PRESENT	128	32	1767 2.93	4211 5.19	5177 5.62	6639 6.32	8219 6.71	4792 7.83	6015 8.53	7899 8.87	9896 9.37
GIFT	128	29	1587 2.88	3824 5.11	4722 5.32	6082 6.11	7767 6.61	4420 7.49	5548 8.43	7325 8.56	9432 9.01
SIMON	128	45	1629 2.86	3614 5.20	4487 5.27	5621 5.93	7603 6.44	4211 7.28	5311 7.86	6866 8.03	9277 9.97
PICCOLO	128	32	1462 7.69	3870 9.86	4763 9.47	6241 10.10	8217 12.15	4196 12.81	5123 12.56	6873 12.79	9062 15.30
KATAN	80	762	1080 3.87	2946 5.84	3610 6.04	4746 6.69	6684 7.49	3450 8.41	4293 9.07	5776 9.98	8092 9.38
KTANTAN	80	762	601 4.23	2039 5.38	2457 5.07	3069 5.47	4207 6.01	2373 9.45	2881 9.28	3710 9.82	5073 10.94
CRAFT	128	32	**949** 3.19	**2342** 5.13	**2857** 5.25	**3698** 5.56	**5014** 6.26	**2670** 7.59	**3269** 8.59	**4325** 8.22	**5864** 10.18
CRAFT+Dec.	128	32	**1089** 3.60	**2609** 5.07	**3169** 5.40	**4069** 6.19	**5459** 6.43	**2938** 9.08	**3583** 9.38	**4699** 9.09	**6336** 9.95
CRAFT+Tw.	128	32	**1193** 3.37	**2801** 5.15	**3420** 5.27	**4518** 5.56	**6657** 6.25	**3129** 8.54	**3833** 8.74	**5148** 8.23	**7507** 10.43
CRAFT +Dec.+Tw.	128	32	**1339** 3.99	**3066** 5.36	**3731** 5.43	**4891** 6.09	**7110** 6.64	**3397** 9.12	**4145** 9.81	**5520** 9.98	**7982** 11.21
SKINNY	192	41	2206 4.00	4540 5.63	5656 5.74	7119 6.34	8553 6.74	5272 7.69	6690 8.26	8676 8.79	10640 9.34

TABLE 3.6: Area (GE) and Latency (ns) Comparison of Round-based Implementations with a $[4+q,4,d]^{16}$-Code, using a 40 nm Commercial ASIC Library.

Block Cipher	**Key Length**	**Clock Cycles**	**Plain**	$\mathbf{M}_t$				$\mathbf{M}_t^*$			
				$[5,4,2]^{16}$	$[6,4,2]^{16}$	$[7,4,3]^{16}$	$[8,4,4]^{16}$	$[5,4,2]^{16}$	$[6,4,2]^{16}$	$[7,4,3]^{16}$	$[8,4,4]^{16}$
			area latency	area latency	area latency	area latency	area latency	area latency	area latency	area latency	area latency
SKINNY	128	37	2041 1.28	4152 2.55	5154 2.61	6457 2.89	7773 2.88	4835 3.50	6118 3.56	7906 3.84	9714 4.15
LED	128	49	1940 3.98	4906 4.32	5836 4.31	7470 4.54	9658 5.48	5286 4.87	6314 4.99	8197 5.46	10638 6.57
MIDORI	128	17	1616 3.31	3675 3.78	4421 3.40	5938 3.78	7724 4.29	4047 4.51	4896 4.86	6658 4.85	8693 5.83
PRESENT	128	32	2050 1.04	4685 2.53	5844 2.61	7502 2.90	9270 2.96	5369 3.51	6807 3.56	8949 3.87	11221 4.07
GIFT	128	29	1848 0.86	4322 2.31	5367 2.58	6904 2.81	8817 3.00	5006 3.44	6331 3.52	8352 3.79	10761 3.92
SIMON	128	45	1984 0.93	4052 2.43	5076 2.55	6371 2.76	8657 2.91	4735 3.14	6039 3.23	7820 3.51	10600 4.30
PICCOLO	128	32	1631 3.21	4269 4.22	5235 4.31	6945 4.84	9185 5.11	4648 5.52	5712 5.56	7662 6.14	10151 7.18
KATAN	80	762	1236 1.60	3274 2.29	4047 2.36	5342 2.70	7544 2.98	3853 3.42	4844 3.82	6543 4.05	9158 4.51
KTANTAN	80	762	710 1.60	2308 2.36	2792 2.46	3493 2.75	4785 2.71	2696 3.80	3289 3.77	4242 4.08	5794 4.71
CRAFT	128	32	**1091** 1.66	**2660** 2.58	**3253** 2.86	**4223** 3.01	**5668** 2.89	**3041** 3.34	**3740** 3.57	**4961** 3.78	**6668** 4.24
CRAFT+Dec.	128	32	**1246** 2.08	**2948** 2.64	**3595** 2.73	**4627** 2.87	**6149** 2.92	**3327** 3.69	**4077** 3.74	**5363** 3.68	**7149** 4.50
CRAFT+Tw.	128	32	**1355** 1.81	**3167** 2.58	**3880** 2.86	**5149** 3.01	**7503** 2.89	**3548** 3.51	**4368** 3.69	**5887** 3.96	**8508** 4.45
CRAFT Dec.+Tw.	128	32	**1529** 2.02	**3454** 2.65	**4220** 2.73	**5548** 2.88	**7981** 2.92	**3833** 3.96	**4705** 3.62	**6283** 3.85	**8983** 4.73
SKINNY	192	41	2592 1.33	5146 2.16	6453 2.24	8118 2.51	9735 2.62	5982 3.53	7658 3.58	9930 3.86	12165 4.17

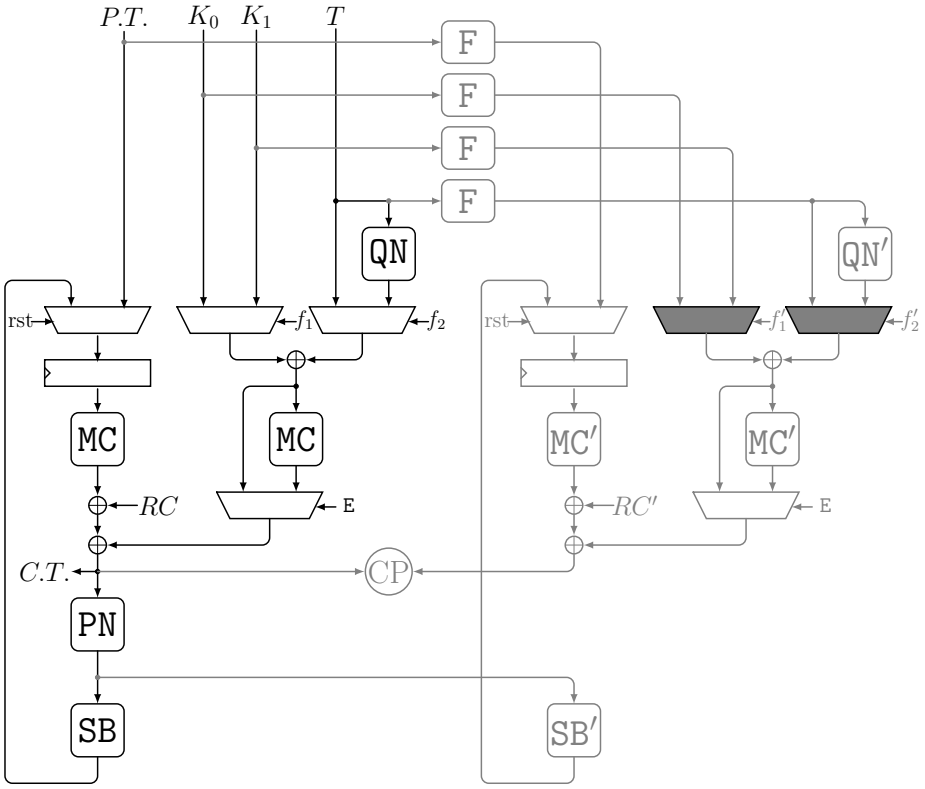


FIGURE 3.18: Round-based Design Architecture of CRAFT with Fault Detection, $q < 4$. Note that the control unit is not shown.

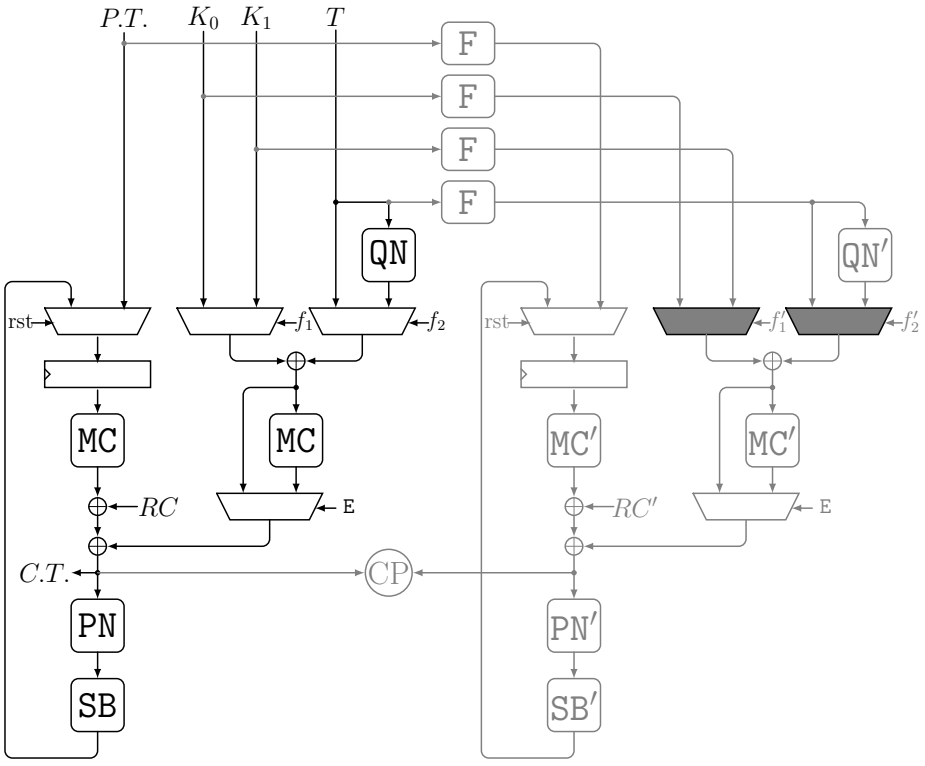


FIGURE 3.19: Round-based Design Architecture of CRAFT with Fault Detection, $q = 4$. Note that the control unit is not shown.

Note that MC of CRAFT has been chosen to 1) let MC operate solely on the redundant part of information for $q < 4$ (see Figure 3.18), and 2) avoid any necessary extra check point at MC input (see Lemma 63 and Theorem 64). This, in addition to the LFSRs with at most 4-bit width (since $s = 4$), helps us to realize such implementations with a low area overhead. The performance figure and area requirement of several implementations compared to that of other ciphers are listed in Tables 3.5 and 3.6. It can be seen that CRAFT outperforms all other considered ciphers with compatible state and key size even when CRAFT supports both encryption and decryption.

It might be thought that the fault-protected implementations of CRAFT are smaller than the others since its unprotected variant is smaller. Tables 3.5 and 3.6 show the inconsistency of this statement. As an example, unprotected LED and GIFT need less area compared to SKINNY, while their fault-protected variants are larger.

According to Section 3.2.4, to provide security against a multivariate adversary $\mathbf{M}_t^*$, extra check points should be defined, and the consistency check module needs to be adjusted. Doing so, CRAFT again achieves the smallest area overhead under all considered settings. The results are given in Tables 3.5 and 3.6.

For further protection against the attacks such as SIFA [Dob+18], an interesting topic is to add error-correction capabilities. Since our underlying fault-detection scheme is based on the application of binary linear codes, it might be promising to adjust the same principle to correct faults.

3.4 FAULT CORRECTION STRUCTURE

In this section, we introduce a methodology that leads to the secure implementation of CEC-based CEC schemes in the presence of fault injection. Indeed, it is an extension of the CED scheme from Section 3.2 to error correction by keeping the same adversary model. The goal is to guarantee the correction of any faults injected inside the circuit by a $\mathbf{M}_t$ or $\mathbf{M}_t^*$ adversary models as defined in Definitions 53 and 54.

After the introduction of SIFA, some techniques have been proposed to counter it. *Binary repetition code* as a basic ECC is used in [Bre+20; Sah+19], where the correction is performed in the non-linear S-box Layer where only a 4-bit S-box has been equipped with single-bit correction. Each S-box output bit is instantiated multiple times and then fed into a Majority Voting (MV) circuitry. In [Dae+19], a combined countermeasure against SCA and single-bit fault SIFA has been proposed, in which the non-linear functions are implemented by *Toffoli* gates, and the whole design must be masked.

In comparison, the methodology introduced in the following deals with error correction and does not force the designer to apply masking. In fact, this scheme's application does not increase the difficulty of equipping the design with a masking countermeasure, as the algebraic degree of the underlying functions stays unchanged.

[Bre+20] Breier, Khairallah, Hou, and Liu, "A Countermeasure Against Statistical Ineffective Fault Analysis"

[Sah+19] Saha, Jap, Roy, Chakraborti, Bhasin, and Mukhopadhyay, *Transform-and-Encode: A Countermeasure Framework for Statistical Ineffective Fault Attacks on Block Ciphers*

[Dae+19] Daemen, Dobraunig, Eichlseder, Gross, Mendel, and Primas, *Protecting against Statistical Ineffective Fault Attacks*

The Correction Point

To correct a faulty codeword, consider the typical ECC construction shown in Figure 3.20. Assume a faulty input codeword $(x \oplus e \| x' \oplus e')$ with the injected fault $(e \| e')$. At the correction point, the syndrome decoder is fed by $F(x \oplus e) \oplus x' \oplus e' = F(e) \oplus e'$; hence, the injected fault $(\tilde{e}|\tilde{e}')$ is predicted. If $wt(e) + wt(e') < d/2$, the predicted fault is the same as the injected one. If so, XORing $(\tilde{e}|\tilde{e}')$ to the input word omits the injected fault. Note that in Figure 3.20, the syndrome decoder is considered in two parts with SD_1 and SD_2 predicting $\tilde{e}$ and $\tilde{e}'$, resp.

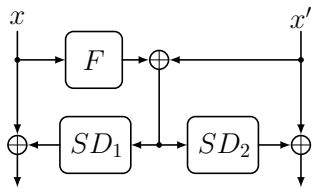


FIGURE 3.20: Correction Point using an ECC.

3.4.1 *The Parity Function and the General Structure*

Consider the general algorithm depicted in Figure 3.4 that is realized by a sequential circuit to apply the strategy. The application of a $[k+m, k]$-code, similar to the CED schemes, depending on if function F is injective, there are two possibilities to implement CEC structure. In the following, only a univariate adversary model is considered, and later a solution is given to extend it to a multivariate adversary model.

Injective F

In this case, as shown in Figure 3.21, the redundant part of the circuit can operate on x' independent of x. The redundant function T' can also be achieved as $T' \circ F = F \circ T$. It is necessary to put a correction point at the input of each operation. Otherwise, the faults injected at the register cells would potentially propagate to multiple output bits of T or T'.

All output bits of each dashed box in Figure 3.21 must be implemented fulfilling the independence property, which may necessitate implementing several instances of F, SD_1 and SD_2. Note that in a code with distance $d = 2t+1$, the output of SD_1 does not change if up to t faults are injected at its input. Thus, F and the corresponding XOR can be instantiated separately, while it does not hold for SD_2 (see Figure 3.21).

Since the multiplexer and the register operate on each bit of the T or T' output independently, they fulfill the independence property automatically. Hence, any fault injected on T or T' fitting to the adversary model is corrected at the next clock cycle's correction point.

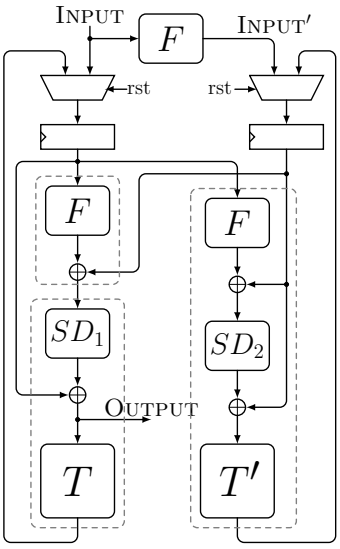


FIGURE 3.21: Proposed ECC-based CEC Scheme using an Injective F.

Non-Injective F

Here, T' needs to receive the original data x to compute $T'(x) = F \circ T(x)$. This means that we only need to correct x. The corresponding construction is shown in Figure 3.22, where the independence property can be divided into two parts (two dashed boxes). It is indeed the same as the left part of Figure 3.21 with a different T'. This can be beneficial to limit the area overhead due to the fact mentioned above as x' is not used after the syndrome decoder.

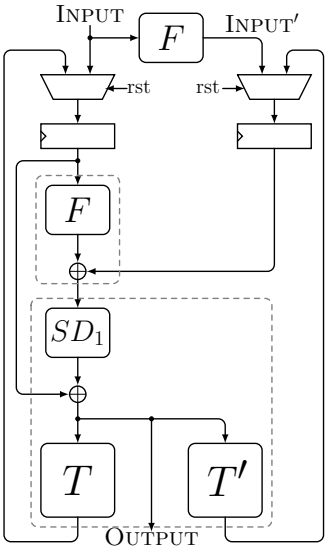


FIGURE 3.22: Proposed ECC-based CEC Scheme using a Non-Injective F.

3.4.2 *Application for an SPN Block Cipher*

Applying the proposed CEC scheme for an SPN block cipher is quite similar to applying CED explained in Section 3.2.2. By decomposing the round functions $\mathtt{R}_i$ to its operations (i. e., $\mathtt{A}_{K_i}$, $\mathtt{S}_i$ and $\mathtt{L}_i$), it is possible to apply the proposed CEC scheme by fulfilling the independence property for each part. In this case, generally, we need two correction points, one for the S-box layer and one for the linear layer.[21] It is worth mentioning that the benefit of decomposing might be marginal since more correction points are required (it depends on the diffusion of $\mathtt{S}_i$ and/or $\mathtt{L}_i$). Figure 3.23 depicts the CEC scheme for an SPN block cipher that an injective function F is considered .

Using an $[s+q, s, d]^m$-code not only makes it easier to implement $\mathtt{S}_i$ but also results in a smaller footprint for the syndrome decoder.[22]

Besides, similar to the CED schemes, if the linear layer of an SPN block cipher is realized by binary multiplication in $\mathbb{F}_{2^s}$, there is no need for an extra correction point for the linear layer.

3.4.3 *Control Signals, Multiplexers, and the Output*

CONTROL SIGNALS: The statements given for different constructions depending on F hold for the FSM as well. Protecting FSM and its control signals in a CEC scheme is similar to protecting them in the CED scheme (see Figures 3.12 and 3.13). The only difference is that instead of a check point, we need to use a correction point as it is explained in Section 3.4.1.

MULTIPLEXERS: Suppose that a multiplexer is switching between a k-bit x and y controlled by a single-bit signal s_i.[23] As stated before, the corresponding parities of s_i, x, and y are m-bit and are denoted by s_i', x' and y', resp.

In a CED scheme, a multiplexer and its corresponding one in the redundant counterpart could separately operate between (s_i, x, y) and (s_i', x', y'), resp. But for a CEC scheme, this is not possible to operate separately. To do this, the multiplexer is combined with its redundant counterpart to switch between $n/m+q$-bit $(x\|x')$ and $(y\|y')$ by an $q+1$-bit control signal $(s\|s')$.[24] This means the combined multiplexer must be implemented by a multiplexer tree in $q+1$ levels.

Assuming that the output of the combined multiplexer for $q+1$-bit signal value i is v_i (i. e., v_is are the inputs of the multiplexer), then we have the following equations:

$$v_i = \begin{cases} (x\|x') & i = \big(\mathtt{0}\|F(0)\big) \oplus \delta \text{ with } \mathrm{hw}(\delta) < d/2, \\ (y\|y') & i = \big(\mathtt{1}\|F(1)\big) \oplus \delta \text{ with } \mathrm{hw}(\delta) < d/2, \\ 0 & \text{otherwise.} \end{cases}$$

If the functions generating the control signals and their redundant part fulfill the independence property, this construction guarantees the correction of $t < d/2$ faulty gates, since $(s\|s')$, is a codeword with distance d.

Note that any arbitrary random values can be given to those inputs v_i, which are tied to zero without affecting the fault coverage. But if the output for such v_i values is a constant value (like above that is fixed to 0 value),

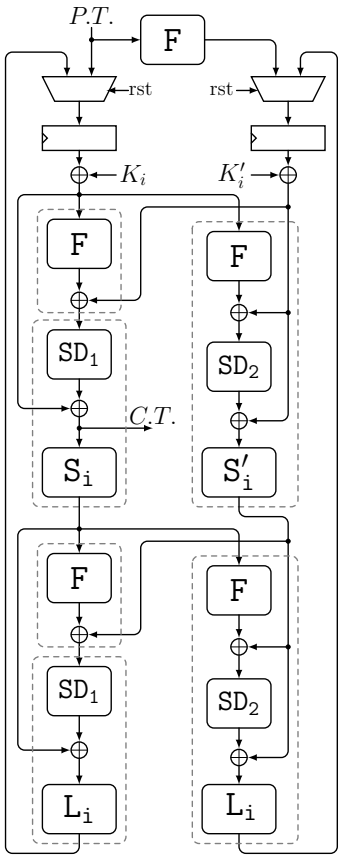


FIGURE 3.23: Proposed CEC Scheme of an SPN Block Cipher using Injective F.

[21] Note that for the same reason as discussed in 3.2.2, there is no need for inserting another correction point for the $\mathtt{A}_{K_i}$.

[22] Usually, the syndrome decoder of a code with a larger rank is much larger than the one for a code with a smaller rank.

[23] Of course, the structure can be extended for multiplexers using larger signal bits.

[24] Note that n/m is the S-box size.

then some of the 2-to-1 multiplexers have equal inputs, which means that the output is fixed and does not depend on the select input. The synthesizer usually removes them to optimize the circuit area-wise, which does not affect the fault propagation and hence the construction's fault coverage.

Note that the signals provided externally (i. e., through I/O port like plaintext, reset signal, or external control signals) cannot be protected internally. As a result, the construction's external signals are not encoded, and the same signal is used in redundant counterparts.

OPTIMIZATION: Instead of using the same F function as the parity function of $[k+p,k,d]$ for the main structure, one possibility for the multiplexers is using $[d,1,d]^k$-code. Therefore, $(s_i \| s'_i)$ can be either $(0,\ldots,0) \in \mathbb{F}_2^d$ or $(1,\ldots,1) \in \mathbb{F}_2^d$. Since both codes' distance is the same, it is possible to correct all faults with a Hamming weight of $t < d/2$. Thus, it achieves the same goal but with a smaller area overhead.

THE OUTPUT: Note that the circuit contains a control signal DONE which indicates the termination of the computation. To avoid sniffing intermediate results by injecting a fault into such a signal, a construction similar to the multiplexer can be used. In this case, the multiplexer outputs the ciphertext if there is no fault injected, i. e.,

$$v_i = \begin{cases} \text{CIPHERTEXT} & i = \big(1 \| F(1)\big), \\ \perp & otherwise. \end{cases}$$

Considering the independence property, this construction guarantees to prevent any sniffing with $t < d$ faults on DONE and its redundancy DONE′.

3.4.4 *Extension to Multivariate*

Suppose the circuits shown in Figures 3.21 and 3.22 with a $[k+m,k,d]$-code, which under the $\mathbf{M}_t$ adversary model corrects all t faults when $t < d/2$. To protect the circuit against the $\mathbf{M}_t^*$ adversary model, $d/2$ must be larger than the maximum number of faults that the adversary can inject between two consecutive correction points. For instance, in the circuits shown in Figures 3.21 and 3.22, there is only one correction point in each clock cycle; hence, the maximum number of faults that can be injected between two consecutive correction points is $2t$. To protect this circuit against the $\mathbf{M}_t^*$ adversary model, we need to embed a code with $d > 4t$.

It is noteworthy to mention that for circuits that include more than one correction point in one clock cycle (like Figure 3.23), only changing the first correction point to use a code with $d = 4t+1$ is enough, while other correction points can still use the code with $d = 2t+1$.

3.4.5 *Case Study and Implementation Results*

To evaluate the overhead and the fault correction capability of the proposed methodology, we choose the CRAFT block cipher (introduced in Section 3.3) as the case study. The reason behind this is that this block cipher is designed for efficient protection against DFA.

The focus here is the round-based implementation of CRAFT. Because of the S-box size, ECC codes are fixed to $[4+q, 4, d]^{16}$ ones where the considered adversary model defines the distance d. Two codes with distance 3 and 5 are examined to be able to correct 1- and 2-bit faults, resp.: $[7, 4, 3]$-code for 1-bit and $[11, 4, 5]$-code for 2-bit corrections in the implementation.

- $[7, 4, 3]^{16}$. Here, F cannot be injective, and the architecture shown in Figure 3.22 is used where the following parity matrix is applied.

$$P_1 = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 0 \end{bmatrix}.$$

  Note that since the linear layer is a binary multiplication, the corresponding linear layer in the redundant part can operate solely.

- $[11, 4, 5]^{16}$. Here, F is injective, and the structure shown in Figure 3.22 is used to apply the following parity matrix.

$$P_2 = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 \end{bmatrix}.$$

- $[d, 1, d]^{64}$ with $d = 3$ or 5, i. e., MV. These cases are included in the investigation to compare the proposed methodology and the common and straightforward MV schemes where the cipher is instantiated $2t+1$ number of times to correct up to t bit faults. To this end, the cipher is instantiated for 3 and 5 times together with the corresponding voting circuitry.

TABLE 3.7: Area (GE) and Latency (ns) of the Implementation of CRAFT Block Cipher with Proposed CEC Scheme, using NanGate 45 nm ASIC Library.

	Area	**Latency**
Plain	1097	0.55
3-bit MV	4502	1.00
$[7, 4, 3]^{16}$	5187	0.87
5-bit MV	7711	1.03
$[11, 4, 5]^{16}$	21617	1.08

The implementations' area and latency are summarized in Table 3.7 that is again borrowed from [SRM19]. It should be noted that only encryption without tweak of CRAFT is considered. Besides, for the *plain* version of the implementation, keeping hierarchy is not used, which means it does not fulfill the independence property.

It can be seen that the proposed scheme has almost no performance benefits compared to the classical MV variants. However, as stated in [Dob+18] and shown by using simulated attacks in [SRM19], faults can be injected on multiple instances of MV with less complications, leading to successful SIFA attacks.

4

Low-Latency Designs

"It is wicked fast, and near impossible to see."
—Oliver explaining the Golden Snitch in "Harry Potter and the Sorcerer's Stone" by J. K. Rowling.

▸ The Need for a Secure and Efficient Low-Latency Block Cipher which also has low-power and low-energy requirements is ever increasing with the widespread of multiple technologies using micro-controllers. While PRINCE [Bor+12] was specifically designed to tackle this problem, the recent lightweight crypto competition for AEAD from the NIST set a specific security level that PRINCE cannot reach. As a low-latency cipher would probably be deployed in a larger environment using such an AEAD primitive, it makes sense to want this low-latency cipher to reach the same security level.

[Bor+12] Borghoff et al., "PRINCE - A Low-Latency Block Cipher for Pervasive Computing Applications - Extended Abstract"

In Section 4.1, we show how to modify PRINCE to reach the NIST's required security level while minimizing the induced overhead, especially in a situation where PRINCE is already deployed. This problem is solved by showing that a carefully built key-schedule is sufficient to provide the required security goal while keeping (almost) all of the remaining design untouched and propose the block cipher PRINCEv2. As proven by various provided experiments, PRINCEv2 only has a very small overhead compared to PRINCE while still reaching the required higher security level. Moreover, the fact that the PRINCE and PRINCEv2 designs are very similar allows one to implement both PRINCE and PRINCEv2 in the same environment (e. g., for backward compatibility) with a very small overhead.

Section 4.1 is based on the article [Bož+20] that the author of the thesis contributed with Dušan Božilov, Maria Eichlseder, Miroslav Knežević, Baptiste Lambin, Gregor Leander, Thorben Moos, Ventzislav Nikov, Yosuke Todo, and Friedrich Wiemer. Explicitly, the author of the thesis was the main contributor in the design and cryptanalysis of the block cipher PRINCEv2. It is important to mention that the author has not contributed to any hardware implementations but to illustrate the results of the theoretical contributions, they are borrowed from [Bož+20].

[Bož+20] Božilov, Eichlseder, Knežević, Lambin, Leander, Moos, Nikov, Rasoolzadeh, Todo, and Wiemer, "PRINCEv2: More Security for (Almost) No Overhead"

As of the second direction of the chapter, in Section 4.2, we study the latency of the S-boxes. First, we introduce a new metric called latency complexity to measure a given Boolean function's latency mathematically. Besides, we introduce an equivalency class of the Boolean functions that the latency complexity of the functions within the same class is invariant under this equivalency. Then, based on the latency complexity, we present an algorithm to efficiently find all the Boolean functions with low-latency complexity. Later, using the low-latency balanced Boolean functions and an efficient

algorithm, we build up to 6-bit bijective low-latency S-boxes. As a result, we present several 5- and 6-bit low-latency S-boxes with cryptographically good properties.

Section 4.2 is based on a pre-print article that the author of the thesis wrote together with Thorben Moos.

4.1 PRINCEV2: NEW KEY SCHEDULE FOR PRINCE

This section describes the result of our efforts to increase the security margins of PRINCE without significantly increasing the latency, area, power, or energy consumption. Recalling that the PRINCE security claim is as follows:

Security Claim 2 (PRINCE [Bor+12]). *An adversary with 2^n chosen plaintext-ciphertext pairs needs at least 2^{126-n} calls to the encryption function to recover the secret key.*

This security level is sufficient for most applications that need a low-latency cipher, yet we set ourselves to explore design opportunities when facing the security requirements NIST put forward in its lightweight crypto competition.[1] Therefore, the targeted security level for PRINCEV2 is 112 bits when the data complexity is limited to 2^{50} bytes. It has to be noted that the NIST lightweight crypto competition does not focus on the design of low-latency block ciphers. Instead, it focuses on AEAD schemes which are, in general, too slow or too big compared to dedicated block ciphers, thus failing to meet the low-latency design challenges mentioned in Section 2.4.2.

[1] https://csrc.nist.gov/projects/lightweight-cryptography.

One design constraint is implementing PRINCEV2 on top of the existing PRINCE architecture without adding a significant area overhead or increasing the latency. Starting with this design constraint, it is tried to minimize the difference from PRINCEV2 to PRINCE. Besides minimizing the overhead of implementing one on top of the other, this has the convenient benefit that a lot of the security analysis that PRINCE received can then either directly be transferred to PRINCEV2 or transferred with small modifications.

To achieve the requested higher security level, a different key schedule is strictly necessary, as without a change there, the generic bound of the FX-construction applies. Through a carefully crafted and analyzed key schedule, it is possible to get a secure cipher meeting the NIST security requirements. Besides the change in the key schedule, only a single XOR is added in the middle rounds. This middle round was unkeyed in PRINCE. From an aesthetic perspective, the new key schedule has the drawback that the α-reflection property is slightly weakened. That is, decryption is not simply encryption with a modified key as in PRINCE but requires slightly more effort.

Besides being beneficial from a security point of view, our minimal changes result in only minimal performance changes in all the dimensions mentioned above. This makes PRINCEV2, in the unrolled architecture that is the goal, nearly as efficient as PRINCE while achieving a higher security level.

It is noteworthy to emphasize that the problem we are trying to solve in this section is quite general but not an easy task. It touches the least understood part of block cipher design, namely the design of the key-scheduling.

For an existing cipher with a potentially non-optimal key-scheduling (here PRINCE), this translates to the question of how to increase security while minimizing the resulting overhead. For the interesting – being deployed in several products – the case of PRINCE, we came up with an elegant, yet simple and efficient solution to this problem. This simplicity is an advantage concerning the objective.

4.1.1 *Specification*

As discussed above, the aim is to keep the changes to PRINCE minimal. The same round function is used to achieve this and only change the middle layer, key schedule, and the round constants compared to PRINCE. To be self-contained, PRINCE's general structure and the round function are briefly recalled before giving the updated parts for PRINCEV2.

PRINCE's Round Operations

It consists of a 64-bit block and a 128-bit key. The 128-bit key is split into two 64-bit keys K_0 and K_1. The state is viewed as a 4×4 square array of nibbles. The notation $I[i, j]$ denotes the nibble located at row i and column j of the state. One can also view this 4×4 square array as a vector by concatenating the columns. Thus, $I[i]$ denotes the nibble in the i-th position of this vector, i. e., $I[i, j] = I[4 \cdot j + i]$.

The encryption function iterates the round function $\mathtt{R}$ five times, then applies the middle layer $\mathtt{R}'$, followed by five applications of the inverse round function $\mathtt{R}^{-1}$. The round function itself applies an S-box layer SB, followed by a linear layer consisting of a MixColumn operation MC and a ShiftRows SR. The S-box applied in SB is given in Table 4.1.

TABLE 4.1: The 4-bit S-box of PRINCE and PRINCEV2.

x	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
$S(x)$	b	f	3	2	a	c	9	1	6	7	8	0	e	5	d	4

And the ShiftRows permutation applied in SR is the same used in the AES, i. e.,

$$P = [0, 5, 10, 15,\ 4, 9, 14, 3,\ 8, 13, 2, 7,\ 12, 1, 6, 11]\ ,$$

where for all $0 \leq i \leq 15$, the i-th nibble is replaced by $P[i]$-th one.

The MixColumns operation is built from the following four 4×4 matrices:

$$M_1 = \begin{bmatrix} 0&0&0&0 \\ 0&1&0&0 \\ 0&0&1&0 \\ 0&0&0&1 \end{bmatrix},\ M_2 = \begin{bmatrix} 1&0&0&0 \\ 0&0&0&0 \\ 0&0&1&0 \\ 0&0&0&1 \end{bmatrix},\ M_3 = \begin{bmatrix} 1&0&0&0 \\ 0&1&0&0 \\ 0&0&0&0 \\ 0&0&0&1 \end{bmatrix},\ M_4 = \begin{bmatrix} 1&0&0&0 \\ 0&1&0&0 \\ 0&0&1&0 \\ 0&0&0&0 \end{bmatrix},$$

or in other words M_i is the 4×4 identity matrix where the i-th row is replaced by the zero vector. From these M_i, we build the matrices $\widehat{M}_0$, $\widehat{M}_1$ and M' as follows.[2]

[2] In the PRINCE proposal paper [Bor+12], these matrices are denoted by $\widehat{M}^{(0)}$ and $\widehat{M}^{(1)}$, resp.

$$\widehat{M}_0 = \begin{bmatrix} M_1M_2M_3M_4 \\ M_2M_3M_4M_1 \\ M_3M_4M_1M_2 \\ M_4M_1M_2M_3 \end{bmatrix}, \quad \widehat{M}_1 = \begin{bmatrix} M_2M_3M_4M_1 \\ M_3M_4M_1M_2 \\ M_4M_1M_2M_3 \\ M_1M_2M_3M_4 \end{bmatrix}, \quad M' = \begin{bmatrix} \widehat{M}_0 & O & O & O \\ O & \widehat{M}_1 & O & O \\ O & O & \widehat{M}_1 & O \\ O & O & O & \widehat{M}_0 \end{bmatrix},$$

i. e., M' is the 64×64 block diagonal matrix with blocks $(\widehat{M}_0, \widehat{M}_1, \widehat{M}_1, \widehat{M}_0)$. Finally, the MC layer multiplies the state with M' and is an involution.[3]

[3]Note that M' is also self-transpose, that makes left multiplication of the matrix the same as its right multiplication, i. e., for $x \in \mathbb{F}_2^{64}$, $x \cdot M' = M' \cdot x^T$.

The round function application is interleaved with additions of the same round key K_1 and round constant RC_i.

Overall, for PRINCE$_{core}$, we have the structure shown in Figure 4.1 (top), where

$$\mathtt{R} = \mathtt{SR} \circ \mathtt{MC} \circ \mathtt{SB}, \quad \mathtt{R}' = \mathtt{SB} \circ \mathtt{MC} \circ \mathtt{SB}, \quad \text{and} \quad \mathtt{R}^{-1} = \mathtt{SB}^{-1} \circ \mathtt{MC} \circ \mathtt{SR}^{-1}.$$

Note that here, R is the round function without the key addition.

WHITENING KEYS To follow the FX-structure, PRINCE$_{core}$ is XORed with pre- and post-whitening keys K_0 and K_0', resp., to complete a PRINCE encryption. K_0' is a simple[4] and bijective linear mapping of K_0 where

$$K_0' = (K_0 \ggg 1) \oplus (K_0 \gg 63).$$

[4]Note that implementation of this mapping only need one XOR gate.

ROUND CONSTANTS The round constants for PRINCE are given in Table 4.2 that every round constant in the second half of the encryption process ($i \geq 6$) is the same as its mirror-wise corresponding round constant in the first half of the encryption process XORed with constant α. This means for any $0 \leq i \leq 11$, $RC_i \oplus RC_{11-i} = \alpha$.

DECRYPTION AND α-REFLECTION The choice of round constants causes similarity of PRINCE decryption with its encryption in a way that PRINCE$_{core}$ decryption with key K_1 is the same as PRINCE$_{core}$ encryption with key $K_1 \oplus \alpha$. This property is called α-reflection. Thus PRINCE decryption with pre-, post-whitening, and round keys K_0, K_0', and K_1, resp., is the same as PRINCE encryption with pre-, post-whitening, and round keys K_0', K_0, and $K_1 \oplus \alpha$, resp.

PRINCEv2's Round Operations

For PRINCEv2, the forward round R and backward round $\mathtt{R}^{-1}$ are kept the same with the same operations SB, MC, and SR. The structure for one full encryption is shown in Figure 4.1 (bottom) and in full detail in Figure 4.2.

MIDDLE ROUND While the middle round of PRINCE is a key-less operation, in PRINCEv2, the following middle round is used.

$$R'' = \mathtt{SB}^{-1} \circ \mathtt{A}_\beta \circ \mathtt{A}_{K_1} \circ \mathtt{MC} \circ \mathtt{A}_{K_0} \circ \mathtt{SB}.$$

KEY SCHEDULES Given the 128-bit master key $K = (K_0 \| K_1)$, the i-th round key is defined as

$$K_i := \begin{cases} K_0 & i \in \{0, 2, 4, 6, 8, 10\} \\ K_1 & i \in \{1, 3, 5, 7, 9, 11\} \end{cases},$$

that is alternating between the two parts of the master key K_0 and K_1.

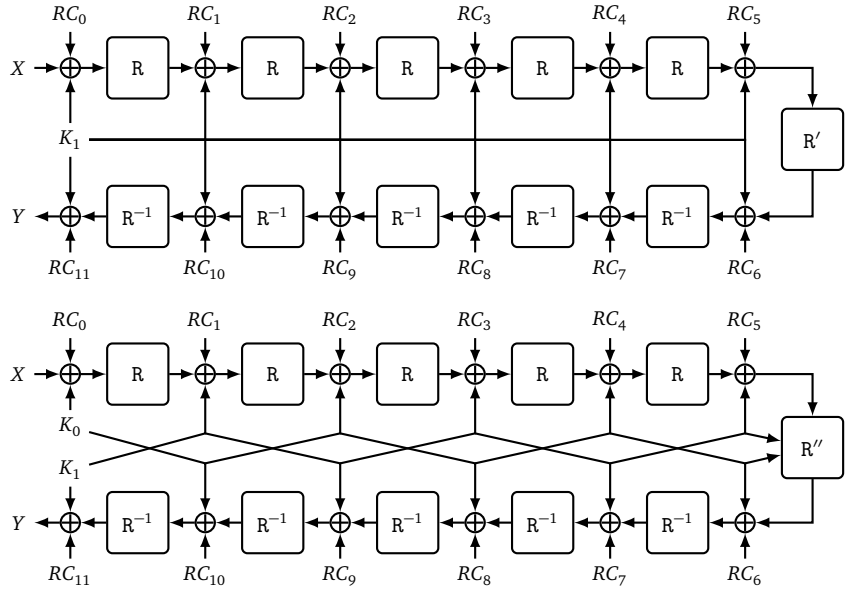


FIGURE 4.1: (Top) PRINCE$_{core}$ Structure; (Bottom) PRINCEV2 Structure. Note that values of RC_7, RC_9 and RC_{11} in PRINCEV2 are different than the ones in PRINCE.

TABLE 4.2: Round Constants of PRINCE and PRINCEV2.

	Constant
	RC_0 0000000000000000
	RC_1 13198a2e03707344
	RC_2 a4093822299f31d0
	RC_3 082efa98ec4e6c89
	RC_4 452821e638d01377
	RC_5 be5466cf34e90c6c
PRINCE	RC_6 7ef84f78fd955cb1 $= RC_5 \oplus \alpha$
PRINCE	RC_7 85840851f1ac43aa $= RC_4 \oplus \alpha$
PRINCE	RC_8 c882d32f25323c54 $= RC_3 \oplus \alpha$
PRINCE	RC_9 64a51195e0e3610d $= RC_2 \oplus \alpha$
PRINCE	RC_{10} d3b5a399ca0c2399 $= RC_1 \oplus \alpha$
PRINCE	RC_{11} c0ac29b7c97c50dd $= RC_0 \oplus \alpha$
PRINCEv2	RC_6 7ef84f78fd955cb1 $= RC_5 \oplus \alpha$
PRINCEv2	RC_7 7aacf4538d971a60 $= RC_4 \oplus \beta$
PRINCEv2	RC_8 c882d32f25323c54 $= RC_3 \oplus \alpha$
PRINCEv2	RC_9 9b8ded979cd838c7 $= RC_2 \oplus \beta$
PRINCEv2	RC_{10} d3b5a399ca0c2399 $= RC_1 \oplus \alpha$
PRINCEv2	RC_{11} 3f84d5b5b5470917 $= RC_0 \oplus \beta$
	$\alpha =$ c0ac29b7c97c50dd
	$\beta =$ 3f84d5b5b5470917

ROUND CONSTANTS The round constants are derived as for PRINCE, but instead of adding the same α for every round constant in the second half of the encryption process ($i \geq 6$), it alternates between adding α and β as defined in Table 4.2.

ENCRYPTION AND DECRYPTION Even though PRINCEV2 does not fulfill the α-reflection property anymore, the choice of round keys and constants allows to implement both encryption and decryption with only a small area and delay overhead. This is shown in Figure 4.3. In this figure the extra control signal **dec** switches between encryption and decryption. In particular, the `Swap` function is defined as

$$\texttt{Swap}(K_0, K_1, \text{dec}) = \begin{cases} K_0, K_1 & \text{if dec} = 0 \\ K_1 \oplus \beta, K_0 \oplus \alpha & \text{if dec} = 1 \end{cases}.$$

4.1.2 *Rationale*

The main objectives almost immediately resulted in clear design rationales. The first design rationale was to leave the round function, and not less important the number of rounds, (almost) unchanged, and only change the key-scheduling. Here, the round-constants are thought of as part of the key-scheduling, even if it was not presented this way in the original PRINCE paper. This rationale both drastically simplified the design process and made it much more challenging. The simplification is due to the choices being narrowed down to a key-scheduling that has to be picked. More challenging, as the analysis had to be much more precise and careful, the security margin would decrease. It is important to highlight that the security margin is relative to the claimed security level, not in absolute terms.

The only compromise from the goal of leaving the round function unchanged is the middle round. Some attacks that have been developed since the publication of PRINCE take explicit advantage of the symmetry and the

key-less middle rounds (See Sections 6.2.6 and 6.2.7). To make those attacks, particularly MitM attacks and accelerated exhaustive search procedures, less of a concern, it seems to be a good trade-off to spend two (actually one as will be explained in Section 4.1.3) additional XOR on the critical path.

For the key-scheduling, we were again highly restricted by the requirement of limiting the overhead of implementing decryption on top of encryption. This implies, as it did in PRINCE, that a complicated key-update is not a proper choice but a simple, up to the constants, periodic key-scheduling is the best. Hence, we opted for one of the simplest possible options: iterated round-keys.

Originally, in PRINCE, the round keys derived from the 128-bit key $(K_0 \| K_1)$ correspond to

$$K_0 \oplus K_1, K_1, K_1, K_1, K_1, K_1, K_1 \oplus \alpha, K_1 \oplus \alpha, K_1 \oplus \alpha, K_1 \oplus \alpha, K_1 \oplus \alpha, K_0' \oplus K_1 \oplus \alpha,$$

where K_0' results from a simple and efficient bijective linear mapping from K_0, and α is a constant value given in Table 4.2.

In particular, the value of K_0 is used only in the whitening keys, limiting the security generically. The α-reflection property follows as replacing K_1 with $K_1 \oplus \alpha$ reverts the order of the round keys (except for the outer whitening keys).

In PRINCEv2, using a master key $(K_0 \| K_1)$, the round keys are chosen as

$$K_0, K_1, K_0, K_1, K_0, K_1, K_0, K_1 \oplus \beta, K_0 \oplus \alpha, K_1 \oplus \beta, K_0 \oplus \alpha, K_1 \oplus \beta, K_0 \oplus \alpha, K_1 \oplus \beta,$$

where α and β are constant values given in Table 4.2. The constants are chosen as digits of $\pi = 3.1415\ldots$, as they were done in PRINCE. The new constant β is simply the next in line looking at the binary digits of π, that the following SAGEMATH code to is used to generate the round constants:

```
a = RealField(prec=2000)(pi)-3
for i in range(1, 9):
b = (floor(a*2^(64*i)) + 2^64) % 2^64
print("0x%016x" % (b))
```

Then the output of the code is:

```
0x243f6a8885a308d3
0x13198a2e03707344
0xa4093822299f31d0
0x082efa98ec4e6c89
0x452821e638d01377
0xbe5466cf34e90c6c
0xc0ac29b7c97c50dd
0x3f84d5b5b5470917
```

Here, replacing K_0 by $K_1 \oplus \beta$ and K_1 by $K_0 \oplus \alpha$ does ensure that the first rounds of the encryption circuit perform decryption as required. However, when reaching the middle round, an additional modification is required to ensure the second half works as well. As shown in Section 4.1.3, this does not significantly harm performance.

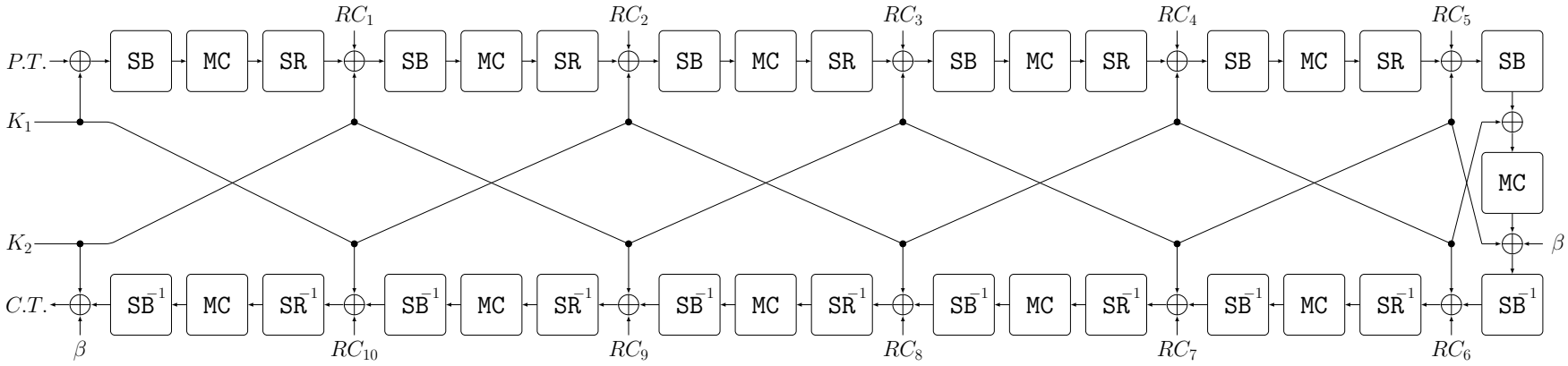


FIGURE 4.2: PRINCEV2 Structure for Encryption.

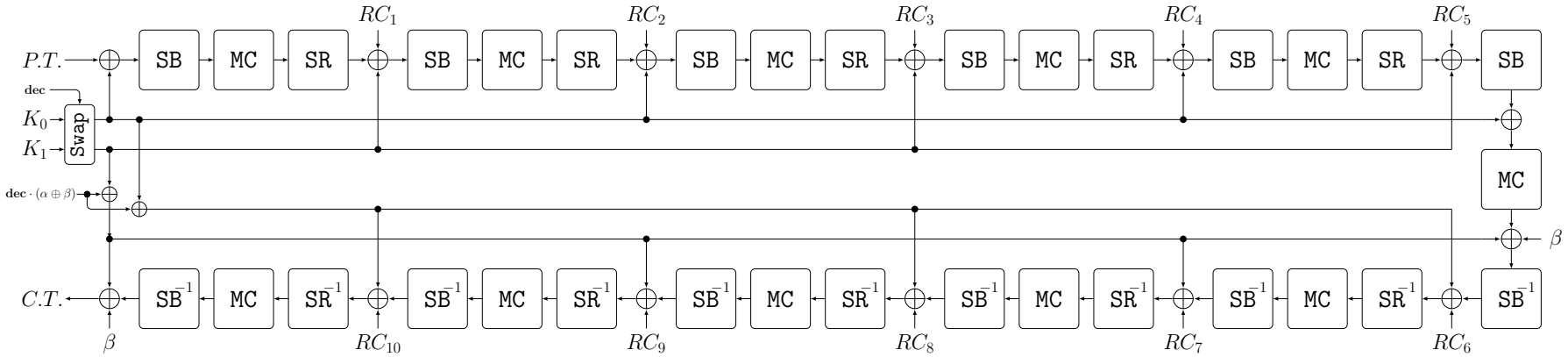


FIGURE 4.3: PRINCEV2 Structure for Encryption and Decryption.

The only case where this would not be necessary is when α equals β. However, this would introduce a set of weak-keys. Namely, if $K_1 \oplus K_2 = \alpha$, then encryption would equal decryption; that is, the whole cipher would be an involution.

Finally, let us explicitly state the claim we want our design to be tested against:

Security Claim 3 (PRINCEv2)**.** *We claim that there is no attack against* PRINCE*v2 with data complexity below* 2^{50} *bytes (*2^{47} *chosen plaintext-ciphertext pairs) and time complexity below* 2^{112}*. We do not claim any security in the related-key setting, and related-keys have to be avoided at the protocol level.*

This claim is backed up by the extensive security analysis that is provided in Section 6.2. It is interesting to see how the advance in state of the art has made the analysis more precise (e. g., for Boomerang attacks using connectivity tables and for integral attacks using the division property) and more straightforward (using mainly MILP-based tools). Those improvements are an important tool to enable a cipher design with a very tight security claim: For a cipher optimized for low-latency, a large security margin is nothing but wasted performance.

4.1.3 *Hardware Implementation*

The primary objective of PRINCE and PRINCEv2 is to offer low-latency single-cycle encryption and decryption. This objective requires a short critical path in round-unrolled non-pipelined hardware implementations. This means the ciphers aim for a small logic depth in circuit representation. Furthermore, adding decryption functionality to an encryption circuit should induce minimal area/latency overhead. PRINCE achieves this goal in part due to the so-called α-reflection property. This property has been imitated by several other low-latency constructions (e. g., MANTIS [Bei+16] and QARMA [Ava17]) and mandates that decryption with one key corresponds to encryption with a related key. Due to the modified key schedule, PRINCEv2 does not fulfill the α-reflection property of PRINCE. Yet, it implements a modified version that keeps the decryption overhead in hardware reasonably small.

A secondary design goal is keeping its unrolled implementation cost-efficient, including a small hardware footprint (little occupied chip area) and low energy consumption. In fact, the costs should be lower compared to unrolled implementations of other lightweight block ciphers. According to the original proposal, unrolled PRINCE with decryption and encryption capability can be clocked at frequencies up to 212.8 MHz when synthesized in NanGate 45 nm, an open-source standard cell library, and requires as little as 8260 GE of area [Bor+12]. PRINCEv2 aims to achieve similar performance figures while providing stronger security guarantees overall. Staying close to the initial design of PRINCE enables us to recycle and build upon the extensive security analysis it has already received. Furthermore, it allows constructing circuits that can perform encryption and decryption in both the new PRINCEv2 *and* original PRINCE at low overhead. This provides needed

TABLE 4.3: Area (GE), Latency (ns) and Energy (pJ) Characteristics of Unrolled PRINCE and PRINCEV2 with Minimum Latency Constraint.

Technology	Mode	Cipher	Area	Latency	Energy
90 nm LP*	Enc	PRINCE	16244.25	**4.101177**	1.993172
		PRINCEV2	17661.25	**4.047311**	2.230068
	Enc+Dec	PRINCE	17808.00	**4.106262**	2.213275
		PRINCEV2	18888.75	**4.151113**	2.424250
65 nm LP*	Enc	PRINCE	19877.75	**2.866749**	1.602513
		PRINCEV2	18798.25	**2.944367**	1.492794
	Enc+Dec	PRINCE	19966.00	**2.946442**	1.594025
		PRINCEV2	21171.25	**2.930153**	1.696559
40 nm LP*	Enc	PRINCE	17177.00	**2.521302**	0.617719
		PRINCEV2	16556.50	**2.509131**	0.592155
	Enc+Dec	PRINCE	17377.50	**2.541220**	0.630223
		PRINCEV2	17799.50	**2.583466**	0.648450
28 nm HPC**	Enc	PRINCE	38145.33	**1.108886**	1.258586
		PRINCEV2	33470.33	**1.103273**	1.108789
	Enc+Dec	PRINCE	35297.67	**1.119593**	1.181171
		PRINCEV2	38962.33	**1.148693**	1.299172

* LP = Low Power ** HPC = High Performance Computing

legacy support and backward compatibility for various of applications and environments where PRINCE is already deployed.

In the following, the novel PRINCEV2 design is compared to the original PRINCE concerning the minimum latency and minimum area achieved by unrolled implementations. Note that the author of the thesis was not contributed to any of the following hardware implementations, and these results are borrowed from PRINCEV2 proposal paper [Bož+20].

Since gate count and delay numbers depend on the particular technology used, synthesis results from 4 different commercial standard cell libraries of feature sizes between 90 nm and 28 nm are provided in the PRINCEV2 proposal paper. This redundancy minimizes the influence of a single technology on the comparison's interpretation.

The results for minimum latency constraint are given in Table 4.3. Note that circuits that can only encrypt (Enc) and circuits that can both encrypt and decrypt (Enc+Dec) are distinguished.

Several interesting observations can be made. Firstly, all four distinct circuits perform decidedly similar in terms of minimum latency. The difference falls in the range of single-digit picoseconds in several cases. Interestingly, the encryption-only version of PRINCEV2 outperforms PRINCE in terms of minimum latency in three of four technologies. This may be counter-intuitive, as PRINCEV2 adds two key additions to the middle round. However, as shown in Figure 4.4, those two key additions can be merged into a single one regarding the critical path by calculating and adding $\texttt{MC}(K_1)$ in parallel.

This optimization not only improves the minimum latency but also saves area.[5] One may expect the synthesis tool to perform such an optimization implicitly by itself, as two key additions and a MC operation in the middle essentially result in a sequence of four consecutive XORs per bit. Yet, the results suggest that it is indeed required to perform the optimization algorithmically in the RTL code. Additionally, it has to be noted that the

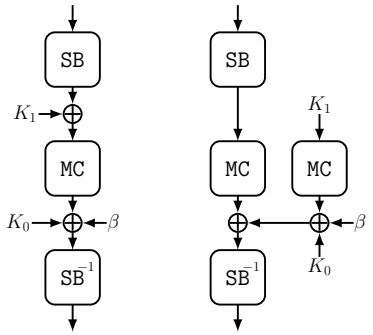


FIGURE 4.4: Simple Latency Optimization Strategy in the Middle Round of PRINCEV2. It removes one key addition from the critical path.

[5] Area is saved by this optimization since slower cells with a lower drive strength can be selected for the parallel calculations. Those cells have a smaller area footprint.

original PRINCE design applies two key additions (whitening and round key) to the input before the first Sbox stage. PRINCEv2, on the other hand, applies only one. Hence, the difference between the latency of PRINCE and PRINCEv2 in encryption-only mode comes down to whether the synthesizer implements the additional key XOR at the input more efficiently or the one in the middle round. More often than not, the middle round key addition is more efficient latency-wise since the synthesizer has more freedom to move that XOR stage around (e. g., to the input, output or intermediate signals of the MC operation), while the two key XORs at the input have a fixed location and cannot be optimized beyond instantiating a three-input XOR per bit, since all inputs to the operation (key and plaintext) arrive at approximately the same time. Original PRINCE also requires an additional key XOR at the output. Yet, since the last round's output arrives much later than the keys, the key parts will be added to each other beforehand, and only one of the XOR stages affects the critical path.

Regarding the cipher variants with decryption capability, the situation is slightly different compared to the encryption-only versions. The more complex key-multiplexing in PRINCEv2 required to choose between encryption and decryption, as apparent in Figure 4.3, induces additional delay. In the worst case, this results in an overhead of 2.6%, but on average, the overhead is about 1.2%. Table 4.3 also reports area and energy numbers for the highly constrained circuits. The energy values correspond to the average energy consumed by one evaluation of the unrolled circuits at maximum clock frequency (corresponding to minimum latency). While PRINCEv2 is often more area and energy-efficient than PRINCE in encryption-only mode, PRINCEv2 with decryption capability consistently requires the largest area and consumes the most energy. Yet, the margins are still very thin. In summary, PRINCE's most important property and main selling point, namely high-speed single-cycle encryption, is well preserved by PRINCEv2. For scenarios that require no decryption but only encryption, it may even be slightly improved.

As a second step, the minimum area that can be achieved by the unrolled circuits is evaluated. In this regard, the same synthesis scripts as before is executed, but without timing constraints. The results are reported in Table 4.4.

In contrast to the latency results, a consistent overhead for a minimum area can be observed for the PRINCEv2 circuits. This is expected due to the additional operations in the middle round and the more complex key-multiplexing to decide between encryption and decryption.

Yet, the average overhead is less than 1.2% for the encryption-only and less than 3.5% for the full versions. This is a relatively small price to pay for the additional security PRINCEv2 provides.

When comparing the low-latency and low-area implementations in Tables 4.3 and 4.4 resp., it can be seen that area increases 2 to 4 times from low-area to low-latency constraint. Latency scales down 3 to 4 times. Energy consumption increases between 4 and 10 times due to a dependency on both factors (caused by leakage currents). These metrics should be carefully considered when choosing operating frequency and target technology for a given application.

TABLE 4.4: Area (GE), Latency (ns) and Energy (pJ) Characteristics of Unrolled PRINCE and PRINCEV2 with Minimum Area Constraint.

Technology	Mode	Cipher	Area	Latency	Energy
90 nm LP*	Enc	PRINCE	**7937.50**	12.859908	0.569694
		PRINCEV2	**8111.25**	12.856450	0.574683
	Enc+Dec	PRINCE	**8183.00**	14.015245	0.616671
		PRINCEV2	**8440.75**	15.513536	0.628298
65 nm LP*	Enc	PRINCE	**8316.00**	11.434771	0.433378
		PRINCEV2	**8385.25**	11.504968	0.430286
	Enc+Dec	PRINCE	**8547.75**	12.349355	0.440872
		PRINCEV2	**8792.75**	13.376949	0.456154
40 nm LP*	Enc	PRINCE	**8563.75**	10.144847	0.212027
		PRINCEV2	**8608.50**	10.063908	0.207317
	Enc+Dec	PRINCE	**8780.00**	10.886960	0.217739
		PRINCEV2	**9039.75**	11.798657	0.226534
28 nm HPC**	Enc	PRINCE	**8197.00**	3.599936	0.127798
		PRINCEV2	**8292.00**	3.682593	0.127786
	Enc+Dec	PRINCE	**8426.33**	4.260999	0.131239
		PRINCEV2	**8844.67**	4.323993	0.134909

* LP = Low Power ** HPC = High Performance Computing

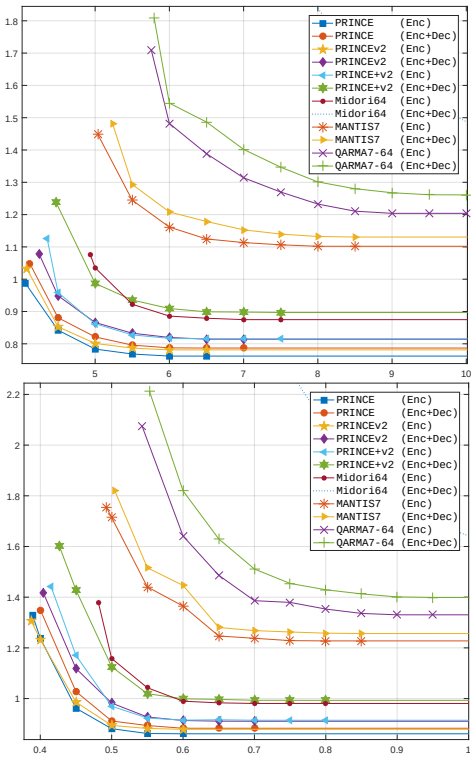


FIGURE 4.5: Comparison of Unrolled Implementation of Block Ciphers in NanGate 45 nm (top) and 15 nm (bottom) Open Cell Libraries. X axis is the latency in nanoseconds and Y axis is the area in 10^4 GE.

Finally, PRINCE and PRINCEV2 are compared to other lightweight block ciphers proposed in the literature. Only a few cryptographic primitives have made low-latency a primary design objective. To the best of our knowledge, none of those who have shared all design goals and security claims with PRINCE or PRINCEV2. Hence, the following comparison involves ciphers with different security and performance claims and is only supposed to put their hardware efficiency in relation to each other, without concluding the superiority of one or the other. In particular, PRINCE and PRINCEV2 is compared to the low-latency tweakable block ciphers MANTIS [Bei+16] and QARMA [Ava17]. Yet, since both of those constructions are tweakable, unlike all PRINCE versions, they can not easily achieve the same latency and area as PRINCE and PRINCEV2. MIDORI [Ban+15] is also included in the comparison, as the authors have partially aimed for a low logic depth as well. However, MIDORI primarily targets energy efficiency in round-based implementations and has no claim to provide low-latency in unrolled representation. Additionally, a combination of PRINCE and PRINCEV2 is developed, called PRINCE+V2. This combined cipher offers a control signal to select whether the input should be processed according to the PRINCE or the PRINCEV2 specification. In environments where PRINCE is already deployed, this can be useful for backward compatibility and legacy support. All 6 ciphers are analyzed in two modes each (Enc and Enc+Dec) in NanGate 45 nm and 15 nm Open Cell Libraries and their area-vs-latency trade-off are evaluated. The results are depicted in Figure 4.5 for both NanGate 45 nm and 15 nm technologies.

The results in these two libraries demonstrate that PRINCE and PRINCEV2 are the most suitable choices for high-speed encryption, as long as a tweak input is not required. All PRINCE and PRINCEV2 variants outperform the other ciphers both in terms of minimum latency and minimum area. PRINCEV2 in encryption-only mode is roughly 20 percent faster than MIDORI and

MANTIS, and 40 percent faster than QARMA. At the same time, its minimum area is about 15 percent smaller than MIDORI, 30 percent smaller than MANTIS and 40 percent smaller than QARMA. The results are similar when comparing both encryption and decryption implementations, except for MIDORI being significantly larger and slower. This outcome is unsurprising since all compared ciphers except MIDORI are reflection ciphers, i. e., they use a variant of the α-reflection property introduced by PRINCE. Please note that for this reason the full version of MIDORI, including decryption, is barely visible in Figure 4.5 as it simply does not fit in the frame due to its much higher latency and area caused by the extra multiplexers required in each round. Also note that particular instances of MANTIS and QARMA are chosen for the comparison as they are supposed to offer a similar security level as PRINCE and PRINCEv2, while being tweakable.

In the original proposal, the maximum achievable frequency of unrolled PRINCE in NanGate 45 nm was given as 212.8 MHz [Bor+12]. Here, the unrolled PRINCE can be clocked at 242.8 MHz for the full variant and 246.3 MHz for the encryption-only version in the same technology, which corresponds to a 12.4% or 15.7% higher performance respectively. The minimum area was given as 8260 GE in the original proposal, while implementations here, are as small as 7868.67 GE for the full variant and 7620.00 GE for the encryption-only version. That corresponds to a 5.0% or 8.4% higher area efficiency resp. As a conclusion, the base-implementation used for the construction of PRINCEv2 and PRINCE+v2 (and partially for MANTIS) is well optimized.

We thus believe that PRINCEv2 is a major improvement over PRINCE and we expect it to be widely adopted in the near future. Moreover, this work shows that one can improve the security level of some lightweight primitives with minimal downsides. An open question is thus to see if similar improvements could be made for other microcontroller-targeted ciphers such as MIDORI, MANTIS and QARMA, which could lead to interesting future work.

4.2 LOW-LATENCY S-BOXES

Studying the properties of all n-bit Boolean functions is not an easy task when $n > 5$. The applied vectorial Boolean functions in cryptography with $n > 5$ are usually found by their mathematical properties, and their implementations are considered in second priority. For instance, the S-box used in the AES is the best known 8-bit permutation (up to affine equivalence), but its implementation is quite challenging, especially when we want to implement the cipher together with a countermeasure.

One of the hardware properties of (vectorial) Boolean functions is their latency. Finding the minimum possible latency of implementing a Boolean function with a synthesizer is not possible if the number of inputs is high and if we only use the truth table of the function.

As explained in Section 2.4.1, the gate depth complexity defined in Definition 38 is usually considered to mathematically model the minimum latency of all possible implementations of a (vectorial) Boolean function. In this section, we first modify this metric to have a model closer to reality,

which we call the *latency complexity* of the Boolean function. We present a unique structure (circuit) to implement any Boolean function with a latency complexity of d. We also define an equivalence relation to partition the set of n-bit Boolean functions, $\mathcal{B}_n$, such that the latency complexity of Boolean functions is invariant under this equivalence relation.

Next, we discuss the computation complexity of the search for finding the latency complexity of a given Boolean function and present some techniques to make it faster. Using these techniques, we find all the equivalent classes of latency complexity for up to $n = 5$. Besides, we present an algorithm to find all the Boolean functions with a low-latency complexity for $n \in \{6, 7, 8\}$.

Finally, using these classified Boolean functions, we look for the existence of bijective n-bit S-boxes with a given latency complexity and study their cryptographic properties. To do this, we introduce an algorithm to reduce the computation complexity of building S-boxes.

4.2.1 *Latency Complexity of a Boolean Function*

Due to the different delay of the logic gates allowed by the applied ASIC library for implementation, it is not realistic to consider the gate depth complexity as a metric for the latency of the implementation. For instance, in almost all the libraries, the latency of an XOR or an XNOR gate is about twice the latency for other gates with a fan-in number[6] of 2. Another example is the difference in the latency of the gates with a different number of fan-ins; the latency of the gates with a higher number of fan-ins is larger than the latency for similar gates but with a fewer number of fan-ins. On the other hand, except for the INV (inverter) gate that has a fan-in number of one, the 2-bit NAND gate, and then the 2-bit NOR gate always has the minimum latency in almost all the ASIC libraries. To make a view on latency of different gates, we list the latency of all the logic gates in NanGate 15 nm Open Cell Library (OCL) with constraint in the minimum latency in Table 4.5.

For this reason, to have a more accurate metric for the latency, we use the gate depth complexity by restricting ourselves to use only INV and 2-bit NAND and NOR gates.[7] But since the INV gate has lower latency than the two other gates, and more important, because of the reason that is explained later in Proposition 71, we do not consider the INV gates in the gate count of the implementation. Thereby, we define the latency complexity of a (vectorial) Boolean function as in the following definition.

Definition 69 (Latency Complexity)**.** It is the minimum length of the longest path (concerning the number of only NAND and NOR gates) from an input to an output for computing the function while the set of allowed gates to use is $\mathcal{G} = \{\texttt{INV}, \texttt{NAND}, \texttt{NOR}\}$.

In the equations, for simplicity, we use the signs $\wedge, \vee, \overline{\wedge}$, and $\overline{\vee}$ to denote the operation of the AND, OR, NAND, and NOR gates, resp. Besides, we use $\neg y$ to denote the inverted value for the output of a combination y and use $\overline{x}$ to denote the inverted value of the input x. In the following, we explain an example to show how to count the gate depth of implementation when we are looking for its latency complexity.

[6] By fan-in or fan-out numbers the gate, we mean the number of inputs and the outputs to the gate, resp.

TABLE 4.5: The Latency of Logic Gates in NanGate 15 nm Open Cell Library (OCL) for Typical Operating Conditions.

Fan-In	Gate	Latency [ps]
1	INV	1.580082
2	AND2	3.579786
	NAND2	2.030621
	NOR2	2.554366
	OR2	3.643867
	XNOR2	6.788322
	XOR2	5.268465
3	AND3	5.496015
	AOI21	3.394032
	MUX2	6.133133
	NAND3	2.360978
	NOR3	3.787567
	OAI21	2.830147
	OR3	5.862194
4	AND4	7.125210
	AOI22	4.070343
	NAND4	4.659015
	NOR4	5.250172
	OAI22	3.775570
	OR4	7.682688

[7] From now on, for simplicity, we do not mention the gate's number of fan-ins, unless it is more than 2.

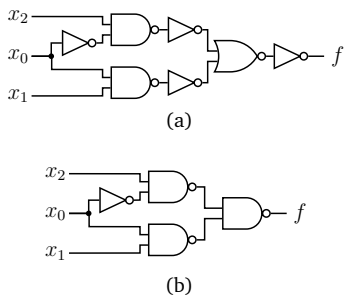


FIGURE 4.6: Implementation of the Function Given in Example 70.

Example 70. The circuit shown in Figure 4.6a is an instance for implementing the function $f(x_0, x_1, x_2) = (x_0 \wedge x_1) \vee (\overline{x}_0 \wedge x_2)$ using the gates in $\mathcal{G} = \{\texttt{INV}, \texttt{NAND}, \texttt{NOR}\}$. Note that f is a balanced function with the ANF representation of $x_0x_1 \oplus x_0x_2 \oplus x_2$. Besides, it represents the functionality of a multiplexer that x_0 acts as the selector to choose one of x_1 or x_2 variables.

The circuit implies that $y = \neg\big(\neg(x_0 \overline{\wedge} x_1) \,\overline{\vee}\, \neg(\overline{x}_0 \overline{\wedge} x_2)\big)$ and the length of the paths from each input to the output (with only counting the NAND and the NOR gates) is 2. Using the equation $\neg(\overline{y}_0 \,\overline{\vee}\, \overline{y}_1) = y_0 \,\overline{\wedge}\, y_1$, one can simplify the implementation in Figure 4.6a to the implementation shown in Figure 4.6b.

Even though the latency complexity of implementing f is 2, to determine it, we need to consider the length of the longest path in all possible implementations of f.

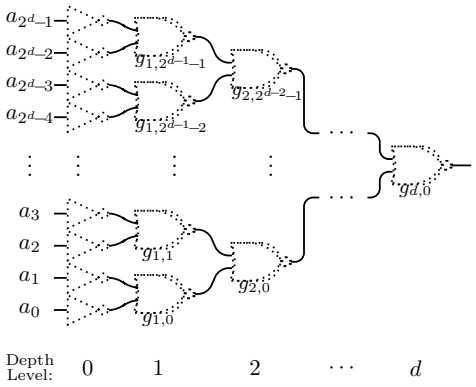


FIGURE 4.7: General Structure for Implementing a Function with a Latency Complexity of d.

Proposition 71. *Any n-bit Boolean function $f(x_0, \ldots, x_{n-1})$ with the latency complexity of d can be realized in hardware by using the structure shown in Figure 4.7. In this structure, each of $g_{i,j}$ gates with $0 < i \leq d$ and $j \in \mathbb{Z}_{2^{d-i}}$ are either a NAND or a NOR gate, and the inputs to the gates in the first level is either a_i or its inverted value, $\overline{a}_i$, while each a_i with $i \in \mathbb{Z}_{2^d}$ is chosen from the set $\{x_0, \ldots, x_{n-1}\}$.*

Proof. To prove the proposition, we need to show that any function with the latency complexity of d can be implemented by 1) using only INV gates in the beginning, i. e., the depth level 0, and 2) using the output of each gate in the input of only one gate in the next depth level. To show the first part, we use the below equations:

$$\neg(y_0 \,\overline{\wedge}\, y_1) = \overline{y}_0 \,\overline{\vee}\, \overline{y}_1\,, \qquad \neg(y_0 \,\overline{\vee}\, y_1) = \overline{y}_0 \,\overline{\wedge}\, \overline{y}_1\,.$$

These equations imply that if there is an INV gate in the output of a NAND or NOR gate, it can be replaced from the output to both of the inputs by changing the gate type to NOR or NAND gate, resp. Thereby, in an implementation of a function, if there is an INV gate in the output of NAND or NOR gate, we can replace it with two INV gates in the depth level of $i-1$ and by changing the NAND/NOR gate's type. Thus, starting from depth level d, we can bring all the INV gates to the previous depth level and update the implementation. By updating the implementation, we mean changing the type of the corresponding NAND or NOR gate and removing (if there exist) two repeated INV gates in the depth level $i-1$.

To show the second part of the proof, assume that the output of a NAND or a NOR gate is used in the input of other gates more than once. This can be eliminated by repeating the implementation of the corresponding sub-circuit of this output in such a way that the output of each repetition is used only once. Besides, assume that the output of a gate in the depth level i is used in the depth level of i' with $i' > i+1$. This also can be eliminated by using similar equations to $\neg y \,\overline{\wedge}\, \neg y = y$, i. e., repeating the inverted circuit for y. Therefore, we make sure that the output of a gate in the depth level i is used only once in an input of a gate in the depth level of $i+1$. □

Note that to prove the second part of the above proof, we may repeat the implementation of a sub-circuit several times, and area-wise this is not efficient. Even though the structure of Proposition 71 makes it possible to easily represent the implementation of a Boolean function with the latency complexity of d and therefore (as it comes later) to study the latency complexity of all the functions in $\mathcal{B}_n$. For comparison, in the previous work by Stoffelen [Sto16], it is considered that the output of gates can be used in the input of several gates and in all the next depth levels. This causes to have more variables in the SAT equations and thus more complexity for solving it.

[Sto16] Stoffelen "Optimizing S-Box Implementations for Several Criteria Using SAT Solvers"

As shown in Figure 4.7, for a low-latency implementation of a function, we only need to use the INV gates once and at the beginning (i. e., depth level of 0). Moreover, in the unrolled implementation, it is necessary to use several *buffer* gates for the input variables of each combinatorial circuit to amplify the voltage of the wires, but if the input variable needs to go through an INV gate, there will be no need to use such a buffer. That means, in the depth level 0, each input variable a_i will go through a buffer gate (with output of a_i) or through an INV gate (with output of $\neg a_i$). This is why we do not count the INV gates in the gate count of the longest path.

In the rest of this section, for implementing the Boolean function f, we focus on the structure of Proposition 71. By mapping each $g_{i,j}$ gate (with $0 < i \leq d$ and $j \in \mathbb{Z}_{2^{d-i}}$) to a binary value, we use $G \in \mathbb{F}_2^{2^d-1}$ to denote the type of gates, i. e., $G := (g_{0,0}, \ldots, g_{0,2^{d-1}-1}, g_{1,0}, \ldots, g_{1,2^{d-2}-1}, \ldots, g_{d,0})$. We map a NAND gate to 0 and a NOR gate to 1. By $\alpha \in \mathbb{F}_2^{2^d}$, we denote that if a_i with $i \in \mathbb{Z}_{2^d}$ goes through an INV gate. More precisely, if $\alpha[i] = 1$, then $\overline{a}_i$ is used as the input of the gate in depth level 1, otherwise a_i itself. We use $\pi \in \mathbb{Z}_n^{2^d}$ to denote the choice of x_j variables by a_i variables, i. e., $a_i = x_{\pi[i]}$, and by $P : \mathbb{F}_2^n \mapsto \mathbb{F}_2^{2^d}$, we denote the corresponding mapping applied by π, i. e., $x \mapsto P(x)$ with $\forall i \in \mathbb{Z}_{2^d}$, $P(x)[i] = x[\pi(i)]$.

Moreover, we use $\mathcal{I}_{i,j}$ with $0 < i \leq d$ and $j \in \mathbb{Z}_{2^{d-i}}$ to denote the corresponding sub-circuit from the inputs $x_0, \ldots, x_{n-1}$ to output of the gate $g_{i,j}$. Therefore, each $\mathcal{I}_{i+1,j}$ with $0 < i < d$ and $j \in \mathbb{Z}_{2^{d-i}}$ can be represented by $(\mathcal{I}_{i,2\cdot j}, \mathcal{I}_{i,2\cdot j+1}, g_{i,j})$ tuple. Thus, $\mathcal{I}_{d,0}$ is the implementation of f. Besides, each of $\mathcal{I}_{i,j}$ can be represented by the corresponding $(G_{i,j}, \alpha_{i,j}, \pi_{i,j})$ tuple that $G_{i,j} \in \mathbb{F}_2^{2^i-1}$, $\alpha_{i,j} \in \mathbb{F}_2^{2^i}$, and $\pi_{i,j} \in \mathbb{Z}_n^{2^i}$.

Determining the gate depth complexity of a given Boolean function is an NP-hard problem [BMP08]. Since the latency complexity is equivalent to the gate depth complexity, determining the gate depth complexity of a given Boolean function is also an NP-hard problem. By knowing that the latency complexity of an n-bit Boolean function is not smaller than d, one can try all the possibilities for the structure of Proposition 71. To do so, we need to go through all 2^{2^d-1} choices for G, all 2^{2^d} choices for α, and all n^{2^d} choices for π, which ends up with a total computational complexity of about $2^{2^d \cdot (2+\log_2(n))}$. It is clear that if $d > 5$, it is not possible to do this computation.

[BMP08] Boyar, Matthews, and Peralta, "On the Shortest Linear Straight-Line Program for Computing Linear Forms"

In [Sto16], Stoffelen presented a SAT-solver-based technique to optimize the implementations of Boolean functions with $n \leq 5$ for different criteria such as the gate depth complexity. While this technique gives a solution for low-latency implementation for small Boolean functions, it is not able to find all the Boolean functions with low-latency complexity for $n > 5$.

In the next section, we present techniques to speed up the search algo-

rithm to compute the latency complexity of a given Boolean function and also present an efficient algorithm to find the Boolean functions with a given latency complexity.

4.2.2 *The Boolean Functions with Low-Latency Complexity*

In this section, we first introduce an equivalent relation within $\mathcal{B}_n$ such that the latency complexity stays invariant over this relation. Next, we explain some reductions in the search for computing the latency complexity of a given Boolean function and also on finding all Boolean functions with a given latency complexity. Then, we determine the latency complexity of all n-bit Boolean functions with $n < 6$. Finally, we present an algorithm to find all n-bit Boolean functions with a given latency complexity for $n \geq 6$.

Extended Bit Permutation Equivalence

It is clear that a bit permutation or a constant addition in the input or the output of a Boolean function does not change the latency complexity. These properties are explained in detail in the following proposition.

Proposition 72. *Let f_1 and f_2 be two n-bit Boolean functions such that $f_2(x) = f_1\big(P'(x) \oplus \alpha'\big) \oplus c$ for any $x \in \mathbb{F}_2^n$ with P' being a bit permutation function with corresponding permutation of π', and $\alpha' \in \mathbb{F}_2^n$, $c \in \mathbb{F}_2$ being constant values. Then the latency complexity of f_1 and f_2 are the same. Moreover, if (G, α, π) is the instantiation of implementing f_1 in the structure of Figure 4.7, with P being the mapping applied by π, the corresponding implementation for f_2 can be realized by $(G, \alpha \oplus P(\alpha'), \pi \circ \pi')$ if $c = 0$; and if $c = 1$ then it can be implement by $(\overline{G}, \overline{\alpha} \oplus P(\alpha'), \pi \circ \pi')$ while $\overline{G}$ and $\overline{\alpha}$ denote that complement value of G and α, resp.*

Proof. A bit permutation can be realized in hardware by replacing the wires, and a constant addition can be realized by adding INV gates. Therefore it is clear that we can modify the implementation for f_1 and use it for f_2 without changing the length of the longest path (with only counting the NAND and NOR gates). Since this is true for any implementation of f_1, then the latency complexity of f_2 is the same as the latency complexity of f_1.

To prove the second part of the proposition, we first consider that $c = 0$. The input of the structure in case of f_1 for a given $x \in \mathbb{F}_2^n$ is $P(x)$, while for f_2 needs to be $P(P'(x) \oplus \alpha')$ that P' is the corresponding function for bit permutation of π'. Since P is a linear function, then $P(P'(x) \oplus \alpha') = P \circ P'(x) \oplus P(\alpha')$. The function $P \circ P'$ can be realized by choice function of $\pi \circ \pi'$. The next part, $P(\alpha')$, also can be combined with the INV gates in the depth level of 0. To do this combination, instead of using INV gates with α representation, we use INV gates with representation $\alpha \oplus P'(\alpha')$.

In the case of $c = 1$, we implement $\neg f_2 = f_1\big(P'(x) \oplus \alpha'\big)$ as it is explained with the case for $c = 0$ and then insert an extra INV gate in the output of $\neg f_2$ to realize implementation of f_2. It is possible to replace the INV gate in depth level of d with two INV gates in the depth level of $d-1$ by changing the gate type, i. e., $g_{d,0}$. Repeating this for d times, we end up with 2^d extra INV gates in depth level 0 and changing all the gate types. This means we changed G to $\overline{G}$ and $\alpha \oplus P'(\alpha')$ to $\overline{\alpha \oplus P'(\alpha')} = \overline{\alpha} \oplus P'(\alpha')$. □

Definition 73 (Extended Bit Permutation Equivalence)**.** Two n to m-bit Boolean functions F and G are called *extended bit permutation equivalent* [LP07] if there exist a bit permutation of n bits P_i, a bit permutation of m bits P_o, $\alpha \in \mathbb{F}_2^n$, and $\beta \in \mathbb{F}_2^m$ in such a way that $G = P_o \circ G \circ P_i(x \oplus \alpha) \oplus \beta$.

[LP07] Leander and Poschmann, "On the Classification of 4 Bit S-Boxes"

Proposition 72 implies that the extended bit permutation equivalent functions behave equally with respect to the latency complexity. Therefore, instead of studying whole $\mathcal{B}_n$, it is enough to study the representative Boolean functions for each equivalence class.

Table 4.6 shows the number of n-bit Boolean functions up to extended bit permutation equivalence for $n \leq 5$, while N_1 denotes the number of all Boolean functions up to the equivalence, N_2 denotes the number of full-dependent Boolean functions, N_3 denotes the number of balanced Boolean functions, and N_4 denotes the number of full-dependent and balanced Boolean functions up to the equivalence. Besides, by a full-dependent function, we mean the functions in which the output is dependent on all input variables.

TABLE 4.6: Number of Boolean Functions up to Extended Bit Permutation Equivalence. N_1, N_2, N_3, and N_4 denote the number of all Boolean functions, full-dependent Boolean functions, balanced Boolean functions, and full-dependent and balanced Boolean functions up to the equivalence, resp.

n	N_1	N_2	N_3	N_4
2	4	2	2	1
3	14	10	6	4
4	222	208	58	52
5	616 126	615 904	86 603	86 545

Possible Speed-Up Techniques

To find all the n-bit Boolean functions, or to find all possible implementations of a given Boolean function with latency complexity of d, one can compute all the possible functions in the structure of Proposition 71. As mentioned before, the computational complexity of this search is about $2^{2^d \cdot (2+\log_2(n))}$. In the following, we explain some techniques to reduce the computational complexity of this search. Thereby, we try to remove all the redundant computations through all the possibilities.

REDUCTION ON THE POSSIBILITIES FOR G: Since both of $(\mathcal{I}_{d-1,0}, \mathcal{I}_{d-1,1}, g_{d,0})$ and $(\mathcal{I}_{d-1,1}, \mathcal{I}_{d-1,0}, g_{d,0})$ tuples build the same implementation for $\mathcal{I}_{d,0}$, it is enough to search through one of these tuples. Furthermore, a similar reduction is valid for implementing the other smaller sub-circuits; i. e., both $(\mathcal{I}_{i,2\cdot j}, \mathcal{I}_{i,2\cdot j+1}, g_{i+1,j})$ and $(\mathcal{I}_{i,2\cdot j+1}, \mathcal{I}_{i,2\cdot j}, g_{i+1,j})$ tuples build the same implementation for $\mathcal{I}_{i+1,j}$.

This redundancy can be eliminated by limiting the possibilities for G. Precisely, if $N_{G,i}$ is the number of possible choices for $G_{i,2\cdot j}$ and $G_{i,2\cdot j+1}$, instead of going through all $2 \cdot N_{G,i}^2$ possibilities for $G_{i+1,j}$, it is enough to only check the $2 \cdot \frac{N_{G,i} \cdot (N_{G,i}+1)}{2} = N_{G,i} \cdot (N_{G,i}+1)$ possibilities.[8] Table 4.7 shows the number of possible choices for G after this reduction.

TABLE 4.7: Number of Reduced Choices for G.

d	1	2	3	4	5	6
$N_{G,d}$	2	6	42	$2^{10.82}$	$2^{21.64}$	$2^{43.28}$

[8] Note that the multiplication by 2 is because of the number of possible choices for $g_{i+1,j}$.

On the other hand, we already know that if (G, α, π) is the corresponding implementation for function f, then we can implement $f \oplus 1$ function by using $(\overline{G}, \overline{\alpha}, \pi)$. Thus, if we are searching the Boolean functions up to the extended bit permutation equivalence, we can use this technique to fix the type of one of the gates. For simplicity, we fix $g_{d,0}$ to be a NAND gate. It is important to mention that this reduction is only valid when we search the Boolean functions reduced by the equivalence.

REDUCTION ON THE POSSIBILITIES FOR α: Similar to reducing the possibilities for G, it is possible to reduce the number of possibilities for α. Since the order of inputs to the NAND or the NOR gate do not affect the output, we can

reduce the computation complexity by fixing order of the inputs. Precisely, $x_{\pi(2\cdot i)} \oplus \alpha[2\cdot i]$ and $x_{\pi(2\cdot i+1)} \oplus \alpha[2\cdot i+1]$ are the inputs to the gate $g_{0,i}$ for any $i \in Z_{2^{d-1}}$, and swapping these two inputs does not change the implemented function. We can omit this redundant computation by considering only the choices that $\alpha[2\cdot i] \leq \alpha[2\cdot i+1]$ for any $i \in Z_{2^{d-1}}$. Therefore, instead of checking for all $4^{2^{d-1}} = 2^{2^d}$ possibilities for α, it is enough to check for only $3^{2^{d-1}}$ of them.

Note that using these two reductions, for a given n-bit Boolean function, determining the possible implementations with latency complexity of d is about $2^{11.7} \cdot n^8$, $2^{23.5} \cdot n^{16}$, and $2^{47} \cdot n^{32}$ for $d = 3$, $d = 4$, and $d = 5$, resp.

REDUCTION ON THE POSSIBILITIES FOR π: To find the low-latency implementation of Boolean functions up to the extended bit permutation equivalence, we can also reduce the number of possibilities for π. Since both the implementation with (G, α, π) and the implementation with $(G, \alpha, \pi \circ \pi')$ with π' being a permutation of $\mathbb{Z}_n$, build bit permutation equivalent Boolean functions. Therefore, we can reduce the possibilities for π in a way that if we use π, we do not check for any other $\pi \circ \pi'$ choices. This reduction reduces the computational complexity by a factor of $n!$.

Besides, the implementation with (G, α, π) and the implementation with $(G, \alpha \oplus P(\alpha'), \pi)$ with $\alpha' \in \mathbb{F}_2^n$, build Boolean functions those are different only in a constant addition in the input. Again, we can reduce the possibilities for α in a way that if we use α, we do not check for any other $\alpha \oplus P(\alpha')$ choices. Note that using this reduction requires knowing the choice for π. Since we previously reduced the choices for α with $\alpha_{2\cdot i} \leq \alpha_{2\cdot i+1}$ restriction for any $i \in Z_{2^{d-1}}$, this reduction reduces the computational complexity by some factor smaller than 2^n.

Altogether, even with using all these reductions on the search space, the computational complexity of the search for all n-bit Boolean functions with a latency complexity of d (up-to the extended bit permutations equivalence) will be higher than

$$2^{11.73} \cdot \frac{n^8}{n! \cdot 2^d}, \qquad 2^{23.5} \cdot \frac{n^{16}}{n! \cdot 2^d}, \qquad \text{and} \qquad 2^{47} \cdot \frac{n^{32}}{n! \cdot 2^d}$$

for $d = 3$, $d = 4$, and $d = 5$, resp. For instance, finding all the 6-bit Boolean functions with latency complexity of 4 and 5 need more than 2^{50} and 2^{115} computations, resp. In the following, we present an efficient algorithm to find all the n-bit Boolean functions with latency complexity of d.

An Efficient Algorithm to Find Boolean Functions with Low-Latency Complexity

To find all the n-bit Boolean functions with latency complexity of d, we only need to find them up to the extended bit permutation equivalence. To do this, we assume that for each $d' < d$ and $n' \leq n$, we already found the set of all full-dependent n'-bit Boolean functions with latency complexity of d' reduced by the extended bit permutation. We denote these sets by $\mathcal{F}_{n',d'}$.

If $f \in \mathcal{F}_{n,d}$, then there exist two Boolean functions f_0 and f_1 with a latency complexity of d_0 and d_1, resp., such that $f = f_0 \overline{\wedge} f_1$ and $d_0, d_1 < d$. Note that we do not consider $f_0 \overline{\vee} f_1$ because it is equivalent with $f_0 \overline{\wedge} f_1$.

Assume that f_0 and f_1 are n_0- and n_1-bit full-dependent Boolean functions, resp. Then we know that for each $k \in \mathbb{F}_2$, there exist $f_k^* \in \mathcal{F}_{n_k,d_k}$, bit permutation function P_k with corresponding permutation $\pi_k \in \mathbb{Z}_n^{n_k}$ such that for i and j with $0 \le i < j < n_k$, we have $\pi_k(i) \neq \pi_k(j)$,[9] $\alpha_k \in \mathbb{F}_2^{n_k}$, and $c_k \in \mathbb{F}_2$ which form f_k; i. e., $f_k(x) = f_k^*\big(P_k(x) \oplus \alpha_k\big) \oplus c_k$. As explained previously, in the speed-up techniques, since we are looking for only extended bit permutation equivalence Boolean functions, we can reduce the choices for both π_k and both α_k. For simplicity, we choose $\pi_0 = (0, \ldots, n_0-1)$, $\alpha_0 = 0$. Besides for π_1, we put the restriction that for $0 \le i < j < n_1$, if $n_0 \le \pi_1[i]$ and $n_0 \le \pi_1[j]$, then $\pi_1[i] < \pi_1[j]$. Besides, we restrict α_1 in a way that for $0 \le i < n_1$ if $n_0 \le \pi_1[i]$, then $\alpha_1[i] = 0$.

[9] Note that this restriction is because of that the Boolean functions are full-dependent and therefore π_k must be a permutation of $\mathbb{Z}_{n_k}$.

These reductions on building $\mathcal{F}_{n,d}$, decrease the number of possibilities for π_1 to $(n_0! \cdot n_1!)/\big((n-n_0)! \cdot (n-n_1)! \cdot (n_0+n_1-n)!\big)$ and for α_1 to $2^{n_0+n_1-n}$. All together, the computational complexity of this algorithm to compute all reduced n-bit Boolean functions with latency complexity of d is about

$$\sum_{\substack{d_0,d_1,n_0,n_1 \\ n_0 \le n_1 \le n \\ d_0,d_1 < d}} \frac{n_0! \cdot n_1! \cdot 2^{n_0+n_1-n+2}}{(n-n_0)! \cdot (n-n_1)! \cdot (n_0+n_1-n)!} \cdot |\mathcal{F}_{n_0,d_0}| \cdot |\mathcal{F}_{n_1,d_1}| .$$

For instance, in the case of 6-bit Boolean functions, for latency complexity of 4 and 5, we need to compute f function for about 2^{34} and 2^{39} choices, resp.

It is noteworthy to mention that the set of the Boolean functions built by each of these choices is not the same as $\mathcal{F}_{n,d}$. To achieve $\mathcal{F}_{n,d}$, we need to remove the equivalent functions. This reduction requires about $2^n \cdot n!$ computations for each function. Therefore, to compute $\mathcal{F}_{6,4}$ and $\mathcal{F}_{6,5}$, we need to do about 2^{49} and 2^{55} computations, resp. In a similar way, the computational complexity of $\mathcal{F}_{7,4}$ and $\mathcal{F}_{8,5}$ are about 2^{53} and 2^{60}, resp.

Using the above algorithm, we compute all the Boolean functions in $\mathcal{F}_{n,d}$ for n and d values whose computation complexity is less than 2^{50} computations. The number of functions in $\mathcal{F}_{n,d}$ is shown in Table 4.8.

TABLE 4.8: Number of Full-Dependent n-bit (Balanced) Boolean Functions with Latency Complexity of d and Reduced by the Extended Bit Permutation.

$d \backslash n$	3	4	5	6	7	8
2	3	3				
3	5	54	159	170	64	20
4	2	149	109 674	18 449 993	?	?
5		2	506 013	?	?	?
6			148	?	?	?

Balanced

$d \backslash n$	3	4	5	6	7	8
2	1					
3	2	6	8	3		
4	1	45	12 128	833 520	?	?
5		1	74 390	?	?	?
6			27	?	?	?

4.2.3 *The Bijective S-boxes with Low-Latency Complexity*

Using the Boolean functions with low-latency complexity found in the previous section, in this section, we build bijective S-boxes with a low-latency complexity. To build an n-bit S-box with latency complexity of d, as of the S-box's coordinate functions, we can use all the balanced Boolean functions equivalent to one of the representative balanced functions in one of the $\mathcal{F}_{n',d'}$ sets with $d' \le d$ and $n' \le n$. To do this, we restrict ourselves to only find the S-boxes that each of its coordinate functions is a full-dependent Boolean function, i. e., each output bit of the S-box is dependent on all of the input bits. Besides, we put a limit on the linearity and the uniformity of the S-boxes. Namely, we are interested in the S-boxes such as S with $\mathrm{lin}(S) \le \ell$ and $\mathrm{uni}(S) \le u$. Therefore, we only need to use the balanced Boolean functions equivalent to one of the balanced functions and with linearity at most ℓ in one of the $\mathcal{F}_{n,d'}$ sets with $d' \le d$. We define $\mathcal{F}^*_{n,d,\ell}$ as following:

$$\mathcal{F}^*_{n,d,\ell} = \Big\{ f \mid f \in \bigcup_{d'=1}^{d} \mathcal{F}_{n,d'}, f : \text{balanced}, \mathrm{lin}(f) \le \ell \Big\}$$

that is the set of all n-bit full-dependent Boolean functions with a latency complexity at most d and linearity at most ℓ.

Any full-dependent n-bit S-box with latency complexity d and linearity ℓ can be formed as $S = (f_0, \ldots, f_{n-1})$ such that for all $i \in \mathbb{Z}_n$, we have $f_i = f_i^*\big(P_0(\cdot) \oplus \alpha_0\big) \oplus c_0$ where $f_i^* \in \mathcal{F}^*_{n,d,\ell}$, P_i is a bit permutation function, $\alpha_i \in \mathbb{F}_2^n$, and $c_i \in \mathbb{F}_2$. Searching through all the possibilities, for all f_i, P_i, α_i, and c_i, we need to consider $\big(|\mathcal{F}^*_{n,d,\ell}| \cdot n! \cdot 2^{n+1}\big)^n$ cases. By fixing $\alpha_0 = 0$, P_0 to the identity function, and $c_i = 0$ for all $i \in \mathbb{Z}_n$, we can find all the S-boxes up to extended bit permutation equivalence. Besides, due to the bit permutation in the output bits, we can also fix the order of the coordinate functions. Then the computational complexity of the search is reduced to about $|\mathcal{F}^*_{n,d,\ell}|^n \cdot (n!)^{n-2} \cdot 2^{n^2-n}$. For instance, to build 6-bit S-boxes, the complexity of this search is about $2^{68} \cdot |\mathcal{F}^*_{6,d,\ell}|^6$, and considering that $|\mathcal{F}^*_{6,4,16}| = 1037$, then finding 6-bit S-boxes with the minimum linearity, 16, and the minimum latency, 4, needs considering about 2^{128} possibilities. In the following, we present an algorithm to reduce the computational complexity of this search.

Step-By-Step Method for Building Bijective S-boxes

We use the property of the bijective S-boxes together with the definition for linearity.

Lemma 74. *For an n-bit bijective S-box S with linearity ℓ, each of its non-trivial component functions, namely $\langle \alpha, S \rangle$ with $\alpha \in \mathbb{F}_2^n \setminus \{0\}$, is balanced and has a linearity of at most ℓ.*

Using Lemma 74 makes it possible to filter out some of the possibilities, only by having some of the coordinate functions. Precisely, assume that f_0^* and f_1 are already chosen, then without choosing other coordinate functions, we can check for balancedness and linearity of $f_0^* \oplus f_1$. If $f_0^* \oplus f_1$ is balanced and has a linearity at most ℓ, then we choose the third coordinate function, f_2. Again, we can check for balancedness and linearity of $f_0^* \oplus f_2$, $f_1 \oplus f_2$, and $f_0^* \oplus f_1 \oplus f_2$ functions. Continuing in this way, after choosing the last coordinate function, f_{n-1}, we can check for balancedness and linearity of other $2^{n-1}-1$ component functions. If these $2^{n-1}-1$ conditions are met, then we have a bijective S-box with linearity at most ℓ, and we can compute its uniformity.

Assuming that the average probability of satisfying all 2^i-1 conditions over all possible choices for f_i, is denoted by p_i, then the computational complexity of this search is about

$$\frac{1}{n!} \cdot |\mathcal{F}^*_{n,d,\ell}| \cdot N_f \cdot \Big(1 + p_1 \cdot N_f \cdot \big(1 + p_2 \cdot N_f \cdot \big(\ldots (1 + p_{n-1})\big)\big)\Big) \approx$$
$$\frac{1}{n!} \cdot |\mathcal{F}^*_{n,d,\ell}| \cdot N_f \cdot (1 + p_1 \cdot N_f + p_1 \cdot p_2 \cdot N_f^2 + \ldots + p_1 \cdot \ldots \cdot p_{n-2} \cdot N_f^{n-2})$$

where $N_f = |\mathcal{F}^*_{n,d,\ell}| \cdot n! \cdot 2^n$, and the first division by $n!$ is because of that (due to the output bit permutation) the coordinate functions can be ordered. Note that without using this step-by-step choosing of the coordinate functions, the complexity of the search is about $|\mathcal{F}^*_{n,d,\ell}| \cdot N_f^{n-1}/n!$, which is significantly

larger than when we choose the coordinate functions step-by-step. Moreover, we use the following technique to omit a dominant part of the computations.

After choosing f_0^* in step 0, we find all the possible choices for f_1, i. e., we compute the set of

$$\mathcal{F}^{\dagger}_{n,d,\ell,1}(f_0^*) = \big\{ f \mid f = f^*\big(P(\cdot) \oplus \alpha\big), f^* \in \mathcal{F}^*_{n,d,\ell}, \alpha \in \mathbb{F}_2^n, \\ P : \text{bit permutation}, f \oplus f_0^* \text{ fulfills the conditions} \big\}.$$

By fulfilling the conditions by function g, we mean that g is balanced and $\mathrm{lin}(g) \leq \ell$. We know that not only $f_1 \in \mathcal{F}^{\dagger}_{n,d,\ell,1}(f_0^*)$, but also all other coordinate functions must be in this set; i. e., for each $1 \leq i < n$, $f_i \in \mathcal{F}^{\dagger}_{n,d,\ell,1}(f_0^*)$. Note that determining $\mathcal{F}^{\dagger}_{n,d,\ell,1}(f_0^*)$, for a given f_0^*, needs N_f computations and it includes $N_f \cdot p_1$ Boolean functions. Therefore, to build the S-box, in step 0, we choose $f_0^* \in \mathcal{F}^*_{n,d,\ell}$ and compute $\mathcal{F}^{\dagger}_{n,d,\ell}(f_0^*)$.

In step 1, we choose $f_1 \in \mathcal{F}^{\dagger}_{n,d,\ell,1}(f_0^*)$, and then find all the possible choices for f_2, i. e., we compute the set of

$$\mathcal{F}^{\dagger}_{n,d,\ell,2}(f_0^*, f_1) = \big\{ f \in \mathcal{F}^{\dagger}_{n,d,\ell,1}(f_0^*) \mid f \oplus f_1 \text{ and } f \oplus f_1 \oplus f_0^* \\ \text{fulfill the conditions} \big\}.$$

Note that since we only check for $f \in \mathcal{F}^{\dagger}_{n,d,\ell,1}(f_0^*)$, it already fulfills the conditions for $f \oplus f_0^*$. Again, we know that not only $f_2 \in \mathcal{F}^{\dagger}_{n,d,\ell,2}(f_0^*, f_1)$, but also all the next coordinate functions must be in this set; i. e., for each $2 \leq i < n$, $f_i \in \mathcal{F}^{\dagger}_{n,d,\ell,2}(f_0^*, f_1)$. Determining $\mathcal{F}^{\dagger}_{n,d,\ell,2}(f_0^*, f_1)$, for given f_0^* and f_1, needs $N_f \cdot p_1$ computations and it includes $N_f \cdot p_1 \cdot p_2$ Boolean functions.

In step 2, we choose $f_2 \in \mathcal{F}^{\dagger}_{n,d,\ell,2}(f_0^*, f_1)$, and then find all the possible choices for f_3, i. e., we compute the set of

$$\mathcal{F}^{\dagger}_{n,d,\ell,3}(f_0^*, f_1, f_2) = \big\{ f \in \mathcal{F}^{\dagger}_{n,d,\ell,2}(f_0^*, f_1) \mid f \oplus f_2,\ f \oplus f_2 \oplus f_1, \\ f \oplus f_2 \oplus f_0^* \text{ and } f \oplus f_2 \oplus f_1 \oplus f_0^* \text{ fulfill the conditions} \big\}.$$

Again, since we only check for $f \in \mathcal{F}^{\dagger}_{n,d,\ell,2}(f_0^*, f_1)$, it already fulfills the conditions for $f \oplus f_0^*$, $f \oplus f_1$, and $f \oplus f_1 \oplus f_0^*$. Besides, we know that not only $f_3 \in \mathcal{F}^{\dagger}_{n,d,\ell,3}(f_0^*, f_1, f_2)$, but also all the next coordinate functions must be in this set; i. e., for each $3 \leq i < n$, $f_i \in \mathcal{F}^{\dagger}_{n,d,\ell,3}(f_0^*, f_1, f_2)$. Determining $\mathcal{F}^{\dagger}_{n,d,\ell,3}(f_0^*, f_1, f_2)$, for given f_0^*, f_1 and f_2, needs $N_f \cdot p_1 \cdot p_2$ computations and it includes $N_f \cdot p_1 \cdot p_2 \cdot p_3$ Boolean functions.

We continue in this way until we have chosen all the coordinate functions for the S-box and fulfill the conditions. Therefore, the built S-box is a bijection with linearity at most ℓ. Then, we can check for the condition on the S-box's uniformity. On average, this techniques reduces the computational complexity of building a bijective S-box to

$$\frac{1}{n!} \cdot |\mathcal{F}^*_{n,d,\ell}| \cdot N_f \cdot \Big(1 + p_1 \cdot (p_1 \cdot N_f) \cdot \big(1 + p_2 \cdot (p_2 \cdot N_f) \cdot \big(\ldots (1 + p_{n-1}) \big) \big) \Big) \approx \\ \frac{1}{n!} \cdot |\mathcal{F}^*_{n,d,\ell}| \cdot N_f \cdot (1 + p_1^2 \cdot N_f + p_1^2 \cdot p_2^2 \cdot N_f^2 + \ldots + p_1^2 \cdot \ldots \cdot p_{n-2}^2 \cdot N_f^{n-2})$$

Clearly, the modification explained above makes the step-by-step algorithm faster than before.

USING THE UPPER LIMIT ON THE UNIFORMITY: Similar to using the upper limit on the S-box's linearity, we can also use the limit on the uniformity of the S-box in the intermediate steps of the algorithm.

Lemma 75. *For an n-bit S-box $S = (f_0, \ldots, f_{n-1})$ with uniformity u, the uniformity of the sub-S-box $S'_i = (f_0, \ldots, f_i)$ with $i < n$ is limited by*

$$\min\{u \cdot 2^{n-i-1}, 2^n\} .$$

Applying this lemma, in step i of the step-by-step algorithm, after choosing the coordinate function f_i, it is possible to check uniformity of the sub-S-box $S'_i = (f^*_0, f_1 \ldots, f_i)$. If it meets all the conditions, then we continue; otherwise, we choose another function as i-th coordinate function.

It is noteworthy to mention that for small i values, this condition does not filter the choices for the coordinate function. For instance, when $i = 0$, then uniformity of f^*_0 is limited by $\min\{u \cdot 2^{n-1}, 2^n\} = 2^n$ that this condition is trivial and indeed we do not need to check for it. For the case $i = 1$, the uniformity of (f^*_0, f_1) is limited by $\min\{u \cdot 2^{n-2}, 2^n\}$, and it is non-trivial if $u = 2$, i. e., it is meaningful to check this condition if we are looking for APN S-boxes.

Results on the Low-Latency Bijective S-boxes

TABLE 4.9: Number of the Boolean functions in $\mathcal{F}^*_{n,d,\ell}$ with Minimum Possibility for linearity ℓ, i. e., Number of Full-Dependent n-bit Balanced Boolean Functions with Minimum Possible Linearity and with Latency Complexity of d and Reduced by the Extended Bit Permutation.

n	3	4	5	6
lin	4	8	8	16
$d=2$	1			
$d=3$	2	4		
$d=4$		34	88	1073
$d=5$			1279	?

To build low-latency S-boxes, we start with a lower limit on the linearity of the S-box. The minimum possible linearity of the Boolean functions in $\mathcal{B}_n$ to build bijective S-boxes is equal to 4, 8, 8, and 16 for n equals 3, 4, 5, and 6, resp. Table 4.9 shows the number of full-dependent n-bit balanced Boolean functions reduced by the extended bit permutation and with the given minimum linearity and with latency complexity of d. In the following, we explain the results for building bijective S-boxes for different n values. Note that the number of Boolean functions and the number of the S-boxes are the ones reduced by extended bit permutation equivalence.

3-BIT S-BOXES: Even though, there is one Boolean function in $\mathcal{F}^*_{3,2,4}$, using only this function and its equivalent Boolean functions, it is not possible to build bijective 3-bit S-boxes with linearity of smaller than 8. But for the latency complexity of 3, since all the 3-bit balanced Boolean functions with linearity 4 are included in $\mathcal{F}^*_{3,3,4}$, we can build all the bijective 3-bit S-boxes with linearity 4 and uniformity 2 those are affine equivalent to the inversion in $\mathbb{F}_{2^3}$. Up-to extended bit permutation equivalence, there are 7 of such S-boxes.

4-BIT S-BOXES: There are 4 Boolean functions in $\mathcal{F}^*_{4,3,8}$ with linearity of 8. Using these Boolean functions and their equivalent Boolean functions, it is possible to build 152 different (up to extended bit permutation equivalence) 4-bit bijective S-boxes with linearity 8 and uniformity of 4.

Note that $\mathcal{F}^*_{4,4,8}$ includes all the 4-bit Boolean functions with linearity 8. Therefore, we can build any 4-bit bijective S-box with linearity 8 and uniformity 4, those are named as *Golden* S-boxes by [LP07].

5-BIT S-BOXES: Minimum linearity of the Boolean functions in $\mathcal{F}_{5,3}$ is 16 that there are 4 of such Boolean functions in $\mathcal{F}^*_{5,3,16}$. Using these Boolean

functions and their equivalent Boolean functions, it is possible to build 4 of bijective S-boxes of 5-bits with linearity 16 and uniformity of 6.

Also, there are 88 Boolean functions in $\mathcal{F}^*_{5,4,8}$ that using these functions and their equivalent Boolean functions, it is possible to build 2 510 different (up to extended bit permutation equivalence) bijective S-boxes of 5-bits with linearity 8 and uniformity of 2.

Note that the rest of the balanced 5-bit Boolean functions with minimum linearity are included in $\mathcal{F}^*_{5,5,8}$. This means that any 5-bit S-box with minimum linearity can be built with a latency complexity of at most 5.

6-BIT S-BOXES: $\mathcal{F}_{6,4}$ includes 1073 balanced 6-bit Boolean functions with linearity 16. Using these Boolean functions, we find a bijective S-box with linearity 16 and uniformity of 4. The truth table of this S-box in hexadecimal value is given below.

```
(00,01,02,03,04,06,3e,3c,08,11,0e,17,2b,33,35,2d,
 19,1c,09,0c,15,13,3d,3b,31,2c,25,38,3a,26,36,2a,
 34,1d,37,1e,30,1a,0b,21,2e,1f,29,18,0f,3f,10,20,
 28,05,39,14,24,0a,0d,23,12,27,07,32,1b,2f,16,22)
```

It is noteworthy to mention that since we did not finish all the computations for this search, we are not sure if this is the only 6-bit S-box with low-latency complexity, linearity of 14 and uniformity of 4.

Part III

CRYPTOGRAPHIC ANALYSIS

5

Computing EDP of Differentials and ELP of Linear Hulls

Differential and Linear Cryptanalysis are the two most prominent statistical attacks; thus the resistance of block ciphers against them must be studied carefully. However, as explained in Sections 2.3.5 and 2.3.6, it is shown that evaluating or even approximating the probability of a differential or the correlation of a linear hull is not an easy task. As *truncated characteristics* are the essential objects for assessing the security of many modern SPN ciphers against differential and linear cryptanalysis, the absence of suitable algorithms to tackle this problem was – and still is – an interesting research area within symmetric cryptography.

In this chapter, we show in Section 5.1 that although the previous main method for approximating the EDP of a truncated differential is fast, it is not reliable and may produce estimations very different from the correct value. Then, in Section 5.2, we introduce new methods that, based only on Assumption 16, allow to practically compute the EDP of (truncated) differentials and the ELP of (multidimensional) linear hulls. That is, it is possible to compute the exact sum of the EDP or ELP of all characteristics that follow a given activity pattern. Later, in Section 5.3, these methods are applied to various recent SPN ciphers, and we discuss the results.

This chapter is based on the article [ELR20] that the author of the thesis wrote together with Maria Eichlseder and Gregor Leander.

[ELR20] Eichlseder, Leander, and Rasoolzadeh, "Computing Expected Differential Probability of (Truncated) Differentials and Expected Linear Potential of (Multidimensional) Linear Hulls in SPN Block Ciphers"

5.1 PREVIOUS METHODS FOR ESTIMATING EDP

Since the introduction of (truncated) differentials, various methods for estimating the corresponding probability have been proposed, e. g., [KB96; Mor+99; MG01; AL13; Köl14; Can+15; Leu16; AK19; EK18]. Generally speaking, there are two main opposite approaches, as well as a continuum of trade-offs and variations between: The first direction is to treat a differential as a collection of individual differential characteristics and derive an estimate by enumerating and evaluating individual characteristics based on the transition probabilities for the non-linear layer. The second direction focuses on local properties such as the conditions imposed on characteristics

by the pattern of the truncated differential characteristic, primarily in the linear layer, and to derive an estimate based on these conditions.

Unfortunately, until now, those algorithms either add additional assumptions to be efficient or work only in very specific cases. In the former case, they result only in approximations of the probabilities instead of exact values. Even more problematic, it is often not possible to determine if this approximation under- or overestimates the correct value.

In this section, we summarize and discuss the previously proposed methods. Also, an example is introduced to illustrate their limitations, which will serve as a running example throughout the chapter.

5.1.1 *Enumerating Characteristics*

A simple, but generally not very efficient approach to evaluate $\mathrm{EDP}(\alpha_0 \xrightarrow{\mathtt{Enc}} \alpha_r)$ or $\mathrm{EDP}(\mathcal{U} \xrightarrow{\mathtt{Enc}} \mathcal{V})$ follows directly from its definition under Assumption 16:

$$\mathrm{EDP}(\alpha_0 \xrightarrow{\mathtt{Enc}} \alpha_r) = \sum_{\alpha_1,\ldots,\alpha_{r-1}} \prod_{i=0}^{r-1} \mathrm{Prob}\left(\alpha_i \xrightarrow{\mathrm{S}_i} \mathrm{L}^{-1}(\alpha_{i+1})\right).$$

We can compute or approximate this probability by enumerating and individually evaluating all or many of the compatible differential characteristics $(\alpha_0, \ldots, \alpha_r)$ based on their S-box transition probabilities. This works well when only a few compatible characteristics dominate this probability, and these can be identified efficiently. However, because of limiting the applicability of this approach, in practice, this is often not the case, particularly for larger sets $\mathcal{U}, \mathcal{V}$. A heuristic automated approach for enumerating compatible characteristics is implemented in the SAT-based tool CryptoSMT by Kölbl [Köl14] and was evaluated on differentials (α_0, α_r) for a large selection of ciphers by Ankele and Kölbl [AK19].

[Köl14] Kölbl *CryptoSMT: An Easy to Use Tool for Cryptanalysis of Symmetric Primitives*

[AK19] Ankele and Kölbl "Mind the Gap - A Closer Look at the Security of Block Ciphers against Differential Cryptanalysis"

5.1.2 *Transition Probabilities Matrices*

For a precise evaluation, we could consider the $2^n \times 2^n$ transition matrix of differential probabilities from each input difference to each output difference for one round, i. e., the scaled DDT of one round. Under Assumption 16, the powers of this matrix correspond to the differential behavior of multiple rounds. Multiplying this transition probabilities matrix with a vector representing the initial distribution of differences, for example, uniformly among all differences for a fixed activity pattern, gives the distribution of output differences (assuming independence). We obtain the EDP of the truncated differential with marginalization by summing the probability of all differences compatible with the output difference pattern. Correlation matrices introduced by Daemen, Govaerts, and Vandewalle [DGV95] are an equivalent tool for linear cryptanalysis.

[DGV95] Daemen, Govaerts, and Vandewalle "Correlation Matrices"

Explicitly computing or multiplying the full $2^n \times 2^n$ matrices is, however, only feasible in very particular cases, i. e., $n \leq 32$ and a transition matrix that is very sparse, see [AL13] for an application to Katan32 and [Goh19] for Speck32. In most cases, and in particular, this remains infeasible for any n of practical significance. Therefore, the transition matrices must be

[AL13] Albrecht and Leander, "An All-In-One Approach to Differential Cryptanalysis for Small Block Ciphers"

[Goh19] Gohr, "Improving Attacks on Round-Reduced Speck32/64 Using Deep Learning"

reduced by considering only relevant submatrices corresponding to specific differences or by taking marginal distributions for groups of differences.

Also, some fine-grained approaches have been proposed, for example, the transition probabilities matrices for the stochastic analysis of CRYPTON block cipher by Minier and Gilbert [MG01] or the differential analysis of LAC's underlying block cipher, LBLOCK-s, by Leurent [Leu16].

[MG01] Minier and Gilbert "Stochastic Cryptanalysis of Crypton"

[Leu16] Leurent "Differential Forgery Attack Against LAC"

It has also been proposed to apply different marginalization criteria for different layers of the cipher, for example, in the semi-truncated differential analysis of MANTIS by Eichlseder and Kales [EK18]. Sometimes, it is possible to derive a precise closed-form expression for the powers of the matrix, as in the analysis of PRINCE by Canteaut, Fuhr, Gilbert, Naya-Plasencia, and Reinhard [Can+15].

[EK18] Eichlseder and Kales "Clustering Related-Tweak Characteristics: Application to MANTIS-6"

[Can+15] Canteaut, Fuhr, Gilbert, Naya-Plasencia, and Reinhard "Multiple Differential Cryptanalysis of Round-Reduced PRINCE"

These approaches represent trade-offs between the previously discussed two main approaches: Considering only the small submatrices for specific differences is related to enumerating characteristics; Marginalization by simple criteria such as activity patterns is related to evaluating local constraints. Finding a good trade-off requires a dedicated analysis of the specific target cipher and attack scenario.

One way to make this computation efficient is to partition the underlying characteristics by their activity patterns. By partitioning over activity patterns, $\mathrm{EDP}\left(\alpha_0 \xrightarrow{\mathtt{Enc}} \alpha_r \mid (\delta_0, \ldots, \delta_{r-1})\right)$ is used to denote the sum of probabilities of characteristics whose activity pattern is $(\delta_0, \ldots, \delta_{r-1})$. Thereby,

$$\mathrm{EDP}(\alpha_0 \xrightarrow{\mathtt{Enc}} \alpha_r) = \sum_{\delta_1, \ldots, \delta_{r-2}} \mathrm{EDP}\left(\alpha_0 \xrightarrow{\mathtt{Enc}} \alpha_r \mid (\delta_0, \ldots, \delta_{r-1})\right),$$

where

$$\mathrm{EDP}\left(\alpha_0 \xrightarrow{\mathtt{Enc}} \alpha_r \mid (\delta_0, \ldots, \delta_{r-1})\right) = \sum_{\substack{\alpha_1, \ldots, \alpha_{r-1} \\ \forall i\ 0<i<r, \\ \delta(\alpha_i)=\delta_i, \\ \delta(\mathrm{L}_i^{-1}(\alpha_i))=\delta_{i-1}}} \prod_{i=0}^{r-1} \mathrm{Prob}\left(\alpha_i \xrightarrow{\mathtt{S}_i} \mathrm{L}_i^{-1}(\alpha_{i+1})\right).$$

Note that if $\delta(\alpha_0) \neq \delta_0$ or $\delta\left(\mathrm{L}_{r-1}(\alpha_r)\right) \neq \delta_{r-1}$, then these probabilities are equal to zero. Hence, it is always considered that $\delta(\alpha_0) = \delta_0$ and $\delta\left(\mathrm{L}_{r-1}^{-1}(\alpha_r)\right) = \delta_{r-1}$.

Similarly, for truncated differentials, we use the notation $\mathrm{EDP}\left(\mathcal{U} \xrightarrow{\mathtt{Enc}} \mathcal{V} \mid (\delta_0, \ldots, \delta_{r-1})\right)$. Furthermore, we denote the set of all $(2^s-1)^{\mathrm{hw}(\delta_i)}$ values for α_i such that $\delta(\alpha_i) = \delta_i$ by $\mathcal{U}_{\delta_i}$ and the set of all $(2^s-1)^{\mathrm{hw}(\delta_i)}$ values for α_{i+1} such that $\delta\left(\mathrm{L}_i^{-1}(\alpha_{i+1})\right) = \delta_i$ by $\mathcal{V}_{\delta_i}$. Therefore, if $\mathcal{U}$ and $\mathcal{V}$ be same as the $\mathcal{U}_{\delta_0}$ and $\mathcal{V}_{\delta_{r-1}}$, resp., such a truncated that is restricted to a given activity pattern, is called *truncated differential characteristic* or simply *truncated characteristic* and $\mathrm{EDP}\left(\mathcal{U}_{\delta_0} \xrightarrow{\mathtt{Enc}} \mathcal{V}_{\delta_{r-1}} \mid (\delta_0, \ldots, \delta_{r-1})\right)$ is used to denote its probability. Recalling that $\mathrm{hw}(\delta_i)$ is the Hamming weight of δ_i, which is equal to the number of active S-boxes in the i-th round.

Definition 76 (Truncated Differential Characteristic)**.** For a given activity pattern $(\delta_0, \ldots, \delta_{r-1})$, the truncated differential characteristic is the set of all differential characteristics that follow the activity pattern, i. e., all $(\alpha_0, \ldots, \alpha_r)$ with $\delta(\alpha_0) \in \delta_0$, $\delta(\alpha_r) \in \delta_{r-1}$, and $\delta(\alpha_i) = \delta_i, \delta(\mathrm{L}_i^{-1}(\alpha_i)) = \delta_{i-1}$ for any i with $0 < i < r$.

Note that this is different from the typical definition of truncated differentials where $\mathcal{U}$ and $\mathcal{V}$ cover all the corresponding linear subspaces (excluding zero) where $|\mathcal{U}| = 2^{s\cdot \mathrm{hw}(\delta_0)} - 1$ and $|\mathcal{V}| = 2^{s\cdot \mathrm{hw}(\delta_{r-1})} - 1$. The probability of this notion of truncated characteristics is defined as

$$\mathrm{EDP}\left(\mathcal{U}_{\delta_0^*} \xrightarrow{\texttt{Enc}} \mathcal{V}_{\delta_{r-1}^*} \mid (\delta_0, \ldots, \delta_{r-1})^*\right) = \sum_{\substack{\delta'_0,\ldots,\delta'_{r-1} \\ \forall i \in \mathbb{Z}_r,\ \delta'_i \preceq \delta_i}} \frac{(2^s-1)^{\mathrm{hw}(\delta'_0)}}{2^{s\cdot \mathrm{hw}(\delta_0)}-1} \cdot \mathrm{EDP}\left(\mathcal{U}_{\delta'_0} \xrightarrow{\texttt{Enc}} \mathcal{V}_{\delta'_{r-1}} \mid (\delta'_0, \ldots, \delta'_{r-1})\right),$$

which is described over uniformly distributed differences in $\mathcal{U}_{\delta_0^*}$.

Partitioning a (truncated) differential over the truncated characteristics (or the underlying activity patterns) makes it possible to give a close lower estimation by only considering the ones which have a prominent role in the EDP of the (truncated) differential. Those are usually the truncated characteristics with less number of active S-boxes or the ones with less number of conditions in the linear layers.

5.1.3 *Moriai's et al. Classical Approach*

[Mor+99] Moriai, Sugita, Aoki, and Kanda "Security of E2 against Truncated Differential Cryptanalysis"

Moriai, Sugita, Aoki, and Kanda [Mor+99] introduced a method to estimate the EDP of a truncated differential for SPN-like block ciphers. This method, like all the previous methods, assumes independent round keys. This approach has since been widely applied due to its simple, efficient evaluation and its relatively good accuracy for AES-like designs with large, highly uniform S-boxes and strong linear diffusion layers.

The key idea is that, for many truncated characteristics, the S-box does not seem to have a (significant) influence on the EDP. Indeed, the linear layer seems to be much more dominating in this case. More formally, the main idea behind this method is to assume uniform differential probabilities for the S-box.

Assumption 77 (S-box with uniform DDT)**.** *For an s-bit bijective S-box S, we assume that all differential transitions through the S-box are of the form*

$$\mathrm{Prob}(a \xrightarrow{S} b) = \begin{cases} 1 & \text{if } a = b = 0 \\ p & a \neq 0 \text{ and } b \neq 0 \\ 0 & \text{otherwise,} \end{cases}$$

where p is equal to $(2^s-1)^{-1}$.

Note that there is no such S-box that fulfills this assumption. However, assuming an S-box with uniform DDT together with Assumption 16, for the EDP of a truncated characteristic, we would have

$$\mathrm{EDP}\left(\mathcal{U}_{\delta_0} \xrightarrow{\texttt{Enc}} \mathcal{V}_{\delta_{r-1}} \mid (\delta_0, \ldots, \delta_{r-1})\right) = p^{\mathrm{hw}(\delta_0)} \cdot \sum_{\substack{\alpha_0, \alpha_r, \\ \alpha_0 \in \mathcal{U}_{\delta_0}, \\ \alpha_r \in \mathcal{V}_{\delta_{r-1}}}} \sum_{\substack{\alpha_1,\ldots,\alpha_{r-1} \\ \forall i\ 0<i<r, \\ \alpha_i \in \mathcal{U}_{\delta_i} \cap \mathcal{V}_{\delta_{i-1}}}} \prod_{i=0}^{r-1} p^{\mathrm{hw}(\delta_i)}.$$

After simplifications for α_0 and α_r and by defining $A_i = \mathcal{U}_{\delta_i} \cap \mathcal{V}_{\delta_{i-1}}$, i. e., as the set of all possible αs such that $\delta(\alpha) = \delta_i$ and $\delta\left(\mathrm{L}_i^{-1}(\alpha)\right) = \delta_{i-1}$, the

above equation can be rewritten as

$$\sum_{\substack{\alpha_1,\ldots,\alpha_{r-1}\\ \forall i\ 0<i<r,\\ \alpha_i\in A_i}} \prod_{i=0}^{r-2} p^{\mathrm{hw}(\delta_i)} = \prod_{i=0}^{r-2} \left(p^{\mathrm{hw}(\delta_i)} \cdot |A_{i+1}|\right).$$

One can translate $p^{wt(\delta_i)} \cdot |A_{i+1}|$ as the probability that β_i with $\delta(\beta_i) = \delta_i$, transforms to a value with activity pattern δ_{i+1} over the linear layer L_i. We denote this probability with $\mathrm{Prob}(\delta_i \xrightarrow{\delta \mathrm{L}_i} \delta_{i+1})$.

Applying this method, one only needs to know the values of $\mathrm{Prob}(\delta_j \xrightarrow{\delta \mathrm{L}_i} \delta_k)$ for each δ_j, δ_k and each linear layer L_i. This is basically a linear algebra problem, and indeed an efficient algorithm was given in [Mor+99] to compute those values. After computing all $\mathrm{Prob}(\delta_j \xrightarrow{\delta \mathrm{L}_i} \delta_k)$, one can easily compute the EDP of a truncated characteristic and therefore the EDP of a truncated differential under Assumptions 16 and 77.

While Moriai's *et al.* approach remains by far the simplest and most widely used, as it relies on wrong assumptions on the S-box behavior, the validity of the results is unclear. Even worse, it is impossible in general to decide whether the resulting values are an under- or overestimate of the real probabilities.

Example 78. Figure 5.1 shows a truncated characteristic for 5-round MIDORI which is borrowed from [MA19]. The EDP of this truncated characteristic, $\mathrm{EDP}\left(\mathcal{U}_{\delta_0} \to \mathcal{V}_{\delta_4} \mid (\delta_0, \ldots, \delta_4)\right)$ is equal to

[MA19] Moghaddam and Ahmadian, *New Automatic search method for Truncated-differential characteristics: Application to Midori, SKINNY and CRAFT*

$$\begin{aligned}
15^{-3} \cdot \sum \big(& \mathrm{Prob}(a_0 \xrightarrow{S} x_0) \cdot \mathrm{Prob}(a_1 \xrightarrow{S} x_0) \cdot \mathrm{Prob}(a_2 \xrightarrow{S} x_0) \cdot \mathrm{Prob}(x_0 \xrightarrow{S} x_1) \cdot \\
& \mathrm{Prob}(x_1 \xrightarrow{S} x_2) \cdot \mathrm{Prob}(x_1 \xrightarrow{S} x_3) \cdot \mathrm{Prob}(x_1 \xrightarrow{S} x_4) \cdot \mathrm{Prob}(x_4 \xrightarrow{S} x_5) \cdot \\
& \mathrm{Prob}(x_4 \xrightarrow{S} x_6) \cdot \mathrm{Prob}(x_4 \xrightarrow{S} x_7) \cdot \mathrm{Prob}(x_2 \xrightarrow{S} x_8) \cdot \mathrm{Prob}(x_2 \xrightarrow{S} x_9) \cdot \\
& \mathrm{Prob}(x_2 \xrightarrow{S} x_6) \cdot \mathrm{Prob}(x_3 \xrightarrow{S} x_9) \cdot \mathrm{Prob}(x_3 \xrightarrow{S} x_6) \cdot \mathrm{Prob}(x_3 \xrightarrow{S} x_5) \cdot \\
& \mathrm{Prob}(x_5 \xrightarrow{S} c_0) \cdot \mathrm{Prob}(x_5 \xrightarrow{S} c_1) \cdot \mathrm{Prob}(x_6 \xrightarrow{S} c_2) \cdot \mathrm{Prob}(x_9 \xrightarrow{S} c_3) \cdot \\
& \mathrm{Prob}(x_9 \xrightarrow{S} c_4) \cdot \mathrm{Prob}(x_8 \xrightarrow{S} c_7) \cdot \mathrm{Prob}(x_7 \xrightarrow{S} c_8) \cdot \\
& \mathrm{Prob}(x_7 \oplus x_8 \xrightarrow{S} c_5) \cdot \mathrm{Prob}(x_7 \oplus x_8 \xrightarrow{S} c_6) \big)
\end{aligned}$$

where the sum is over $a_0, a_1, a_2, x_0, \ldots, x_9, c_0, \ldots, c_8 \in \mathbb{F}_2^4 \backslash \{0\}$.

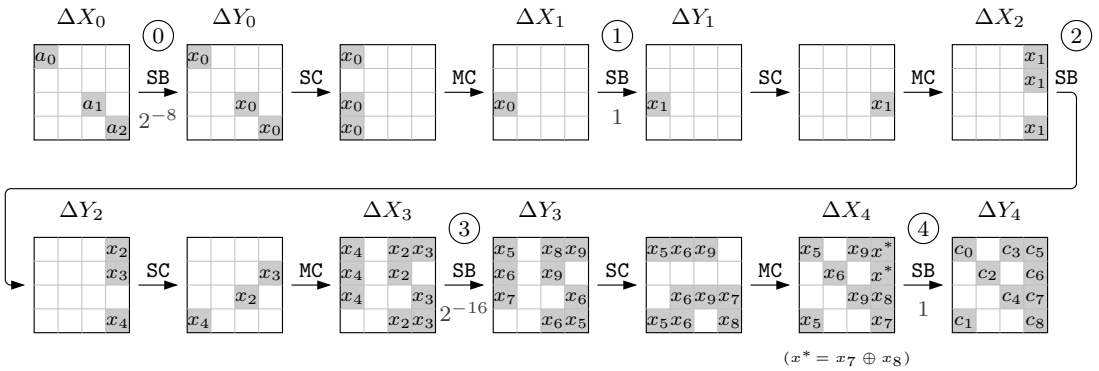


FIGURE 5.1: 5-round Truncated Characteristic for MIDORI64 from [MA19]. Note that the final linear layer is omitted.

To compute the exact value of this EDP, we need to do 25 (number of active S-boxes) table look-ups and arithmetic operations for each 15^{22} value of $a_0, \ldots, c_8$ which is about $2^{90.60}$ table look-ups and arithmetic operations.

On the other hand, using Moriai's *et al.* method, it is enough to multiply

$$\text{Prob}(\delta_0 \xrightarrow{\delta L} \delta_1) = 15^{-2}, \qquad \text{Prob}(\delta_1 \xrightarrow{\delta L} \delta_2) = 1,$$
$$\text{Prob}(\delta_2 \xrightarrow{\delta L} \delta_3) = 1, \qquad \text{Prob}(\delta_3 \xrightarrow{\delta L} \delta_4) = 14 \cdot 15^{-5},$$

resulting in $14 \cdot 15^{-7} \approx 2^{-23.54}$.

To compute the $\text{EDP}\left(\mathcal{U}_{\delta_0^*} \to \mathcal{V}_{\delta_4^*} \mid (\delta_0, \ldots, \delta_4)^*\right)$, we need to consider all other possible underlying activity patterns $(\delta'_0, \ldots, \delta'_4)$. There is only one such activity pattern $(\delta_0, \ldots, \delta'_4)$, where δ_4 and δ'_4 are different on the corresponding S-boxes for c_5 and c_6. $(\delta_0, \ldots, \delta'_4)$ occurs if and only if $x_7 = x_8$, then $\text{Prob}(\delta_3 \xrightarrow{\delta L} \delta'_4) = 15^{-5}$ and $\text{EDP}\left(\mathcal{U}_{\delta_0} \to \mathcal{V}_{\delta_4} \mid (\delta_0, \ldots, \delta'_4)\right) = 15^{-7} \approx 2^{-27.35}$. In this way, $\text{EDP}\left(\mathcal{U}_{\delta_0^*} \to \mathcal{V}_{\delta_4^*} \mid (\delta_0, \ldots, \delta_4)^*\right)$ is equal to $15^{-3}/(16^3 - 1) \approx 2^{-23.72}$, approximated as 2^{-24} in [MA19].

Using the techniques in Section 5.2, we compute the exact value of $\text{EDP}\left(\mathcal{U}_{\delta_0^*} \to \mathcal{V}_{\delta_4^*} \mid (\delta_0, \ldots, \delta_4)^*\right)$ for the S-box given in the specification of MIDORI, which is $2^{-20.60}$. Thus, there is a gap between the exact value and the value given by the method of Moriai *et al.* To show that changes of the S-box can make a significant difference in the value of the EDP, all 4-bit bijective S-boxes are tried, and the EDP of the same characteristic is computed.[1] The minimum observed value is $2^{-22.36}$ for the following two S-boxes:

$$(0, 8, 3, b, 6, 5, 2, 9, e, 4, a, c, 7, 1, d, f),$$
$$(0, 7, 9, e, 2, b, a, 4, 5, 8, d, f, 1, c, 3, 6),$$

while the maximum is $2^{-8.09}$ for any linear S-box. Clearly, the EDP of a truncated differential depends not only on the linear layer of the cipher but also on its S-boxes. To show that this EDP can be different for S-boxes within an affine equivalence class, all the 4-bit S-boxes within the 16 affine equivalence classes with minimum possible uniformity and linearity, which are known as *Golden* S-boxes [LP07], are checked. The minimum and maximum values for the EDP within S-boxes of the same class are given in Table 5.1. Interestingly, the smallest logarithmic difference between the minimum and maximum EDP is for G_3, the affine equivalence class of the inversion in $\mathbb{F}_{2^4}$. The DDTs of the S-boxes in this class are those closest to uniformly distributed. Also, note that the two S-boxes given above are golden S-boxes.

[1] Due to the structure of the MIDORI cipher, it is enough to check for representative S-boxes of affine equivalence classes together with a bijective 4×4 matrix in the output of S-box, i. e., $302 \times 20160 \approx 2^{22.5}$ S-boxes.

[LP07] Leander and Poschmann, "On the Classification of 4 Bit S-Boxes"

TABLE 5.1: The Minimum and Maximum EDP of the Truncated Differential shown in Figure 5.1 among the Golden S-boxes. Note that instead of the probability p, $-\log_2 p$ is shown.

	min	max
G_0	21.55	20.77
G_1	21.60	20.60
G_2	21.61	20.79
G_3	22.30	22.11
G_4	22.36	21.77
G_5	22.26	21.95
G_6	22.28	21.70
G_7	22.32	21.74
G_8	21.62	20.44
G_9	22.07	21.44
G_{10}	22.10	21.40
G_{11}	22.27	21.58
G_{12}	22.36	21.69
G_{13}	22.31	21.78
G_{14}	22.06	21.60
G_{15}	22.06	21.64

5.2 OUR METHOD FOR COMPUTING EDP AND ELP

In this section, we present our methods to compute the EDP of differentials with a given activity pattern under Assumption 16, which is also useful to compute the ELP of linear hulls. We extend it to compute the EDP of truncated differentials or the ELP of multidimensional linear hulls. We also discuss several techniques to reduce the time and/or memory complexity

of the algorithms and finally present an efficient algorithm for computing the EDP of truncated characteristic or the ELP of a truncated linear characteristic. We use two different approaches, one based on a state-view on the differences and one based on a word-view. While the state-based method is easier to apply, the word-based method, in many cases, outperforms the former by better minimizing the dependency between parts of the differential characteristics. Moreover, and of independent technical interest, we use a graph view on the dependency to optimize the order of the computation efficiently for the word-based approaches.

5.2.1 *State-by-State Method*

To compute the $\mathrm{EDP}\big(\alpha_0 \xrightarrow{\mathtt{Enc}} \alpha_r \mid (\delta_0,\ldots,\delta_{r-1})\big)$, first we need to find all possible α_i such that $\delta(\alpha_i)=\delta_i$ and $\delta\big(\mathrm{L}_i^{-1}(\alpha_i)\big)=\delta_{i-1}$ for each i $(1\le i<r)$ and save them in a table A_i. We use $A_i[j]$ to denote j-th possible value for α_i. By expanding the product and separating the sum, we get

$$\sum_{\substack{\alpha_1,\ldots,\alpha_{r-1}\\ \forall i\ 0<i<r,\\ \alpha_i\in A_i}}\prod_{i=0}^{r-1} P\big(\alpha_i \xrightarrow{\mathrm{S}_i} \mathrm{L}_i^{-1}(\alpha_{i+1})\big) = \sum_{\alpha_{r-1}\in A_{r-1}} \mathrm{Prob}\big(\alpha_{r-1} \xrightarrow{\mathrm{S}_{r-1}} \mathrm{L}_{r-1}^{-1}(\alpha_r)\big)\cdot$$

$$\Bigg(\ldots\Big(\sum_{\alpha_2\in A_2}\mathrm{Prob}\big(\alpha_2 \xrightarrow{\mathrm{S}_2} \mathrm{L}_2^{-1}(\alpha_3)\big)\cdot\Big(\sum_{\alpha_1\in A_1}\mathrm{Prob}\big(\alpha_1 \xrightarrow{\mathrm{S}_1} \mathrm{L}_1^{-1}(\alpha_2)\big)\cdot\mathrm{Prob}\big(\alpha_0 \xrightarrow{\mathrm{S}_0} \mathrm{L}_0^{-1}(\alpha_1)\big)\Big)\Big)\cdots\Bigg)$$

This equation offers a round-by-round method to compute the probability. For the S_0 layer and for each $0\le j<|A_1|$, we compute

$$\mathrm{Prob}\big(\alpha_0 \xrightarrow{\mathrm{S}_0} \mathrm{L}_0^{-1}(A_1[j])\big)$$

and insert it in table T_1 as $T_1[j]$. Then, for the S_1 layer and for each $0\le j<|A_2|$, we compute

$$\sum_{k=0}^{|A_1|-1} T_1[k]\cdot\mathrm{Prob}\big(A_1[k] \xrightarrow{\mathrm{S}_1} \mathrm{L}_1^{-1}(A_2[j])\big)$$

and insert it in $T_2[j]$. Continuing in the same way, for the S_i layer and for each $0\le j<|A_{i+1}|$, we compute

$$\sum_{k=0}^{|A_i|-1} T_i[k]\cdot\mathrm{Prob}\big(A_i[k] \xrightarrow{\mathrm{S}_i} \mathrm{L}_i^{-1}(A_{i+1}[j])\big)$$

and insert it in $T_{i+1}[j]$. Finally, for the last round, we compute

$$\sum_{k=0}^{|A_{r-1}|-1} T_{r-1}[k]\cdot\mathrm{Prob}\big(A_{r-1}[k] \xrightarrow{\mathrm{S}_{r-1}} \mathrm{L}_{r-1}^{-1}(\alpha_r)\big)$$

which is equal to the $\mathrm{EDP}\big(\alpha_0 \xrightarrow{\mathtt{Enc}} \alpha_r \mid (\delta_0,\ldots,\delta_{r-1})\big)$.

Regarding the complexity, in round i with $0<i<r-1$ (i. e., all of the rounds except the first and last ones), we need to do multiplication and addition for $|A_i|\cdot|A_{i+1}|$ times. Hence, the complexity of this method in multiplication and addition operations is about

$$|A_1|+|A_{r-1}|+\sum_{i=1}^{r-2}|A_i|\cdot|A_{i+1}|\,.$$

Note that in each round, we only need to keep two consecutive tables, one for the current round to save and one from the previous round to use, so the memory complexity is about

$$\max_{1\le i<r-1}(|A_i|+|A_{i+1}|).$$

Note that this method can also be applied for computing other EDPs and even for computing ELPs with a small modification explained in the following.

To compute the $\mathrm{EDP}\big(\alpha_0 \xrightarrow{\mathtt{Enc}} \alpha_r \mid (\delta_0,\ldots,\delta_{r-1})^*\big)$, we only need to modify the set of α_i values with $1 \le i < r-1$. Thereby, each A_i^* set is defined as the set of all α_i velues such that $\delta(\alpha_i) \preceq \delta_i$ and $\delta\big(\mathrm{L}_i^{-1}(\alpha_i)\big) \preceq \delta_{i-1}$.

In case of the related-tweak EDP of a cipher with linear and separate tweak schedule than the key schedule as explained in Section 2.3.5, we can again use the above method for computing $\mathrm{EDP}\big(\alpha_0 \xrightarrow{\mathtt{Enc}} \alpha_r \mid \Delta T = \gamma,(\delta_0,\ldots,\delta_{r-1})\big)$ by following modification. First, we need to derive the differences in all round tweaks $\gamma_i = \Delta T_i$ from the tweak difference γ. Note that this is possible because the tweak schedule is linear. Then, the A_i sets are defined as the set of all α_i values such that $\delta(\alpha_i) = \delta_i$ and $\delta\big(\mathrm{L}_i^{-1}(\alpha_i \oplus \gamma_i)\big) = \delta_{i-1}$. Besides, we need to modify the S-box layer's output difference in each round and compute $\mathrm{Prob}\big(A_i[k] \xrightarrow{\mathrm{S}_i} \mathrm{L}_i^{-1}(A_{i+1}[j] \oplus \gamma_{i+1})\big)$ instead of the previous one.

To compute the $\mathrm{EDP}\big(\mathcal{U} \xrightarrow{\mathtt{Enc}} \mathcal{V} \mid (\delta_0,\ldots,\delta_{r-1})\big)$ for general sets $\mathcal{U}$ and $\mathcal{V}$ restricted to all elements with activity pattern δ_0 (for $\mathcal{U}$) and all elements transformed by L^{-1} with activity pattern δ_{r-1}, we use a similar method. There, we need to change computations in the first and the last S-box layers. In the first layer, we compute

$$\sum_{\alpha_0\in\mathcal{U}} \mathrm{Prob}\big(\alpha_0 \xrightarrow{\mathrm{S}_0} \mathrm{L}_0^{-1}(A_1[j])\big),$$

while in the last round, we compute

$$\frac{1}{\mathcal{U}} \cdot \sum_{\alpha_r\in\mathcal{V}} \sum_{k=0}^{|A_{r-1}|-1} T_{r-1}[k] \cdot \mathrm{Prob}\big(A_{r-1}[k] \xrightarrow{\mathrm{S}_{r-1}} \mathrm{L}_{r-1}^{-1}(\alpha_r)\big).$$

It is noteworthy to mention that the above extensions and modifications can be combined to compute the remaining EDPs. Besides, note that except the computation complexity of the extension to the truncated EDP, memory and computation complexity of the extensions stays the same, while for the computational complexity of the extension to the truncated EDP, we have

$$|\mathcal{U}| \cdot |A_1| + |\mathcal{V}| \cdot |A_{r-1}| + \sum_{i=1}^{r-2} |A_i| \cdot |A_{i+1}|.$$

This method is not only applicable to compute the EDP of a (truncated) differential; it can be easily modified to compute the ELP of (multidimensional) linear hulls. To do so, we need to replace $\mathrm{Prob}\big(A_i[k] \xrightarrow{\mathrm{S}_i} \mathrm{L}_i^{-1}(A_{i+1}[j])\big)$ by $\mathrm{Cor}\big(A_i[k] \xrightarrow{\mathrm{S}_i} \mathrm{L}_i^{T}(A_{i+1}[j])\big)^2$ and define A_i sets where each A_i with $1 \le i < r-1$ is the set of all α_i values with $\delta(\alpha_i) = \delta_i$ and $\delta\big(\mathrm{L}_i^{T}(\alpha_i)\big) = \delta_{i-1}$.

Algorithm 3 Computing the EDP of a Differential with Given Activity Pattern

Input: α_0, α_r and $(\delta_0, \ldots, \delta_{r-1})$

Output: $\mathrm{EDP}\left(\alpha_0 \xrightarrow{\mathrm{Enc}} \alpha_r \mid (\delta_0, \ldots, \delta_{r-1})\right)$

```
1  procedure ComputeEDP
2      [B_0, A_0, n_0] ← FindAllBetas(δ_0, δ_1, 0)
3      for 0 ≤ j < n_0 do
4          T_0[j] ← SBoxProb(α_0, B_0[j], δ_0, 0)
5      for 1 ≤ i < r − 1 do
6          [B_1, A_1, n_1] ← FindAllBetas(δ_i, δ_{i+1}, i)
7          for 0 ≤ j < n_1 do
8              T_1[j] ← 0
9              for 0 ≤ k < n_0 do
10                 T_1[j] ← T_1[j] + T_0[k] · SBoxProb(A_0[k], B_1[j], δ_i, i)
11         [B_0, A_0, n_0] ← [B_1, A_1, n_1]    ▷ It is just a binary index swapping!
12     p ← 0
13     α_r ← L_{r−1}^{−1}(α_r)
14     for 0 ≤ k < n_0 do
15         p ← p + T_0[k] · SBoxProb(A_0[k], α_r, δ_{r−1}, r−1)
16     return p
```

Finding All Beta Values Fitting to the Activity Pattern of L_i Layer

```
17 procedure FindAllBetas(δ_I, δ_O, i)
18     n ← 0
19     if hw(δ_I) ≤ hw(δ_O) then
20         for all β s.t. δ(β) = δ_I do
21             α ← L_i(β)
22             if δ(α) = δ_O then
23                 B[n] ← β
24                 A[n] ← α
25                 n ← n + 1
26     else
27         for all α s.t. δ(α) = δ_O do
28             β ← L_i^{−1}(α)
29             if δ(β) = δ_I then
30                 B[n] ← β
31                 A[n] ← α
32                 n ← n + 1
33     return [B, A, n]
```

Computing Differential Probability of S_i Layer

```
34 procedure SBoxProb(ΔX, ΔY, δ, i)
35     d ← 1
36     for 0 ≤ j < s do
37         if δ[j] = 1 then
38             d ← d · DDT(S_i)[ΔX[j]][ΔY[j]]
39     return d · 2^{−s·hw(δ)}
```

Algorithm 3 lists our method in detail for computing the $\mathrm{EDP}\left(\alpha_0 \xrightarrow{\texttt{Enc}} \alpha_r \mid (\delta_0, \ldots, \delta_{r-1})\right)$. Note that this algorithm only needs knowledge about the linear layers and DDT of the S-boxes applied in the block cipher. In case that all the linear layers are the same or all the S-boxes are the same, this algorithm can be optimized. More importantly, the FINDALLBETAS procedure can be replaced with a complicated one that, by using linear algebra techniques, reduces the computation cost of this procedure.

Besides, to compute the $\mathrm{EDP}\left(\alpha_0 \xrightarrow{\texttt{Enc}} \alpha_r \mid (\delta_0, \ldots, \delta_{r-1})^*\right)$, we can use the same algorithm by only modifying lines 20, 22, 27, and 29: instead of checking $\delta(\cdot) = \delta_{I/O}$, we check $\delta(\cdot) \preceq \delta_{I/O}$.

In the related-tweak case, to compute the $\mathrm{EDP}\left(\alpha_0 \xrightarrow{\texttt{Enc}} \alpha_r \mid \Delta T = \gamma, (\delta_0, \ldots, \delta_{r-1})^*\right)$, we can again use the same algorithm. First, we derive the differences in all round tweaks $\gamma_i = \Delta T_i$ from the tweak difference γ. We also need to modify FINDALLBETAS to receive one more input γ and change lines 2 and 6 to $\textsc{FindAllBetas}(\delta_i, \delta_{i+1}, \gamma_{i+1})$ and lines 21 and 28 to $\alpha \leftarrow \mathrm{L}(\beta) \oplus \gamma$ and $\beta \leftarrow \mathrm{L}^{-1}(\alpha \oplus \gamma)$, resp.

To compute $\mathrm{EDP}\left(\mathcal{U} \xrightarrow{\texttt{Enc}} \mathcal{V} \mid (\delta_0, \ldots, \delta_{r-1})\right)$ with general $\mathcal{U}$ and $\mathcal{V}$ sets, where the activity pattern of all elements in $\mathcal{U}$ is δ_0 and the activity pattern of elements in $\mathcal{V}$ transformed by L_{r-1}^{-1} is δ_{r-1}, we use a similar method as in Algorithm 3, iterating over all elements of $\mathcal{U}$ and $\mathcal{V}$. Algorithm 4 lists the updated procedure in detail. We can again use the same modifications as for Algorithm 3 to compute the $\mathrm{EDP}\left(\mathcal{U} \xrightarrow{\texttt{Enc}} \mathcal{V} \mid (\delta_0, \ldots, \delta_{r-1})^*\right)$ and the $\mathrm{EDP}\left(\mathcal{U} \xrightarrow{\texttt{Enc}} \mathcal{V} \mid \Delta T = \gamma, (\delta_0, \ldots, \delta_{r-1})\right)$.

Algorithm 4 Computing the EDP of a General Truncated Differential with Given Activity Pattern

Input: $\mathcal{U}, \mathcal{V}$ and $(\delta_0, \ldots, \delta_{r-1})$
Output: $\mathrm{EDP}\left(\mathcal{U} \xrightarrow{\texttt{Enc}} \mathcal{V} \mid (\delta_0, \ldots, \delta_{r-1})\right)$

1 **procedure** COMPUTEEDP
2 $[B_0, A_0, n_0] \leftarrow \textsc{FindAllBetas}(\delta_0, \delta_1, 0)$
3 **for** $0 \le j < n_0$ **do**
4 $T_0[j] \leftarrow 0$
5 **for** $0 \le k < |\mathcal{U}|$ **do**
6 $T_0[j] \leftarrow T_0[j] + \textsc{SBoxProb}(\mathcal{U}[k], B_0[j], \delta_0, 0)$
7 **for** $1 \le i < r-1$ **do**
8 $[B_1, A_1, n_1] \leftarrow \textsc{FindAllBetas}(\delta_i, \delta_{i+1}, i)$
9 **for** $0 \le j < n_1$ **do**
10 $T_1[j] \leftarrow 0$
11 **for** $0 \le k < n_0$ **do**
12 $T_1[j] \leftarrow T_1[j] + T_0[k] \cdot \textsc{SBoxProb}(A_0[k], B_1[j], \delta_i, i)$
13 $[B_0, A_0, n_0] \leftarrow [B_1, A_1, n_1]$
14 $p \leftarrow 0$
15 **for** $0 \le j < |\mathcal{V}|$ **do**
16 $\mathcal{V}[j] \leftarrow \mathrm{L}_{r-1}^{-1}(\mathcal{V}[j])$
17 **for** $0 \le k < n_0$ **do**
18 $p \leftarrow p + T_0[k] \cdot \textsc{SBoxProb}(A_0[k], \mathcal{V}[j], \delta_{r-1}, r-1)$
19 **return** $p \cdot |\mathcal{U}|^{-1}$

Performance Improvements by Eliminating S-boxes

We can improve this method's complexity for computing the EDP of a truncated characteristic by eliminating some unnecessary computations.

Lemma 79. *For a bijective s-bit S-box S, and any non-zero $a \in \mathbb{F}_2^s$, any $b \in \mathbb{F}_2^s$*

$$\sum_{x \in \mathbb{F}_2^s \setminus \{0\}} \mathrm{Prob}(x \xrightarrow{S} a) = 1 \quad or \quad \sum_{x \in \mathbb{F}_2^s \setminus \{0\}} \mathrm{Prob}(a \xrightarrow{S} x) = 1\,,$$
$$\sum_{x \in \mathbb{F}_2^s} \mathrm{Prob}(x \xrightarrow{S} b) = 1 \quad or \quad \sum_{x \in \mathbb{F}_2^s} \mathrm{Prob}(b \xrightarrow{S} x) = 1\,.$$

Moreover, for an S-box layer S *which is application of m parallel s-bit bijective S-boxes, and for any $\alpha, \beta \in \mathbb{F}_2^{s \cdot m}$ with $\delta(\alpha) = \delta_i$ and $\delta(\beta) \preceq \delta_i$, we have*

$$\sum_{\substack{x \in \mathbb{F}_2^{s \cdot m} \\ \delta(x) = \delta_i}} \mathrm{Prob}(x \xrightarrow{\mathrm{S}} \alpha) = 1 \quad or \quad \sum_{\substack{x \in \mathbb{F}_2^{s \cdot m} \\ \delta(x) = \delta_i}} \mathrm{Prob}\left(\alpha \xrightarrow{\mathrm{S}} x\right) = 1,$$
$$\sum_{\substack{x \in \mathbb{F}_2^{s \cdot m} \\ \delta(x) \preceq \delta_i}} \mathrm{Prob}(x \xrightarrow{\mathrm{S}} \beta) = 1 \quad or \quad \sum_{\substack{x \in \mathbb{F}_2^{s \cdot m} \\ \delta(x) \preceq \delta_i}} \mathrm{Prob}\left(\beta \xrightarrow{\mathrm{S}} x\right) = 1.$$

Applying results of Lemma 79 in the EDP of a truncated characteristic, it is possible to omit the computations for the first and last S-box layers.

Lemma 80. *For a truncated characteristic $(\delta_0, \ldots, \delta_{r-1})$*

$$\mathrm{EDP}\left(\mathcal{U}_{\delta_0} \xrightarrow{\mathrm{Enc}} \mathcal{V}_{\delta_{r-1}} \mid (\delta_0, \ldots, \delta_{r-1})\right) = (2^s - 1)^{-\mathrm{hw}(\delta_0)} \cdot \sum_{\substack{\alpha_1, \ldots, \alpha_{r-1} \\ \forall i\ 0 < i < r, \\ \alpha_i \in A_i}} \prod_{i=1}^{r-2} \mathrm{Prob}\left(\alpha_i \xrightarrow{\mathrm{S}_i} \mathrm{L}_i^{-1}(\alpha_{i+1})\right),$$

Note that Lemma 80 can be modified to compute the $\mathrm{EDP}\left(\mathcal{U}_{\delta_0^*} \xrightarrow{\mathrm{Enc}} \mathcal{V}_{\delta_{r-1}^*} \mid (\delta_0, \ldots, \delta_{r-1})^*\right)$ and the $\mathrm{EDP}\left(\mathcal{U}_{\delta_0} \xrightarrow{\mathrm{Enc}} \mathcal{V}_{\delta_{r-1}} \mid \Delta T = \gamma, (\delta_0, \ldots, \delta_{r-1})\right)$ in similar way as explained previously. Algorithm 5 lists the detailed procedure for computing the $\mathrm{EDP}\left(\mathcal{U}_{\delta_0} \xrightarrow{\mathrm{Enc}} \mathcal{V}_{\delta_{r-1}} \mid (\delta_0, \ldots, \delta_{r-1})\right)$ and again it can be modified to compute the $\mathrm{EDP}\left(\mathcal{U}_{\delta_0} \xrightarrow{\mathrm{Enc}} \mathcal{V}_{\delta_{r-1}} \mid (\delta_0, \ldots, \delta_{r-1})^*\right)$ and the $\mathrm{EDP}\left(\mathcal{U}_{\delta_0} \xrightarrow{\mathrm{Enc}} \mathcal{V}_{\delta_{r-1}} \mid \Delta T = \gamma, (\delta_0, \ldots, \delta_{r-1})\right)$.

It is important to mention that, in the case of computing the ELP of a multidimensional linear hull, this technique is not easily applicable. The reason for this is that for a given $a \in \mathbb{F}_2^s$, $\sum_{x \in \mathbb{F}_2^s} \mathrm{Cor}(x \xrightarrow{S} a)^2$ is not a constant value and it varies for different as. Instead, we can compute $\sum_{x \in \mathbb{F}_2^s} \mathrm{Cor}(x \xrightarrow{S} a)^2$ for each $a \in \mathbb{F}_2^s$ and save it in a table, and then use it once for S_0 and once again in the last step for S_{r-1}.

Example 81. Using this technique, the EDP of the truncated characteristic given in Example 78 and Figure 5.1 is equal to

$$\mathrm{EDP}\left(\mathcal{U}_{\delta_0} \xrightarrow{\mathrm{Enc}} \mathcal{V}_{\delta_4} \mid (\delta_0, \ldots, \delta_4)\right) = 15^{-3} \cdot \sum (p_{01} \cdot p_{12} \cdot p_{13} \cdot p_{14} \cdot p_{45} \cdot p_{46} \cdot p_{47} \cdot p_{28} \cdot p_{29} \cdot p_{26} \cdot p_{39} \cdot p_{36} \cdot p_{35}).$$

Algorithm 5 Computing the EDP of a Truncated Differential Characteristic

Input: $(\delta_0, \ldots, \delta_{r-1})$
Output: $\mathrm{EDP}\left(\mathcal{U}_{\delta_0} \xrightarrow{\mathtt{Enc}} \mathcal{V}_{\delta_{r-1}} \mid (\delta_0, \ldots, \delta_{r-1})\right)$

procedure COMPUTEEDP
  $[B_0, A_0, n_0] \leftarrow$ FINDALLBETAS$(\delta_0, \delta_1, 0)$
  for $0 \le j < n_0$ **do**
    $T_0[j] \leftarrow 1$
  for $1 \le i < r-1$ **do**
    $[B_1, A_1, n_1] \leftarrow$ FINDALLBETAS$(\delta_i, \delta_{i+1}, i)$
    for $0 \le j < n_1$ **do**
      $T_1[j] \leftarrow 0$
      for $0 \le k < n_0$ **do**
        $T_1[j] \leftarrow T_1[j] + T_0[k] \cdot$ SBOXPROB$(A_0[k], B_1[j], \delta_i, i)$
    $[B_0, A_0, n_0] \leftarrow [B_1, A_1, n_1]$
  $p \leftarrow 0$
  for $0 \le j < n_0$ **do**
    $p \leftarrow p + T_0[j]$
  return $p \cdot (2^s - 1)^{-\mathrm{hw}(\delta_0)}$

where p_{ij} denotes $P(x_i \xrightarrow{S} x_j)$ and the sum is over $x_0, \ldots, x_9 \in \mathbb{F}_2^4 \setminus \{0\}$ with condition of $x_7 \neq x_8$. If we want to compute the $\mathrm{EDP}\left(\mathcal{U}_{\delta_0^*} \xrightarrow{\mathtt{Enc}} \mathcal{V}_{\delta_4^*} \mid (\delta_0, \ldots, \delta_4)^*\right)$, we just need to compute

$$\mathrm{EDP}\left(\mathcal{U}_{\delta_0^*} \xrightarrow{\mathtt{Enc}} \mathcal{V}_{\delta_4^*} \mid (\delta_0, \ldots, \delta_4)^*\right) = \frac{1}{(16^3 - 1)} \cdot \sum (\ldots)$$

with the sum over $x_0, \ldots, x_9 \in \mathbb{F}_2^4$ with no conditions. To compute these equations without using the proposed state-by-state, we would need to do 13 table look-ups and arithmetic operations for each $14 \cdot 15^9$ and 16^{10} values of $x_0, \ldots, x_9$ which are about $2^{42.67}$ and $2^{43.70}$ computations, resp., with negligible memory.

On the other hand, computing the $\mathrm{EDP}\left(\mathcal{U}_{\delta_0} \xrightarrow{\mathtt{Enc}} \mathcal{V}_{\delta_4} \mid (\delta_0, \ldots, \delta_4)\right)$ with our state-by-state method needs 2 table look-ups and arithmetic operations for each 15^2 values for x_0 and x_1 in $\mathtt{S}_1$ layer, 4 table look-ups and arithmetic operations for each 15^4 values for x_1, x_2, x_3, x_4 in $\mathtt{S}_2$ layer and 10 table look-ups and arithmetic operations for each $14 \cdot 15^7$ values for $x_2, x_3, \ldots, x_9$ in $\mathtt{S}_3$ layer that in total it is about $2^{34.48}$ table look-ups and arithmetic operations. Note that this method, in each round needs $\mathrm{hw}(\delta_i) + 1$ (with $1 \le i < r-1$) table look-ups, one for calling the reduced-round probability from table T_i and the rest for active S-boxes of current (i-th) round. Besides, computing the $\mathrm{EDP}\left(\mathcal{U}_{\delta_0^*} \xrightarrow{\mathtt{Enc}} \mathcal{V}_{\delta_4^*} \mid (\delta_0, \ldots, \delta_4)^*\right)$, in the similar way needs $2 \cdot 16^2 + 4 \cdot 16^4 + 10 \cdot 16^8 \approx 2^{35.32}$ table look-ups and arithmetic operations. Moreover, both computations need about $16^5 = 2^{20}$ blocks of memory.

Not only the active S-boxes in the first and the last rounds can be simplified, but also some of the active S-boxes in the second and second-to-last rounds. This is possible if there is an active nibble of ΔX_1 (or ΔY_{r-2}) that is linearly independent of the other active nibbles of the state to satisfy the conditions of the activity pattern.

Example 82. In the previous example, in ΔX_1, there is only one active S-box (corresponding to x_0) and it can be removed from both above equations:

$$\begin{aligned}\mathrm{EDP}\left(\mathcal{U}_{\delta_0} \xrightarrow{\mathtt{Enc}} \mathcal{V}_{\delta_4} \mid (\delta_0,\dots,\delta_4)\right) &= \frac{1}{15^3} \sum_{\substack{x_0,\dots,x_9 \\ x_7\neq x_8}} (p_{01}\cdot p_{12}\cdot \cdots \cdot p_{35}) \\ &= \frac{1}{15^3} \sum_{\substack{x_1,\dots,x_9 \\ x_7\neq x_8}} \big((\sum_{x_0} p_{01})\cdot p_{12}\cdot \cdots \cdot p_{35}\big) \\ &= \frac{1}{15^3} \sum_{\substack{x_1,\dots,x_9 \\ x_7\neq x_8}} (p_{12}\cdot \dots \cdot p_{35}) \\ \mathrm{EDP}\left(\mathcal{U}_{\delta_0^*} \xrightarrow{\mathtt{Enc}} \mathcal{V}_{\delta_4^*} \mid (\delta_0,\dots,\delta_4)^*\right) &= \dots = \frac{1}{(16^3-1)} \sum_{x_1,\cdots,x_9} \big(p_{12}\cdot \cdots \cdot p_{35}\big).\end{aligned}$$

Note that even though x_7 and x_8 appear only once in ΔY_3, due to the condition $x_7 \neq x_8$, we cannot remove them from equation for the $\mathrm{EDP}\big(\mathcal{U}_{\delta_0} \xrightarrow{\mathtt{Enc}} \mathcal{V}_{\delta_4} \mid (\delta_0,\dots,\delta_4)\big)$. But since there is no such condition in the $\mathrm{EDP}\big(\mathcal{U}_{\delta_0} \xrightarrow{\mathtt{Enc}} \mathcal{V}_{\delta_4} \mid (\delta_0,\dots,\delta_4)^*\big)$, we can remove both of them to get, with the sum over $x_1,\dots,x_6,x_9 \in \mathbb{F}_2^4$:

$$\mathrm{EDP}\left(\mathcal{U}_{\delta_0^*} \xrightarrow{\mathtt{Enc}} \mathcal{V}_{\delta_4^*} \mid (\delta_0,\dots,\delta_4)^*\right) = \frac{1}{(16^3-1)} \sum_{x_1,\cdots,x_6,x_9} \big(p_{12}\cdot p_{13}\cdot p_{14}\cdot p_{45}\cdot p_{46}\cdot p_{29}\cdot p_{26}\cdot p_{39}\cdot p_{36}\cdot p_{35}\big).$$

To compute these probabilities, without using the state-by-state method, we would need $12\cdot 14\cdot 15^8 \approx 2^{38.65}$ and $10\cdot 16^7 \approx 2^{31.32}$ table look-ups and arithmetic operations, resp., with a negligible memory. But by using the state-by-state method, to compute the $\mathrm{EDP}\big(\mathcal{U}_{\delta_0} \xrightarrow{\mathtt{Enc}} \mathcal{V}_{\delta_4} \mid (\delta_0,\dots,\delta_4)\big)$ after removing unnecessary S-boxes, we need $15^4+14\cdot 15^7 \approx 2^{31}$ operations with 2^{20} blocks of memory and to compute the $\mathrm{EDP}\big(\mathcal{U}_{\delta_0^*} \xrightarrow{\mathtt{Enc}} \mathcal{V}_{\delta_4^*} \mid (\delta_0,\dots,\delta_4)^*\big)$, it is $16^4+16^6 \approx 2^{24}$ operations with $2\cdot 16^3 = 2^{13}$ blocks of memory.

Thus, there might be some unnecessary S-boxes that can be simplified in computing the $\mathrm{EDP}\big(\mathcal{U}_{\delta_0} \xrightarrow{\mathtt{Enc}} \mathcal{V}_{\delta_4} \mid (\delta_0,\cdots,\delta_{r-1})\big)$ and the $\mathrm{EDP}\big(\mathcal{U}_{\delta_0^*} \xrightarrow{\mathtt{Enc}} \mathcal{V}_{\delta_{r-1}^*} \mid (\delta_0,\cdots,\delta_{r-1})^*\big)$. It is noteworthy to mention that since the number of such S-boxes may be higher in the second case, the computation cost of the $\mathrm{EDP}\big(\mathcal{U}_{\delta_0^*} \xrightarrow{\mathtt{Enc}} \mathcal{V}_{\delta_{r-1}^*} \mid (\delta_0,\cdots,\delta_{r-1})^*\big)$ might be lower.

Finding Partially Independent S-boxes

To find possible S-boxes to remove from the computation, first, we need to formulate the input and output difference of each active S-box under the conditions of the linear layer to fulfill the activity pattern, as in Example 78. Recall that the size of each word is s bits, the same as the S-box bit-size. For each active S-box, we assign two variables, one for its input and one for its output, and find the linear relations of variables through the linear layer by considering the activity pattern conditions. We obtain a set of linear equations for relations between the variables. We also consider one set for all active S-boxes and one set for all variables. Then, we need to find the S-boxes where either the input variable or the output one is independent of

all input/output variables of other remaining active S-boxes. If we found such an S-box, we remove the S-box from the set of active S-boxes. Also, if there are some linear equations involving the corresponding variable for other side of this active S-box, we simplify these equations in a *Gaussian* way to remove this variable. We repeat this procedure until there is no longer any partially independent S-box. If in the beginning there were n_S active S-boxes, the number of steps to reach the final set of active S-boxes cannot exceed $n_S \cdot (n_S + 1)/2$.

In this way, by removing some of the active S-boxes and variables, the proposed state-by-state method's complexity will be improved because removing independent variables may decrease the number of possible states in each round.

5.2.2 *Simple Word-by-Word Method*

In the previous method, we considered the entire S-box layer together and computed the differential transition through it, i. e., we compute the table of probabilities (i. e., T_i tables) state-by-state. An alternative technique to reduce the time complexity is to consider each S-box of the layer separately and update the probability tables by each active S-box.

Assume that $\Delta X_i = (\Delta X_i[0], \ldots, \Delta X_i[m-1])$ and $\Delta Y_i = (\Delta Y_i[0], \ldots, \Delta Y_i[m-1])$, the states before and after the i-th S-box layer, and we have a table where the probability of the reduced i-round differential is saved for all possible ΔX_i. If we consider the entire S_i-layer together and compute the probability of the $(i+1)$-round differential, then we need $|A_i| \cdot |A_{i+1}|$ computations. But instead, we can separate the S_i layer differential transition into m smaller steps where the j-th step considers the j-th S-box transition. This means in step 0, we consider all possible differential transitions from $(\Delta X_i[0], \Delta X_i[1], \ldots, \Delta X_i[m-1])$ to $(\Delta Y_i[0], \Delta X_i[1], \ldots, \Delta X_i[m-1])$. Then in step 1, we consider all possible transitions from $(\Delta Y_i[0], \Delta X_i[1], \ldots, \Delta X_i[m-1])$ to $(\Delta Y_i[0], \Delta Y_i[1], \ldots, \Delta X_i[m-1])$ and so on. In the last step, all possible transitions from $(\Delta Y_i[0], \ldots, \Delta Y_i[m-2], \Delta X_i[m-1])$ to $(\Delta Y_i[0], \ldots, \Delta Y_i[m-2], \Delta Y_i[m-1])$ are considered. Note that here, we assumed that all S-boxes of the round are active. If an S-box is inactive, we can just go to the next step without updating the tables.

Example 83. Consider Example 82 which gives the simplified EDP $\left(\mathcal{U}_{\delta_0^*} \xrightarrow{\texttt{Enc}} \mathcal{V}_{\delta_4^*} \mid (\delta_0, \ldots, \delta_4)^*\right)$ after removing unnecessary active S-boxes for the example of Figure 5.1. We have 10 S-boxes to compute the probability. First, for each $\Delta X_2 = (0, \ldots, 0, x_1, x_1, 0, x_1)$, we initialize table $T_0[x_1]$ with value 1. For the first S-box, for each $(0, \ldots, 0, x_2, x_1, 0, x_1)$, we initialize $T_1[x_1, x_2]$ with zero and then for each $[x_1, x_2]$, we compute $p_{12} \cdot T_0[x_1]$ and add it to $T_1[x_1, x_2]$. For the second S-box, for each $(0, \ldots, 0, x_2, x_3, 0, x_1)$, we initialize $T_2[x_1, x_2, x_3]$ with zero and then for each $[x_1, x_2, x_3]$, we compute $p_{13} \cdot T_1[x_1, x_2]$ and add it to $T_2[x_1, x_2, x_3]$. For the third S-box, for each $(0, \ldots, 0, x_2, x_3, 0, x_4)$, we initialize $T_3[x_2, x_3, x_4]$ with zero and then for each $[x_1, x_2, x_3, x_4]$, we compute $p_{14} \cdot T_2[x_1, x_2, x_3]$ and add it to $T_3[x_2, x_3, x_4]$. We repeat computing the probability tables for the other S-boxes. Finally, after computing $T_{10}[x_5, x_6, x_9]$, we need to sum all the values of this table and

divide it by 16^3-1 to have the value of the $\mathrm{EDP}\left(\mathcal{U}_{\delta_0^*} \xrightarrow{\mathrm{Enc}} \mathcal{V}_{\delta_4^*} \mid (\delta_0,\ldots,\delta_4)^*\right)$. We summarize the computation of each step below.

1. $\forall x_1:\ T_0[x_1] \leftarrow 1$.
2. $\forall x_1,x_2:\ T_1[x_1,x_2] \leftarrow T_1[x_1,x_2] + T_0[x_1]\cdot p_{12}$.
3. $\forall x_1,x_2,x_3:\ T_2[x_1,x_2,x_3] \leftarrow T_2[x_1,x_2,x_3] + T_1[x_1,x_2]\cdot p_{13}$.
4. $\forall x_1,x_2,x_3,x_4:\ T_3[x_2,x_3,x_4] \leftarrow T_3[x_2,x_3,x_4] + T_2[x_1,x_2,x_3]\cdot p_{14}$.
5. $\forall x_2,x_3,x_4,x_5:$
   $T_4[x_2,x_3,x_4,x_5] \leftarrow T_4[x_2,x_3,x_4,x_5] + T_3[x_2,x_3,x_4]\cdot p_{45}$.
6. $\forall x_2,x_3,x_4,x_5,x_6:$
   $T_5[x_2,x_3,x_5,x_6] \leftarrow T_5[x_2,x_3,x_5,x_6] + T_4[x_2,x_3,x_4,x_5]\cdot p_{46}$.
7. $\forall x_2,x_3,x_5,x_6,x_9:$
   $T_6[x_2,x_3,x_5,x_6,x_9] \leftarrow T_6[x_2,x_3,x_5,x_6,x_9] + T_5[x_2,x_3,x_5,x_6]\cdot p_{29}$.
8. $\forall x_2,x_3,x_5,x_6,x_9:$
   $T_7[x_3,x_5,x_6,x_9] \leftarrow T_7[x_3,x_5,x_6,x_9] + T_6[x_2,x_3,x_5,x_6,x_9]\cdot p_{26}$.
9. $\forall x_3,x_5,x_6,x_9:$
   $T_8[x_3,x_5,x_6,x_9] \leftarrow T_8[x_3,x_5,x_6,x_9] + T_7[x_3,x_5,x_6,x_9]\cdot p_{39}$.
10. $\forall x_3,x_5,x_6,x_9:$
    $T_9[x_3,x_5,x_6,x_9] \leftarrow T_9[x_3,x_5,x_6,x_9] + T_8[x_3,x_5,x_6,x_9]\cdot p_{36}$.
11. $\forall x_3,x_5,x_6,x_9:$
    $T_{10}[x_5,x_6,x_9] \leftarrow T_{10}[x_5,x_6,x_9] + T_9[x_3,x_5,x_6,x_9]\cdot p_{35}$.
12. $p \leftarrow 0$ and $\forall x_3,x_5,x_6,x_9:\ p \leftarrow p + T_{10}[x_5,x_6,x_9]$.
    Then return $p\cdot 2^{-12}$.

Note that we assumed each table is already initialized to 0. It is also noteworthy that in each step, we need to keep only two consecutive tables. This computation needs about $3\cdot 16^5 + 5\cdot 16^4 \approx 2^{22}$ arithmetic operations and $2\cdot 16^5 = 2^{21}$ blocks of memory.

Updating the probability tables using this word-by-word method instead of state-by-state might increase the memory complexity, but decrease the computational time. For the given example, this improvement of time is not a good trade-off. In Section 5.3.1, using the simple word-by-word method, we can compute the probability while it is very expensive using the state-by-state method.

For a cipher whose linear layer is not AES-like (i. e., it is not a linear layer based on multiplications in the $\mathbb{F}_{2^s}$), using the word-by-word methods in an efficient way needs optimizing through the linear layer's specification, and this is can be an uneasy task. On the other hand, for a cipher with an AES-like linear layer, this method can be improved to reduce the computational complexity significantly.

5.2.3 *First Advanced Word-by-Word Method*

Assume that for a given activity pattern in a cipher with an AES-like linear layer, we already formulated the input and output difference of each active S-box under the conditions of the linear layer and removed the unnecessary S-boxes. Now we are left with some variables and S-boxes which can not be further simplified. We define an undirected graph for the probability equation, where each vertex determines one of the remaining variables and each edge determines if there is a relation between the corresponding two variables either through the conditions of the linear layer or because both variables are in an input/output equation for one of the remaining S-boxes.

To compute the EDP of a truncated characteristic, we need to multiply all remaining S-boxes' probability for each possible value of all remaining variables and then sum them. If there is an S-box probability that does not depend on all variables, one can separate the summing into several steps to reduce the computation cost. However, note that when we sum over variable x_i, the value of all neighbor variables of x_i in the corresponding graph must be considered. When we sum over a variable, it means that this variable does not appear in the next steps of the computation, which may help reduce the costs of the next steps. For now, assume that we are doing the sum over variables by their lexicographic order.

To sum over the first variable, for each possible value of its neighbor variables, we compute the sum of multiplication of S-box probabilities corresponding to this variable and save it in table T_0 indexed by the value of neighbor variables. We denote this index with I_0 and the set of corresponding variables with $\mathcal{I}_0$; i. e., I_0 determines the value of $\mathcal{I}_0$. Then, we can remove this variable and the involved S-boxes or linear conditions from the set of remaining variables or S-boxes. We can also update the graph by removing the first variable and all its connecting edges.

Then, to update the probability table and sum over the second variable, we need to consider its neighbor variables. But since we already computed the probabilities involving the first variable and saved them in table T_0, we need to consider the variables in $\mathcal{I}_0$. This means to sum over the second variable, we need to consider all neighbor variables of the second variable in the updated graph together with the variables of $\mathcal{I}_0$ except the second variable itself. We denote this set with $\mathcal{I}_1$ and the corresponding value for all its variables with I_1. Then, for each value of I_1 and the second variable, we compute the sum of multiplication of remaining S-box probabilities corresponding to the second variable together multiplied by the value of index $T_0[I_0]$ and save it in the index I_1 of table T_1. Then, we can again remove this variable and the involved S-boxes or linear conditions from the corresponding sets and update the graph. We continue these steps until the last variable in the set of remaining variables.

Example 84. We illustrate the corresponding graphs for the EDP$\left(\mathcal{U}_{\delta_0} \xrightarrow{\texttt{Enc}} \mathcal{V}_{\delta_4} \mid (\delta_0, \ldots, \delta_4)\right)$ and the EDP$\left(\mathcal{U}_{\delta_0^*} \xrightarrow{\texttt{Enc}} \mathcal{V}_{\delta_4^*} \mid (\delta_0, \ldots, \delta_4)^*\right)$ of the previous example in Figure 5.2.

For instance, in Figure 5.2b, we first sum over x_1, then x_2, x_3, x_4, x_5, x_6 and at the end x_9. This way, we can rewrite the sum of equation from

Example 82 as

$$\sum_{x_5,x_6,x_9}\sum_{x_4}\Big(p_{45}\cdot p_{46}\cdot\sum_{x_3}\big(p_{39}\cdot p_{36}\cdot p_{35}\cdot\sum_{x_2}\big(p_{29}\cdot p_{26}\cdot\sum_{x_1}(p_{12}\cdot p_{13}\cdot p_{14})\big)\big)\Big)$$

For the same graph, if we want to sum over x_1, we must consider the values for x_2, x_3, and x_4. Therefore, we compute $\sum_{x_1}(p_{12}\cdot p_{13}\cdot p_{14})$ and save it in $T_0[x_2,x_3,x_4]$ with $\mathcal{I}_0=[x_2,x_3,x_4]$. In the next step $\mathcal{I}_1=[x_3,x_4,x_6,x_9]$ and we compute $\sum_{x_2}(p_{29}\cdot p_{26}\cdot T_0[x_2,x_3,x_4])$. We summarize each step of the computation for this example in the following and the first steps of updating the corresponding graph in Figure 5.3.

1. $\mathcal{I}_0=[x_2,x_3,x_4]$, $\forall I_0\ \ T_0[I_0]\leftarrow\sum_{x_1}(p_{12}\cdot p_{13}\cdot p_{14})$).
2. $\mathcal{I}_1=[x_3,x_4,x_6,x_9]$, $\forall I_1\ \ T_1[I_1]\leftarrow\sum_{x_2}(p_{29}\cdot p_{26}\cdot T_0[I_0])$.
3. $\mathcal{I}_2=[x_4,x_5,x_6,x_9]$, $\forall I_2\ \ T_2[I_2]\leftarrow\sum_{x_3}(p_{39}\cdot p_{36}\cdot p_{35}\cdot T_1[I_1])$.
4. $\mathcal{I}_3=[x_5,x_6,x_9]$, $\forall I_3\ \ T_3[I_3]\leftarrow\sum_{x_4}(p_{45}\cdot p_{46}\cdot T_2[I_2])$.
5. $\mathcal{I}_4=[x_6,x_9]$, $\forall I_4\ \ T_4[I_4]\leftarrow\sum_{x_5}T_3[I_3]$.
6. $\mathcal{I}_5=[x_9]$, $\forall I_5\ \ T_5[I_5]\leftarrow\sum_{x_6}T_4[I_4]$.
7. $\mathcal{I}_6=\emptyset$, return $2^{-12}\cdot\sum_{x_9}T_5[I_5]$.

Complexity of the Method

Each step of this method needs $2^{s\cdot(|\mathcal{I}_i|+1)}$ computations together with $2^{s\cdot|\mathcal{I}_i|}$ blocks of memory. But since we need to keep only two consecutive tables, the general memory complexity is equal to $2^{s\cdot\max_i|\mathcal{I}_i|}$ while its computational cost is $2^s\cdot\sum_i 2^{s\cdot|\mathcal{I}_i|}$.

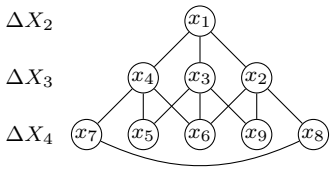
Thus, to reduce the complexity, it is important to keep the size of $\mathcal{I}_i$ as minimal as possible, and this is highly dependent on the order of variables that we are summing over. When the number of variables is low, then we can check for all $n_v!$ possible orders, where n_v denotes the number of remaining variables. Then we can choose the one which gives the minimum complexity. But if we are left with more than 15 variables, to find an optimal order, we need to check at least 2^{40} orders, which is expensive.

Example 85. For the previous example, x_5,x_1,x_4,x_6,x_2,x_3 and x_9 is one of the orders with minimum complexity, illustrated in Figure 5.4. Then, $\max_i|\mathcal{I}_i|=3$ and the corresponding computation needs 2^{13} operations together with 2^{13} memory blocks.

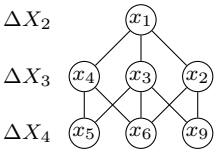
In the following, we explain our solution to find an order for variables with a low (though not necessarily minimal) cost.

Order of Variables in the First Advanced Method

We use a greedy method to choose the order. The first variable is chosen among those with the minimum number of neighbors. Then, for each choice of the first variable, we go to the next step. In the second step, for each remaining variable, we compute the corresponding $\mathcal{I}_1$ and choose the second

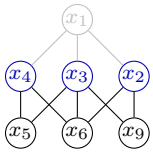


(a): $\mathrm{EDP}\big(\mathcal{U}_{\delta_0}\xrightarrow{\mathrm{Enc}}\mathcal{V}_{\delta_4}\,|\,(\delta_0,\ldots,\delta_4)\big)$

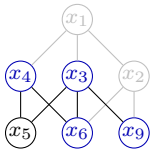


(b): $\mathrm{EDP}\big(\mathcal{U}_{\delta_0^*}\xrightarrow{\mathrm{Enc}}\mathcal{V}_{\delta_4^*}\,|\,(\delta_0,\ldots,\delta_4)^*\big)$

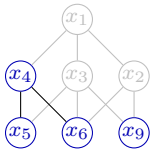
FIGURE 5.2: Graph Representation of the Variable Relations in Example 84.



(a): Resolve x_1

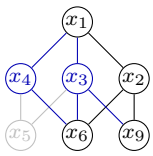


(b): Resolve x_2

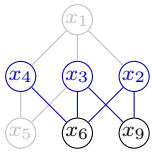


(c): Resolve x_3

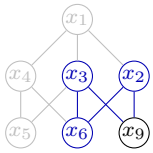
FIGURE 5.3: First Advanced Word-by-Word Method for Example 84.



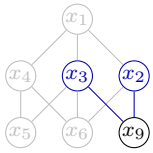
(a): Resolve x_5: $\mathcal{I}_0=[x_3,x_4]$



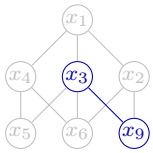
(b): Resolve x_1: $\mathcal{I}_1=[x_2,x_3,x_4]$



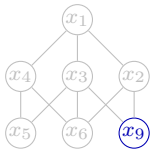
(c): Resolve x_4: $\mathcal{I}_2=[x_2,x_3,x_6]$



(d): Resolve x_6: $\mathcal{I}_3=[x_2,x_3]$



(e): Resolve x_2: $\mathcal{I}_4=[x_3,x_9]$



(f): Resolve x_3: $\mathcal{I}_5=[x_9]$

FIGURE 5.4: First Advanced Word-by-Word Method for Example 84 with an Optimum Order for the Variables.

one from the ones with the minimum $\mathcal{I}_1$. Then, for each choice of the second variable, we go to the next step. We continue until the last variable is chosen and the order is complete. Then, we compute the complexity and keep the order if it is the one with minimum complexity between the orders we already checked.

This approach does not guarantee that the final result is the general minimum through all $n_v!$ possible choices, but it is comparably fast. In particular, if in a step the minimum cost s higher than an upper bound, we can cut the branch and return to the previous step. In this way, we can save time and just go through the orders whose complexity is less than the upper bound.

5.2.4 *Second Advanced Word-by-Word Method*

The previous method of computing the EDP for AES-like ciphers updates the probability tables only using the previous one. In the new method that we are explaining now, we compute each probability table using multiple of the previously computed tables. Same as the previous method, we assume that the input and output difference equations of each active S-box are given, and all the unnecessary S-boxes and variables are removed. We define the corresponding graph as before.

Now, for each remaining active S-box, we assign a table that saves the transition probability of the S-box. That means for each possible value for all variables involving in the input and output difference equations of the S-box, we compute the input and output difference values and take the corresponding probability from DDT. That means, to this point, we need n_s tables, namely $T_0,\dots,T_{n_s-1}$ which each just determines transition probability over one S-box. Similarly, we denote the corresponding index in the table T_i with I_i and the set of corresponding variables with $\mathcal{I}_i$.

In the second part, we use the corresponding graph and update it step by step with a greedy approach similar to the first advanced word-by-word method. We take the first variable with minimum degree in the graph and do the sum over this variable for each possible value of its neighbor variables. To do this, we use all previous tables $T_0,\dots,T_{n_s-1}$ which this variable is involved. For each value of neighbor variables and this variable itself, we get the corresponding values from those tables and multiply them together and then add this value to a new table T_{n_s} indexed by the value of all neighbor variables of the chosen variable, i. e., $\mathcal{I}_{n_s}$. Then we remove all the tables that we used for computing T_{n_s} and update the graph so that we remove the chosen variable and all the connecting edges. But we connect all the neighbor variables of the removed variable to each other. Note that this updating is different from the previous one. We continue this step until the point that the graph is an empty one. Algorithm 6 explains all the details of this method in pseudo-code.

Example 86. For instance, in Example 84 with the given graph in Figure 5.2b, variable x_5 is chosen as the first one and when we update the graph after removing its edges, we add an extra edge between x_3 and x_4. We summarize each step of this method for this example in the following.

Algorithm 6 Computing the EDP of Truncated Differential in Ciphers with Word-wise Linear Layer using the Second Advanced Word-by-Word Method

Input: $n_s, n_v, IE[n_s][n_v], OE[n_s][n_v]$

Output: $\mathrm{EDP}\left(\mathcal{U}_{\delta_0} \xrightarrow{\mathrm{Enc}} \mathcal{V}_{\delta_{r-1}} \mid (\delta_0, \ldots, \delta_{r-1})\right)$

Computing the EDP of the Truncated Characteristic

procedure COMPUTEEDP
  $IT \leftarrow$ DETERMINEINDEXINGTABLE(IE, OE)
  $G \leftarrow$ COMPUTEGRAPHMATRIX(IT)
  $d_{min} \leftarrow n_v$
  for $0 \le i < n_s$ **do**
    $d \leftarrow$ GETDEGREE$(G[i])$
    if $d_{min} > d$ **then**
      $d_{min} \leftarrow d$
    for $0 \le I < 2^{m \cdot d}$ **do**
      $V \leftarrow$ GETVARIABLESVALUE$(I, IT[i])$
      $x \leftarrow$ GETXORVALUE$(V, IE[i])$
      $y \leftarrow$ GETXORVALUE$(V, OE[i])$
      $T[i][I] \leftarrow DDT[x][y] \cdot 2^{-s}$
  $k \leftarrow n_s$
  $r[\,] \leftarrow \{1\}^{n_s+n_v}$
  for $0 \le i < n_v$ **do**
    $d \leftarrow$ GETDEGREE$(G[i])$
    if $d = d_{min}$ **then**
      $d_{min} \leftarrow d$
      $IT[k] \leftarrow G[i]$
      $r'[\,] \leftarrow \{0\}^k$
      for $0 \le j < k$ **do**
        if $r[j] \& IT[i][j]$ **then**
          $r'[j] \leftarrow 1$
      for $0 \le I < 2^{m \cdot d}$ **do**
        $T[k][I] \leftarrow 0$
        $V \leftarrow$ GETVARIABLESVALUE$(I, IT[k])$
        for $0 \le x < 2^m$ **do**
          $V[i] \leftarrow x$
          $p \leftarrow 1$
          for $0 \le j < k$ **do**
            if $r'[j]$ **then**
              $J \leftarrow$ GETCOMPLETEINDEX$(V, IT[j])$
              $p \leftarrow p \times T[j][J]$
          $T[k][I] \leftarrow T[k][I] + p$
      $(G, d_{min}) \leftarrow$ UPDATEGRAPHMATRIX(G, i)
      for $0 \le j < k$ **do**
        if $r'[j]$ **then**
          $r[j] \leftarrow 0$
      $i \leftarrow 0$
      $k \leftarrow k + 1$
      if $k = (n_s + n_v)$ **then**
        break;

Algorithm 6 (continued)

Determining Which Variables Are Involved in Each Probability Table

```
44 procedure DETERMINEINDEXINGTABLE(IE[n_s][n_v], OE[n_s][n_v])
45     IT[][] ← {0}^{(n_s+n_v)×n_v}
46     for 0 ≤ i < n_s do
47         for 0 ≤ j < n_v do
48             IT[i][j] ← IE[i][j] ∨ OE[i][j]
49     return IT
```

Computing The Connectivity Matrix of the Graph

```
50 procedure COMPUTEGRAPHMATRIX(IT[n_s][n_v])
51     G[][] ← {0}^{n_v×n_v}
52     for 0 ≤ i < n_s do
53         for 0 ≤ j < n_v do
54             if IT[i][j] then
55                 G[i][j] ← 1
56                 G[j][i] ← 1
57     return G
```

Computes Degree of a Vertex in the Graph

```
58 procedure GETDEGREE(A[n_v])
59     d ← 0
60     for 0 ≤ i < n_v do
61         d ← d + A[i]
62     return d
```

Get Variables Value From the Complete Index

```
63 procedure GETVARIABLESVALUE(X, A[n_v])
64     V[] ← {0}^{n_v}
65     for 0 ≤ i < n_v do
66         if A[i] then
67             V[i] ← last word of X
68             X get shifted to right by one word
69     return V
```

Get Complete Index From Variables Value

```
70 procedure GETCOMPLETEINDEX(V[n_v], A[n_v])
71     X ← 0
72     for 0 ≤ i < n_v do
73         if A[i] then
74             X get shifted to left by one word
75             X ← X ⊕ V[i]
76     return X
```

Get Xor Value From Variables Value

```
77 procedure GETXORVALUE(V[n_v], A[n_v])
78     X ← 0
79     for 0 ≤ i < n_v do
80         if A[i] then
81             X ← X ⊕ V[i]
82     return X
```

Algorithm 6 (continued)

Updating the Graph Matrix by Removing the Given Variable

```
83  procedure UpdateGraphMatrix(G, v)
84      for 0 ≤ i < n_v − 1 do
85          if G[v][i] then
86              for i + 1 ≤ j < n_v do
87                  if G[v][j] then
88                      G[i][j] ← 1
89                      G[j][i] ← 1
90      for 0 ≤ i < n_v do
91          G[i][v] ← 0
92          G[v][i] ← 0
93      d_min ← n_v
94      for 0 ≤ i < n_v do
95          d ← 0
96          for 0 ≤ j < n_v do
97              if G[i][j] then
98                  d ← d + 1
99          if 0 < d < d_min then
100             d_min ← d
101     return G, d_min
```

0. $\forall x_1, x_2 :\ T_0[x_1, x_2] \leftarrow p_{12}, \quad \forall x_1, x_3 :\ T_1[x_1, x_3] \leftarrow p_{13},$
   $\forall x_1, x_4 :\ T_2[x_1, x_4] \leftarrow p_{14}, \quad \forall x_4, x_5 :\ T_3[x_4, x_5] \leftarrow p_{45},$
   $\forall x_4, x_6 :\ T_4[x_4, x_6] \leftarrow p_{46}, \quad \forall x_2, x_9 :\ T_5[x_2, x_9] \leftarrow p_{29},$
   $\forall x_2, x_6 :\ T_6[x_2, x_6] \leftarrow p_{26}, \quad \forall x_3, x_9 :\ T_7[x_3, x_9] \leftarrow p_{39},$
   $\forall x_3, x_6 :\ T_8[x_3, x_6] \leftarrow p_{36}, \quad \forall x_3, x_5 :\ T_9[x_3, x_5] \leftarrow p_{35}.$
1. $\forall x_3, x_4 :\ T_{10}[x_3, x_4] \leftarrow \sum_{x_5}(T_3[x_4, x_5] \cdot T_9[x_3, x_5])$.
2. $\forall x_2, x_3 :\ T_{11}[x_2, x_3] \leftarrow \sum_{x_9}(T_5[x_2, x_9] \cdot T_7[x_3, x_9])$.
3. $\forall x_2, x_3, x_4 :\ T_{12}[x_2, x_3, x_4] \leftarrow \sum_{x_1}(T_0[x_1, x_2] \cdot T_1[x_1, x_3] \cdot T_2[x_1, x_4])$.
4. $\forall x_3, x_4, x_6 :\ T_{13}[x_3, x_4, x_6] \leftarrow \sum_{x_2}(T_6[x_2, x_6] \cdot T_{11}[x_2, x_3] \cdot T_{12}[x_2, x_3, x_4])$.
5. $\forall x_4, x_6 :\ T_{14}[x_4, x_6] \leftarrow \sum_{x_3}(T_8[x_3, x_6] \cdot T_{10}[x_3, x_4] \cdot T_{13}[x_3, x_4, x_6])$.
6. $\forall x_6 :\ T_{15}[x_6] \leftarrow \sum_{x_4}(T_4[x_4, x_6] \cdot T_{14}[x_4, x_6])$.
7. return $2^{-12} \cdot \sum_{x_6} T_{15}[x_6]$.

Complexity of the Method

Each step in the second part of the method needs $2^{s\cdot(|\mathcal{I}_i|+1)}$ computations together with $2^{s\cdot|\mathcal{I}_i|}$ blocks of memory where $|\mathcal{I}_i|$ is the same as the degree of the corresponding vertex in the updated graph for the variable we are summing over it.

Even though we assign a table for each active S-box, but one can use a complicated code to compute these tables on the fly without using memory to allocate. Note that this technique do not have extra computational cost.

Therefore, the general memory complexity is equal to $\sum_{i=n_s}^{n_s+n_v-1} 2^{s\cdot|\mathcal{I}_i|}$ and its computational cost is $2^s \cdot \sum_{i=0}^{n_s+n_v-1} 2^{s\cdot|\mathcal{I}_i|}$.

Similar to the first method, to reduce the complexity of this method, it is important to keep the size of $\mathcal{I}_i$ as minimal as possible, and this is highly dependent on the order of variables that we are summing over. Again, for a lower number of variables, we can check for all $n_v!$ possible orders, but for a higher number of variables, we suggest using the above-mentioned greedy method.

5.2.5 *Upper-Bound for the EDP of Differentials within a Truncated Characteristic*

Computing the $\mathrm{EDP}\left(\mathcal{U}_{\delta_0} \xrightarrow{\mathtt{Enc}} \mathcal{V}_{\delta_{r-1}} \mid (\delta_0, \ldots, \delta_{r-1})\right)$ has the same or less computational complexity than the one for the $\mathrm{EDP}\left(\alpha_0 \xrightarrow{\mathtt{Enc}} \alpha_r \mid (\delta_0, \ldots, \delta_{r-1})\right)$ with $\alpha_0 \in \mathcal{U}_{\delta_0}$ and $\alpha_r \in \mathcal{V}_{\delta_{r-1}}$. To find differentials of a given activity pattern with the maximum EDP, we need to search through all differentials within the truncated characteristic. This means complexity of finding such differentials, increase the complexity of computing the EDP with factor of $|\mathcal{U}_{\delta_0}| \cdot |\mathcal{V}_{\delta_{r-1}}| = (2^s - 1)^{\mathrm{hw}(\delta_0)+\mathrm{hw}(\delta_{r-1})}$.

Here, we provide an inequality between the EDP of a truncated characteristic and the EDP of differentials within the same activity pattern. That is by having value of the $\mathrm{EDP}\left(\mathcal{U}_{\delta_0} \xrightarrow{\mathtt{Enc}} \mathcal{V}_{\delta_{r-1}} \mid (\delta_0, \ldots, \delta_{r-1})\right)$, this inequality gives an upper-bound for

$$\max_{\substack{\alpha_0 \in \mathcal{U}_{\delta_0} \\ \alpha_r \in \mathcal{V}_{\delta_{r-1}}}} \mathrm{EDP}\left(\alpha_0 \xrightarrow{\mathtt{Enc}} \alpha_r \mid (\delta_0, \ldots, \delta_{r-1})\right).$$

Using Theorem 87, once we computed the EDP of the truncated characteristic, we have an upper-bound for the EDP of underlying differentials. In case that the value for this upper-bound is smaller than 2^{-n}, we can conclude that there is no differential within this activity pattern that is useful as a distinguisher.

Theorem 87. *For a given r-round activity pattern $(\delta_0, \ldots, \delta_{r-1})$, we have*

$$\max_{\substack{\alpha_0 \in \mathcal{U}_{\delta_0} \\ \alpha_r \in \mathcal{V}_{\delta_{r-1}}}} \mathrm{EDP}\left(\alpha_0 \xrightarrow{\mathtt{Enc}} \alpha_r \mid (\delta_0, \ldots, \delta_{r-1})\right) \leq$$

$$\left(\mathrm{uni}(S) \cdot 2^{-s}\right)^{\mathrm{hw}(\delta_0)+\mathrm{hw}(\delta_{r-1})} \cdot \sum_{\substack{\alpha_1, \ldots, \alpha_{r-1} \\ \forall i\ 0<i<r, \\ \alpha_i \in A_i}} \prod_{i=1}^{r-2} \mathrm{Prob}\left(\alpha_i \xrightarrow{\mathtt{S}_i} \mathtt{L}_i^{-1}(\alpha_{i+1})\right)$$

which is the same as

$$\max_{\substack{\alpha_0 \in \mathcal{U}_{\delta_0} \\ \alpha_r \in \mathcal{V}_{\delta_{r-1}}}} \mathrm{EDP}\left(\alpha_0 \xrightarrow{\mathtt{Enc}} \alpha_r \mid (\delta_0, \ldots, \delta_{r-1})\right) \leq$$

$$\left(\mathrm{uni}(S) \cdot 2^{-s}\right)^{\mathrm{hw}(\delta_0)+\mathrm{hw}(\delta_{r-1})} \cdot (2^s - 1)^{\mathrm{hw}(\delta_0)} \cdot \mathrm{EDP}\left(\mathcal{U}_{\delta_0} \xrightarrow{\mathtt{Enc}} \mathcal{V}_{\delta_{r-1}} \mid (\delta_0, \ldots, \delta_{r-1})\right).$$

Note that it is considered that the differential uniformity of all active S-boxes in the first and the last layer are all the same and equal to $\mathrm{uni}(S)$.[2]

[2] Note that this is considered only for simplicity of the inequality and it does not add any restrictions to the theorem.

Proof. Since

$$\prod_{i=0}^{r-1}\mathrm{Prob}\left(\alpha_i \xrightarrow{\mathrm{S}_i} \mathrm{L}_i^{-1}(\alpha_{i+1})\right) \leq \prod_{i=1}^{r-2}\mathrm{Prob}\left(\alpha_i \xrightarrow{\mathrm{S}_i} \mathrm{L}_i^{-1}(\alpha_{i+1})\right)\cdot \max_{\alpha_1}\mathrm{Prob}\left(\alpha_0 \xrightarrow{\mathrm{S}_0} \mathrm{L}_0^{-1}(\alpha_1)\right)\cdot \max_{\alpha_{r-1}}\mathrm{Prob}\left(\alpha_{r-1} \xrightarrow{\mathrm{S}_{r-1}} \mathrm{L}_{r-1}^{-1}(\alpha_r)\right),$$

we have

$$\sum_{\substack{\alpha_1,\ldots,\alpha_{r-1}\\ \forall i\ 0<i<r,\\ \alpha_i\in A_i}}\prod_{i=0}^{r-1}\mathrm{Prob}\left(\alpha_i \xrightarrow{\mathrm{S}_i} \mathrm{L}_i^{-1}(\alpha_{i+1})\right) \leq \sum_{\substack{\alpha_1,\ldots,\alpha_{r-1}\\ \forall i\ 0<i<r,\\ \alpha_i\in A_i}}\Big(\prod_{i=1}^{r-2}\mathrm{Prob}\left(\alpha_i \xrightarrow{\mathrm{S}_i} \mathrm{L}_i^{-1}(\alpha_{i+1})\right)\cdot \max_{\alpha'_1\in A_1}\mathrm{Prob}\left(\alpha_0 \xrightarrow{\mathrm{S}_0} \mathrm{L}_0^{-1}(\alpha'_1)\right)\cdot \max_{\alpha'_{r-1}\in A_{r-1}}\mathrm{Prob}\left(\alpha'_{r-1} \xrightarrow{\mathrm{S}_{r-1}} \mathrm{L}_{r-1}^{-1}(\alpha_r)\right)\Big).$$

Therefore

$$\mathrm{EDP}\left(\alpha_0 \xrightarrow{\mathtt{Enc}} \alpha_r \mid (\delta_0,\ldots,\delta_{r-1})\right) \leq \Big(\sum_{\substack{\alpha_1,\ldots,\alpha_{r-1}\\ \forall i\ 0<i<r,\\ \alpha_i\in A_i}}\prod_{i=1}^{r-2}\mathrm{Prob}\left(\alpha_i \xrightarrow{\mathrm{S}_i} \mathrm{L}_i^{-1}(\alpha_{i+1})\right)\Big)\cdot \max_{\alpha_1\in A_1}\mathrm{Prob}\left(\alpha_0 \xrightarrow{\mathrm{S}_0} \mathrm{L}_0^{-1}(\alpha_1)\right)\cdot \max_{\alpha_{r-1}\in A_{r-1}}\mathrm{Prob}\left(\alpha_{r-1} \xrightarrow{\mathrm{S}_{r-1}} \mathrm{L}_{r-1}^{-1}(\alpha_r)\right).$$

We know that for any $\alpha_0 \in \mathcal{U}_{\delta_0}$ and any $\alpha_r \in \mathcal{V}_{\delta_{r-1}}$

$$\max_{\alpha_1\in A_1}\mathrm{Prob}\left(\alpha_0 \xrightarrow{\mathrm{S}_0} \mathrm{L}_0^{-1}(\alpha_1)\right) \leq \left(uni(S)\cdot 2^{-s}\right)^{\mathrm{hw}(\delta_0)},$$
$$\max_{\alpha_{r-1}\in A_{r-1}}\mathrm{Prob}\left(\alpha_{r-1} \xrightarrow{\mathrm{S}_{r-1}} \mathrm{L}_{r-1}^{-1}(\alpha_r)\right) \leq \left(uni(S)\cdot 2^{-s}\right)^{\mathrm{hw}(\delta_{r-1})}.$$

That means for any $\alpha_0 \in \mathcal{U}_{\delta_0}$ and any $\alpha_r \in \mathcal{V}_{\delta_{r-1}}$, we have

$$\mathrm{EDP}\left(\alpha_0 \xrightarrow{\mathtt{Enc}} \alpha_r \mid (\delta_0,\ldots,\delta_{r-1})\right) \leq \left(uni(S)\cdot 2^{-s}\right)^{\mathrm{hw}(\delta_0)+\mathrm{hw}(\delta_{r-1})}\cdot \Big(\sum_{\substack{\alpha_1,\ldots,\alpha_{r-1}\\ \forall i\ 0<i<r,\\ \alpha_i\in A_i}}\prod_{i=1}^{r-2}\mathrm{Prob}\left(\alpha_i \xrightarrow{\mathrm{S}_i} \mathrm{L}_i^{-1}(\alpha_{i+1})\right)\Big)$$

and this proves the theorem. □

5.3 RESULTS ON SOME SPN BLOCK CIPHERS

In this section, we report on the results of applying our methods to CRAFT, MIDORI, SKINNY, KLEIN, and PRINCE.

5.3.1 KLEIN

KLEIN is a 64-bit block cipher designed by Gong, Nikova, and Law [GNL11] targeted for wireless sensors and RFID tags. It features a 4-bit involutive S-box, a simple byte-wise rotation, and an application of the AES MixColumns step (covering two 4-bit columns at once).

[GNL11] Gong, Nikova, and Law "KLEIN: A New Family of Lightweight Block Ciphers"

Lallemand and Naya-Plasencia [LN15] presented a truncated differential cryptanalysis of KLEIN which is based on four different truncated differential

[LN15] Lallemand and Naya-Plasencia "Cryptanalysis of KLEIN"

TABLE 5.2: Comparing Our Results with the Ones Given in [LN15] for KLEIN Truncated Differentials. Note that instead of the EDP p, we show $-\log_2 p$.

r	4	5	6	7	8	9	10	11	12	13	14	
Case I	21.5	27.5	33.5	39.5	45.5	51.5	57.5	63.5	69.5	75.5	81.5	[LN15]
	21.42	27.75	33.79	39.79	45.79	51.79	57.79	63.79	69.79	75.79	81.79	Ours
Case II	11.5	17.5	23.5	29.5	35.5	41.5	47.5	53.5	59.5	65.5	71.5	[LN15]
	12.04	18.05	24.05	30.05	36.05	42.05	48.05	54.05	60.05	66.05	72.05	Ours
Case III	18	24	30	36	42	48	54	60	66	72	78	[LN15]
	18	24	30	36	42	48	54	60	66	72	78	Ours
Case VI	13	19	25	31	37	43	49	55	61	67	73	[LN15]
	12.9	18.88	24.88	30.88	36.88	42.88	48.88	54.88	60.88	66.88	72.88	Ours

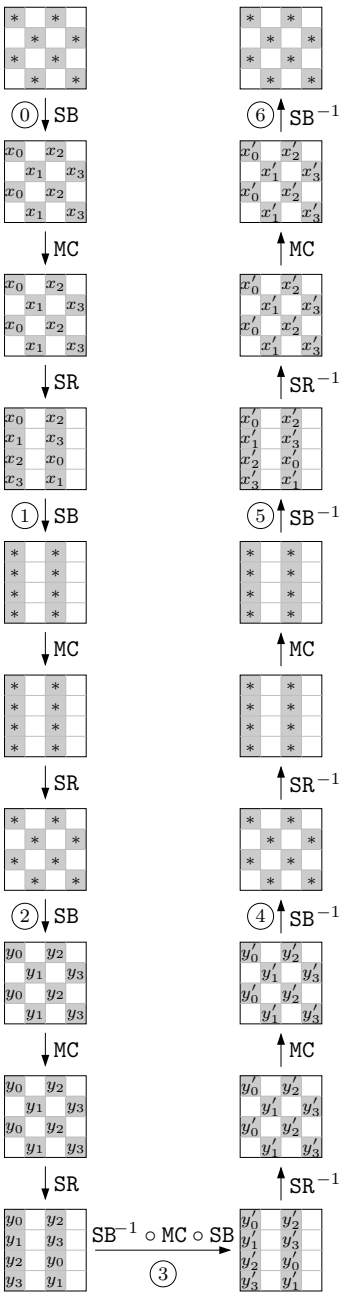


FIGURE 5.5: Truncated Differential Characteristic for PRINCE.

characteristics, entitled case I – IV. The probability of the r-round (with $r > 3$) truncated differential characteristic for each case is estimated as $2^{-6r+2.5}$, $2^{-6r+12.5}$, 2^{-6r+6} and 2^{-6r+11}, resp.

We used our simple word-by-word method to compute the exact probability of these truncated differential characteristics up to 15 rounds. The results are given in the Table 5.2. Interestingly, the difference between the estimated value in [LN15] and our computed value is rather small. These differences in logarithmic scale are +0.29, +0.55, +0.00 and −0.12, resp.

In all the cases of these truncated differential characteristics, each round (after the third round) includes 2^{26} possible states that if we use the state-by-state algorithm then computation of each round needs 2^{52} computations. But if we use the word-by-word algorithm and separate each round to 8 steps for each active S-box, then one round needs about 2^{33} computations. Each round of our programming takes 6 minutes and it needs to save 2^{33} 64-bit blocks, i. e., 64 GB.

5.3.2 PRINCE

Figure 5.5 illustrates an 8-round PRINCE truncated differential characteristic, where $*$ denotes any value in $\mathbb{F}_2^4$, $x_0, x_3, y_0, y_3, x'_0, x'_3, y'_0, y'_3 \in \{0, 2, 8, a\}$ and $x_1, x_2, y_1, y_2, x'_1, x'_2, y'_1, y'_2 \in \{0, 1, 4, 5\}$. Using Moriai's *et al.* method, the probability of the linear layer for rounds indexed as $0, 2, 3$ and 5 is estimated as 2^{-24} and for the rounds indexed as 1 and 4, it is 1. In total, the approximated probability using this method is 2^{-96}. Similar to the one in Figure 5.5, there are seven other truncated differential characteristics with the same input and output activity patterns and the same approximated probability, 2^{-96}.

Using our state-by-state method to compute the exact probability, we derive that the correct value is either 2^{-83} or 2^{-84} (four times the first case and four times the later one), which is significantly higher. Note that due to the removable S-box technique, the computation of first and last rounds are for free. Then, the simple state-by-state method needs $2^8 \cdot 2^{32}$ computation per round. One can compute the corresponding probability for two consecutive rounds for each of the $2^{8\cdot 2}$ different possible states and save it in a large DDT. This means for each of the 256 possible $X = [x_0, x_1, x_2, x_3]$ and 256 possible $Y = [y_0, y_1, y_2, y_3]$, we compute the probability of two corresponding rounds and in the same way for each $Y = [y_0, y_1, y_2, y_3]$ and $Y' = [y'_0, y'_1, y'_2, y'_3]$.[3] Each such a table needs $2^8 \cdot 2^{16} = 2^{24}$ computation. Thus, using the state-by-state technique will need $3 \cdot 2^{2\cdot 8}$ computations, less

[3] To compute the middle round difference transition we must use *Super S-boxes* that are 16-bit S-boxes

than the previous step.

We like to mention that in [Can+15], the authors introduced several 6-round differentials which are based on activity patterns with four active S-boxes per round. Our truncated characteristics cover all these differentials by removing the first and the last S-box layers. The highest differential probability that they could compute is $3 \cdot 2^{-58} \approx 2^{-56.42}$ while the second highest one is $13 \cdot 2^{-61} \approx 2^{-57.30}$. For each 6-round differentials covered by our truncated differential characteristic, we compute the corresponding probability and find out that the two highest ones are about $2^{-55.83}$ and $2^{-57.25}$ for the same differentials as the ones in [Can+15]. Besides, beyond the differentials in [Can+15], we find other ones with a probability significantly higher than 2^{-64}.

We like to mention that one can probably use this truncated differential characteristic to attack 10-round PRINCE by adding one round before and one round after the truncated differential characteristic. The key recovery part and the plaintext/ciphertext differences are the same as the one in [Can+15], but the probability for the whole 10-round of our characteristic is $2^{-71.09}$ which is much higher than theirs ($2^{-91.4}$). As a result, the attack using this approach should be more efficient than previously. Note that in [Can+15], they used 8 characteristics together, but for ours, there is only one other truncated characteristic with the same probability and input/output pattern.

5.3.3 CRAFT, MIDORI *and* SKINNY

We take MIDORI-64 and SKINNY-64 together with CRAFT as our targets to apply our methods. Since all these three ciphers are similar and follow an AES-like structure together with binary multiplication linear layer, in our implementations, we only need to change the corresponding linear matrix and the DDT or LAT.

Truncated Differential Characteristics

First, for each cipher, we identify several characteristics with (close to) the minimum number of active S-boxes for different number of rounds. We denote the number of active S-boxes by n_s. Then, we compute the EDP of each truncated characteristic together with its EDD. We summarized the results in Table 5.3 by showing the maximum and the minimum EDP p and EDD q for the characteristics with the same number of rounds and active S-boxes. Note that instead of the probability p or q, we show $-\log_2 p$ or $-\log_2 q$.

One interesting observation is that we found several 7-round characteristics for MIDORI whose distinguishability is larger than 2^{-64} while using Moriai's *et al*. method, there is no such characteristic.

For SKINNY and CRAFT, the maximum number of rounds where we found characteristics with a (near) minimum number of S-boxes and q greater than 2^{-64} is 10 and 12, resp. which the same maximum number of rounds using Moriai's *et al*. method.

TABLE 5.3: The Minimum and Maximum EDP p and EDD q of the Truncated Differential Characteristics for r Rounds with n_s Active S-boxes of the MIDORI-64, SKINNY-64, and CRAFT Block ciphers (1).

MIDORI-64

r	n_s	EDP min	EDP max	EDD min	EDD max
6	30	43.42	43.42	55.14	55.14
	32	54.28	35.76	73.86	57.89
	33	50.99	39.27	62.71	62.71
	34	61.89	30.49	80.85	58.64
7	35	54.11	46.29	65.83	56.07
	36	53.80	49.90	65.53	61.61
	37	59.26	43.63	72.52	61.69
8	38	60.27	52.17	64.17	63.89
	40	59.12	56.97	76.51	69.51
9	41	60.27	59.98	71.99	71.70

SKINNY-64

r	n_s	EDP min	EDP max	EDD min	EDD max
8	36	48.92	32.96	77.77	60.31
	37	50.02	29.23	70.61	56.64
	38	58.50	26.26	74.20	54.25
	39	58.61	22.73	74.69	53.71
	40	60.69	19.11	75.23	52.63
9	41	50.35	46.44	62.07	62.07
	42	51.11	43.29	66.58	62.82
	43	55.50	41.05	76.26	58.13
	44	61.64	41.90	77.38	61.52
	45	64.13	36.46	84.94	57.64
10	46	54.25	50.35	69.89	69.88
	47	61.64	49.92	73.36	65.94
	48	62.79	46.58	77.19	65.45
	49	65.27	48.91	78.50	57.68
	50	70.10	47.27	83.69	64.21
11	51	64.14	60.20	75.86	75.83
	52	68.02	53.78	81.17	65.50
	53	70.10	53.73	82.34	70.07
	54	73.18	50.58	88.83	66.32

CRAFT

r	n_s	EDP min	EDP max	EDD min	EDD max
9	32	35.30	35.30	47.02	47.02
	33	36.20	31.97	47.92	47.59
	34	38.95	31.79	58.48	47.14
	35	46.83	27.96	65.70	47.50
10	36	39.34	39.33	51.06	51.05
	37	40.28	36.01	52.00	50.30
	38	47.25	35.59	58.97	49.34
	39	50.92	31.98	62.64	51.52
11	40	43.37	43.37	55.09	55.09
	41	50.39	40.04	62.11	55.65
	42	51.22	39.50	63.05	55.46
	43	54.96	36.00	66.68	55.54
12	44	47.40	47.39	59.12	59.12
	45	47.97	44.06	59.69	59.67
	46	53.62	43.85	73.15	59.48
	47	59.00	40.02	73.99	59.56
13	48	51.42	51.42	63.15	63.15
	49	54.51	48.09	66.23	63.70
	50	56.68	47.87	68.40	63.50
	51	63.03	44.05	74.75	63.58
14	52	57.64	55.45	69.36	67.17
	53	58.60	52.11	70.32	67.72
	54	65.55	51.89	77.27	67.43
	55	67.71	48.07	79.43	67.60

TABLE 5.4: The Minimum and Maximum EDP p and EDD q of the Truncated Differential Characteristics for r Rounds with n_c Conditions of the MIDORI-64, SKINNY-64, and CRAFT Block Ciphers (2).

MIDORI-64

r	n_c	EDP min	EDP max	EDD min	EDD max
5	14	34.52	17.76	62.07	57.20

SKINNY-64

r	n_c	EDP min	EDP max	EDD min	EDD max
8	14	49.62	37.00	57.29	52.63
	15	56.50	16.78	67.46	57.29
9	15	56.33	40.19	61.43	57.63
10	15	55.98	49.88	59.88	57.68

CRAFT

r	n_c	EDP min	EDP max	EDD min	EDD max
10	12	36.53	36.53	48.25	48.25
	13	40.61	28.86	56.22	49.35
11	14	44.73	36.72	56.45	55.09
	15	55.79	32.86	68.03	55.58
12	14	44.66	44.66	56.38	56.38
	15	48.73	36.99	64.34	58.71
13	16	52.85	44.86	64.61	63.15

TABLE 5.5: The Maximum EDP of the Differentials and The Maximum ELP of the Linear Hulls for r Rounds with n_s Active S-boxes of the MIDORI-64, SKINNY-64, and CRAFT Block Ciphers.

MIDORI-64

r	n_c	max EDP	max ELP
6	30	44.04	59.68
	32	47.98	65.47
	33	51.75	66.87
	34	51.59	70.64
7	35	58.37	72.02
	36	59.96	–
	37	55.75	76.96
8	38	62.37	76.11
	40	65.19	78.59
9	41	67.13	80.17

SKINNY-64

r	n_c	max EDP	max ELP
8	32	–	59.87
	33	–	57.01
	34	–	64.70
	35	–	57.39
	36	49.50	58.93
	37	–	–
	38	52.24	–
	39	50.94	–
9	40	59.24	–
10	41	67.23	–
	40	65.28	–

Another important but not very surprising observation happens in all the three ciphers, for the same number of rounds, having fewer active S-boxes does not promise a better EDP or EDD.

We also used the characteristics suggested by Moriai's *et al.* method for 5-round MIDORI, 8- to 10-round SKINNY and 10- to 13-round CRAFT with different number of conditions n_c, where

$$n_c = \mathrm{hw}(\delta_{r-1}) - \frac{1}{s} \cdot \sum_{i=0}^{r-2} \log_2 \mathrm{Prob}(\delta_i \xrightarrow{\delta \mathrm{L}} \delta_{i+1}).$$

Using Moriai's *et al.* method of computation, the EDD of a characteristic with n_c conditions is equal to $2^{-s \cdot n_c}$. But we observed that the exact value can be both higher or lower than the value given by Moriai's *et al.* method. We summarized the results in Table 5.4. For instance, for 11-round CRAFT characteristics with $n_c = 15$, the maximum observed value for EDD is $2^{-55.58}$ and the minimum is $2^{-68.03}$, while Moriai's *et al.* method suggests 2^{-60} for all of these characteristics. Clearly, there is a significant gap between the exact value and the one suggested by the previous method.

Differentials and Linear Hulls

For each truncated characteristic, we checked all input/output differences or linear masks to find the ones with the maximum EDP or ELP. We summarized the best EDP or ELP for each number of rounds and number of active S-boxes in Table 5.5.

Again, an observation from this table is that for the same number of rounds, having less active S-boxes do not promise a better EDP or ELP. For example, we found 7-round MIDORI differentials with 37 active S-boxes and probability $2^{-55.75}$, while best found probability for characteristics with 35 or 36 active S-boxes are $2^{-58.37}$ and $2^{-59.96}$, resp.

The maximum number of rounds for which we found a differential with probability higher than 2^{-64} for MIDORI, SKINNY and CRAFT are 8, 9 and 14, resp. For the linear hull case, these numbers are 6, 8, and 13, resp.

Comparing to results from [AK19], they found an 8-round SKINNY differential with probability $2^{-56.55}$ based on 821896 characteristics (which do not necessarily follow the same activity pattern), while we have 8-round differentials with higher probability $2^{-49.50}$ and also 9-round differential with probability $2^{-59.24}$. In the case of MIDORI, there is an 8 round differential with probability $^{-60.86}$ based con 693730 characteristics in [AK19] which is higher than the probability for differential patterns that we checked.

Related-Tweak Differentials of CRAFT

Using the related-tweak characteristics with minimum active S-boxes, we computed the probability of all differentials for each characteristic and the values we found are the same as the ones given in Section 6.1.3.

CRAFT			
r	n_c	max EDP	max ELP
10	36	42.32	47.72
	37	42.18	48.01
	38	41.97	48.23
	39	45.76	49.11
11	40	48.22	52.01
	41	49.46	52.24
	42	49.81	52.49
	43	49.62	53.32
12	44	52.39	56.22
	45	53.62	56.48
	46	53.24	56.71
	47	53.91	57.30
13	48	56.55	60.46
	49	57.22	60.70
	50	57.03	60.94
	51	58.54	61.75
14	52	60.12	64.67
	53	59.92	64.92
	54	59.72	65.16
	55	62.65	65.97
15	56	64.72	68.90
	57	65.78	69.13
	58	65.95	69.37
	59	66.66	

6

Security Analysis of Some Lightweight Block Ciphers

"As light as a feather, and as hard as dragon-scales" —Bilbo Baggins describing Mithril in "The Lord of the Rings: The Fellowship of the Ring" by J.R.R Tolkien.

SECURITY ANALYSIS of cryptographic primitives is an important task not only during the design process by the designers but also afterward by the other researchers. As mentioned in Section 2.3, there are several kinds of attacks and security analysis methods for block ciphers.

In Section 6.1, a detailed analysis of the security of CRAFT is provided. Since the general structure of CRAFT is similar to that of AES, MIDORI [Ban+15], SKINNY and MANTIS [Bei+16], the security analysis of CRAFT is also more or less similar to that of those primitives. We provide the argument to show the security claim for CRAFT block cipher considering the cipher's security against accelerated exhaustive search, TDM, (impossible) differential, (zero-correlation) linear, MitM, integral, division property, and nonlinear invariant attacks. We also discuss the following up analysis by other researchers at the end of the section that as best of our knowledge, there is no attack against the cipher's security claim. This section is based on Section 5 of the article [Bei+19] that the author of the thesis wrote together with Christof Beierle, Gregor Leander, and Amir Moradi.

[Bei+19] Beierle, Leander, Moradi, and Rasoolzadeh, "CRAFT: Lightweight Tweakable Block Cipher with Efficient Protection Against DFA Attacks"

In Section 6.2, first, the security of PRINCEv2 based on the previously published analysis of PRINCE is analyzed. It is shown that several attacks successful against PRINCE, such as certain accelerated exhaustive search and MitM attacks, do not apply to PRINCEv2. Additionally, we provide several new analyses that include a linear attack, a 6-round integral distinguisher based on the division property, a more precise evaluation of boomerang attack using recently published techniques, and a new 10-round Demirci-Selçuk MitM attack. This section is based on Section 5 of the article [Bož+20] that the author of the thesis wrote together with Dušan Božilov, Maria Eichlseder, Miroslav Knežević, Baptiste Lambin, Gregor Leander, Thorben Moos, Ventzislav Nikov, Yosuke Todo, and Friedrich Wiemer.

[Bož+20] Božilov, Eichlseder, Knežević, Lambin, Leander, Moos, Nikov, Rasoolzadeh, Todo, and Wiemer, "PRINCEv2: More Security for (Almost) No Overhead"

In Section 6.3, we provide new insights and a better understanding of differential attacks of PRIDE block cipher. First, we show that two previous differential attacks are incorrect and describe (new and old) properties of the cipher that make such attacks intricate. Based on this understanding, we show how to mount a differential attack properly. This section is based on the article [LR17] that the author of the thesis wrote together with Virginie Lallemand.

[LR17] Lallemand and Rasoolzadeh, "Differential Cryptanalysis of 18-Round PRIDE"

6.1 GENERAL SECURITY ANALYSIS OF CRAFT

In this section, a detailed analysis of the security of CRAFT is provided. In fact, since the general structure of CRAFT is similar to that of AES, MIDORI [Ban+15], SKINNY and MANTIS [Bei+16], the security analysis of CRAFT is also more or less similar to that of those primitives.

From the provided analyses in the following, the most promising cryptanalysis on CRAFT is an accelerated exhaustive search with time complexity of 2^{124} cipher encryptions and data and memory complexity of 16 (see Section 6.1.1).

It is expected that a 30-round version of CRAFT attains 128-bit security against (impossible) differential and (zero-correlation) linear attacks, integral attacks, and invariant attacks as well as MitM attack. It is noteworthy that it does not mean there exists a 30-round attack on CRAFT. It is only an upper bound on the number of rounds that can be attacked.

Hence, considering two extra rounds as a security margin, it is claimed that 32-round CRAFT has 124-bit security in the related-tweak model. Note that this does not imply any security claim in the chosen-key, known-key, or related-key model.

Finally, in Section 6.1.8, the following up analyses by other researchers are briefly discussed that to the best of our knowledge, there is no attack to break the security claim of CRAFT.

6.1.1 *Exhaustive Search*

Due to the simple tweakey schedule of CRAFT, there are some deterministic related-key characteristics that can accelerate the typical exhaustive search attack. Consider K_0, K_1 and T as two halves of the key and tweak, resp. and $K_0' = K_0 \oplus \Delta$, $K_1' = K_1 \oplus \Delta$ and $T' = T \oplus \Delta$ where $\Delta = (x, x, \ldots, x)$ with $x \in \mathbb{F}_2^4$. Then, we have $\mathtt{QN}(\Delta) = \Delta$, which causes the relations below.

$$
\begin{aligned}
TK_0' &= K_0' \oplus T' = (K_0 \oplus \Delta) \oplus (T \oplus \Delta) = K_0 \oplus T = TK_0,\\
TK_1' &= K_1' \oplus T' = (K_1 \oplus \Delta) \oplus (T \oplus \Delta) = K_1 \oplus T = TK_1,\\
TK_2' &= K_0' \oplus \mathtt{QN}(T') = (K_0 \oplus \Delta) \oplus \big(\mathtt{QN}(T) \oplus \Delta\big) = K_0 \oplus \mathtt{QN}(T) = TK_2,\\
TK_3' &= K_1' \oplus \mathtt{QN}(T') = (K_1 \oplus \Delta) \oplus \big(\mathtt{QN}(T) \oplus \Delta\big) = K_1 \oplus \mathtt{QN}(T) = TK_3.
\end{aligned}
$$

This means encryption under two different tweakey tuples of (K_0, K_1, T) and (K_0', K_1', T') are the same. Using these deterministic characteristics, the attacker can accelerate the exhaustive search by a factor of 2^4. First, he asks for encryption of the same plaintext P under 16 different tweaks of $T, T \oplus \Delta_1, \ldots, T \oplus \Delta_{15}$ where $\Delta_x = (x, x, \ldots, x)$ that we show the corresponding ciphertexts by $C_0, C_1, \ldots, C_{15}$. Then, by setting one of the key nibbles to a constant value of zero, for each of 2^{124} possible key candidate (K_0^*, K_1^*), he computes C^*, the encryption of P using K_0^*, K_1^* and T. If C^* is equal to C_x, then $(K_0^* + \Delta_x, K_1^* + \Delta_x)$ is a candidate for the master key. This way, there will remain only about 2^{64} key candidates for the master key, which a check on another pair of plaintext and ciphertext will determine the only candidate for the master key. All together, exhaustive search attack on CRAFT using 16 chosen-plaintext data has complexity of 2^{124} encryptions.

6.1.2 *TDM Trade-offs*

Since CRAFT uses a simple tweakey schedule, it is necessary to analyze the security of the cipher against TDM trade-off attacks. Actually, some choices for the permutation Q in the tweak schedule give an opportunity to the attacker to do a TDM trade-off attack. In the following, the TDM trade-off attack is explained that helps to choose the permutation Q carefully.

In the offline phase, the attacker fixes the tweakeys to $TK_0 = 0, TK_1 = X, TK_2 = T'$ and $TK_3 = X \oplus T'$ which means:

$$K_0 = T, \quad K_0 \oplus K_1 = X, \quad T \oplus \mathtt{QN}(T) = T'.$$

For a fixed plaintext P and all possible values of X and T', he computes the corresponding ciphertext C and saves the X value in the index (T', C) of a table Tab. In the online phase, by asking the encryption for the same plaintext P and for all possible tweaks T, he receives the corresponding ciphertext C_T. Then for each value of T, he gets a candidate for $K_0 \oplus K_1$ by looking up to the index $\left(T \oplus \mathtt{QN}(T), C_T\right)$ of Tab. By doing an exhaustive search on the 2^{64} key candidates, he can find the correct value of the 128-bit key. Considering $d = \dim\left(\mathcal{T} \oplus \mathtt{QN}(\mathcal{T})\right)$,[1] this attack requires 2^{64+d} pre-computations, 2^{64+d} memory, 2^{65} online computions and 2^{64} data. But with some small modifications, it can be changed to all online attack with 2^{64+d} computations, 2^{64} data and memory.

[1] Recalling that $\mathcal{T}$ is the space of tweak that is $\mathbb{F}_2^{64}$.

Since a circular permutation Q has the maximum value for $\dim\left(\mathcal{T} \oplus \mathtt{QN}(\mathcal{T})\right)$ is 60, it is decided to use such a Q that improves the security of CRAFT against this attack. This means the TDM trade-off attack on CRAFT has 2^{124} time, 2^{64} data and memory complexity.

6.1.3 *Differential and Linear*

To argue for the resistance of CRAFT against differential and linear attacks, lower bounds on the minimum number of active S-boxes are computed, both in the single-tweak and related-tweak adversary model.

Table 6.1 shows the lower bounds on the minimum number of active S-boxes in the single-tweak model (ST) and the related-tweak model (RT) for 1 up to 17 rounds. To compute these bounds, two techniques are used: Matsui's recursive algorithm as explained in [Mat95] and MILP as explained in [Mou+11; Sun+13]. It is noteworthy that both approaches take only the properties of the linear layer into account and are independent of the specification of the S-box. After 13 rounds, all bounds are high enough to ensure that no distinguisher based on a single (related-tweak) differential (resp. linear) characteristic exists. Since the maximum differential probability (resp. absolute linear correlation) for an active CRAFT S-box is 2^{-2} (resp. 2^{-1}), having at least 32 active S-boxes will cause the probability (resp. absolute correlation) of a differential (resp. linear) characteristic to be less than or equal to 2^{-64} (resp. 2^{-32}). Hence, such a characteristic is not useful to distinguish the cipher from a random permutation. Since there is no cancellation of active S-boxes in linear characteristics in the related-tweak model, only the single-tweak model for the linear attack is considered [KLW17]. Thus, the bounds for single-tweak give valid bounds also for the related-tweak case.

[Mat95] Matsui, "On Correlation Between the Order of S-boxes and the Strength of DES"

[Mou+11] Mouha, Wang, Gu, and Preneel, "Differential and Linear Cryptanalysis Using Mixed-Integer Linear Programming"

[Sun+13] Sun, Hu, Song, Xie, and Wang, *Automatic Security Evaluation of Block Ciphers with S-bP Structures against Related-key Differential Attacks*

[KLW17] Kranz, Leander, and Wiemer, "Linear Cryptanalysis: Key Schedules and Tweakable Block Ciphers"

TABLE 6.1: Lower Bounds on the Minimum Number of Active S-boxes up to 17 Rounds. RT_i refers to the characteristic starting with round R_{4j+i}.

r	Lin.	Diff.				
		ST	RT_0	RT_1	RT_2	RT_3
1	1	1	0	0	0	0
2	2	2	1	1	1	1
3	4	4	2	2	2	2
4	6	6	4	5	4	5
5	10	10	6	7	6	7
6	14	14	12	10	12	10
7	20	20	14	15	16	15
8	26	26	19	18	19	18
9	32	32	22	22	21	21
10	36	36	25	24	24	24
11	40	40	27	28	27	28
12	44	44	32	32	30	31
13	48	48	36	35	34	34
14	52	52	38	38	39	38
15	56	56	40	43	41	39
16	60	60	46	45	42	41
17	64	64	49	46	44	47

Note that only a single characteristic is considered in analysis so far. Thus, the distinguishers might be stronger due to differential (resp. linear hull) effects. To have a better estimation about the probability of differentials (resp. potential of linear hulls), by fixing input and output differences (resp. masks) in the differential (resp. linear hull), all different single characteristics are found, which follow the same S-box activity pattern with the minimum number of active S-boxes. Then by summing all the probabilities (resp. square correlations) of each characteristic, a lower bound for the probability of the corresponding differential (resp. potential of the linear hull) is found. To this, the techniques introduced in Section 5.2 are applied. Note that it is a lower bound; because for a fixed input and output difference (resp. mask), there might be some other single characteristics that are not following the S-box activity pattern. However, since for such characteristics, the number of active S-boxes will be higher, it is assumed that their effect on the probability of differential (resp. potential of linear hull) is negligible.

In the single-tweak case, for 9-round CRAFT which has at least 32 active S-boxes, four optimal differentials are found, each having a probability of $2^{-39.58}$. Similarly, four optimal linear hulls are found, each having a potential of $2^{-43.53}$. Using the same method, the highest probability and potential are computed for a higher number of rounds. In the single-tweak differential case, for 14 round CRAFT, the following eight differentials with the probability of $2^{-60.12}$ (for the first four) and $2^{-60.64}$ (for the last four) are found:

$$\texttt{00a0 a0a0 0000 a0a0} \longrightarrow \texttt{a000 0000 0000 a0a0}$$
$$\texttt{000a 0a0a 0000 0a0a} \longrightarrow \texttt{0a00 0000 0000 0a0a}$$
$$\texttt{a000 a0a0 0000 a0a0} \longrightarrow \texttt{00a0 0000 0000 a0a0}$$
$$\texttt{0a00 0a0a 0000 0a0a} \longrightarrow \texttt{000a 0000 0000 0a0a}$$
$$\texttt{a0aa 00aa 0000 00aa} \longrightarrow \texttt{a000 0000 0000 00aa}$$
$$\texttt{0aaa 00aa 0000 00aa} \longrightarrow \texttt{0a00 0000 0000 00aa}$$
$$\texttt{aaa0 aa00 0000 aa00} \longrightarrow \texttt{00a0 0000 0000 aa00}$$
$$\texttt{aa0a aa00 0000 aa00} \longrightarrow \texttt{000a 0000 0000 aa00}$$

In the linear case, for 13 round CRAFT, the following four linear hulls with potential of $2^{-60.46}$ are found:

$$\texttt{2002 0000 2002 2202} \longrightarrow \texttt{0220 0000 0000 0002}$$
$$\texttt{2002 0000 2002 2022} \longrightarrow \texttt{2002 0000 0000 0020}$$
$$\texttt{0220 0000 0220 0222} \longrightarrow \texttt{2002 0000 0000 0200}$$
$$\texttt{0220 0000 0220 2220} \longrightarrow \texttt{0220 0000 0000 2000}$$

To recover the key, several rounds can be appended before and after the differentials or linear hulls. The number of appended rounds depends on the minimum number of rounds that achieve full diffusion, i. e., 7 rounds for CRAFT. Hence, the attacker can add at most 7 rounds both at the beginning and at the end of the characteristic. Thus, the total number of appended rounds can be at most $2 \times 7 = 14$ rounds. Therefore, it is expected that an attacker cannot have a successful single-tweak differential attack on more

than $14+14=28$ rounds and (single-/related-tweak) linear attack on more than $13+14=27$ rounds. It is noteworthy that 14 rounds for appending to the trails is an upper bound and is not always possible. For example, for the above-mentioned differentials and linear hulls, the maximum possible number of rounds to append is 7. It means that using those differentials or linear hulls, the attacker cannot have a successful single-tweak differential attack on 22 rounds or a linear attack on 21 rounds.

In the related-tweak cases, the differentials are dependent on the starting round, i. e., the index of related-tweak. For each $0 \leq i \leq 3$, in a process similar to the single-tweak case, the below differentials are found as the longest ones with a probability higher than 2^{-64}:

RT_0: 15-round with a probability of $2^{-55.14}$ and $\Delta T = \texttt{0000 0000 00a0 0000}$

$$\texttt{0000}\,x\texttt{000 0000 0000} \longrightarrow \texttt{0000 0000 00a0}\,z\texttt{000}$$

RT_1: 16-round with a probability of $2^{-57.18}$ and $\Delta T = \texttt{0000 0000 000a 0000}$

$$\texttt{000}x'\,\texttt{00}y\,x\,\texttt{000a 000}x \longrightarrow \texttt{0000 000}z\,\texttt{a000 0000}$$

RT_2: 17-round with a probability of $2^{-60.14}$ and $\Delta T = \texttt{0000 0000 00a0 0000}$

$$x\texttt{000}\,x'\texttt{a00 0000}\,x\texttt{a00} \longrightarrow \texttt{0000 0000 00a0}\,z\texttt{000}$$

RT_3: 16-round with a probability of $2^{-55.14}$ and $\Delta T = \texttt{0000 0000 00a0 0000}$

$$\texttt{0aa0 0000 00a0 0000} \longrightarrow \texttt{0000 0000 00a0}\,z\texttt{000}$$

where x, y and z is one of the `5`, `a`, `d` and `f` hexadecimal values and $x' = x \oplus \texttt{a}$.

For the key recovery, the number of rounds that can be appended for an RT_i differential is at most $4+i$ rounds before and 7 rounds after the differential. But for the above differentials, this number can be less than or equal to 10, 10, 12, and 13 rounds, resp. Therefore, it is the attacker cannot have a successful related-tweak differential attack on 30 rounds.

It is noteworthy that in our entire analyses, the time complexity to check the feasibility of the attacks is not considered. Hence, the number of rounds that can be analyzed by the attacker is an upper bound.

6.1.4 *Impossible Differentials and Zero-Correlation Linear Hulls*

The structure of CRAFT is quite similar to SKINNY. Both ciphers employ very sparse and binary MixColumn operations, and there exist several activity patterns that deterministically propagate through it. Because of that, there might exist distinguishers based on impossible differentials over a high number of rounds. Indeed, for SKINNY, among the conducted cryptanalytic attacks so far the bests are based on impossible differentials (see [Bei+16, Section 4.3] and [Ank+17; LGS17]). Therefore, it is crucial to evaluate the resistance of CRAFT against those attacks.

With the MILP approach (see [Cui+16; ST17]), it is possible to search for (truncated) impossible differentials over reduced-round versions of CRAFT. Thereby, both the input and output activity patterns are constrained to have at most two active nibbles, resp. The largest number of rounds for which an

[Ank+17] Ankele, Banik, Chakraborti, List, Mendel, Sim, and Wang, "Related-Key Impossible-Differential Attack on Reduced-Round Skinny"

[LGS17] Liu, Ghosh, and Song, "Security Analysis of SKINNY under Related-Tweakey Settings (Long Paper)"

[Cui+16] Cui, Jia, Fu, Chen, and Wang, *New Automatic Search Tool for Impossible Differentials and Zero-Correlation Linear Approximations*

[ST17] Sasaki and Todo, "New Impossible Differential Search Tool from Design and Cryptanalysis Aspects - Revealing Structural Properties of Several Ciphers"

impossible differential exists is 13. In total, the following twelve different 13-round truncated impossible differentials are found, where γ and δ can take any non-zero difference in $\mathbb{F}_2^4$. The first one is depicted in more detail in Figure 6.1 where a black cell indicates an active nibble (i. e., a non-zero difference), white indicates a passive nibble (i. e., a zero difference), and gray indicates a nibble that might be active or passive.

$$\begin{aligned}
&000\gamma\,0000\,000\gamma\,0000 \longrightarrow 0000\,0000\,\delta000\,0000\\
&000\gamma\,0000\,000\gamma\,0000 \longrightarrow 0000\,0000\,0\delta00\,0000\\
&000\gamma\,0000\,000\gamma\,0000 \longrightarrow 0000\,0000\,00\delta0\,0000\\
&00\gamma0\,0000\,00\gamma0\,0000 \longrightarrow 0000\,0000\,000\delta\,0000\\
&00\gamma0\,0000\,00\gamma0\,0000 \longrightarrow 0000\,0000\,00\delta0\,0000\\
&00\gamma0\,0000\,00\gamma0\,0000 \longrightarrow 0000\,0000\,0\delta00\,0000\\
&0\gamma00\,0000\,0\gamma00\,0000 \longrightarrow 0000\,0000\,000\delta\,0000\\
&0\gamma00\,0000\,0\gamma00\,0000 \longrightarrow 0000\,0000\,\delta000\,0000\\
&0\gamma00\,0000\,0\gamma00\,0000 \longrightarrow 0000\,0000\,00\delta0\,0000\\
&\gamma000\,0000\,\gamma000\,0000 \longrightarrow 0000\,0000\,000\delta\,0000\\
&\gamma000\,0000\,\gamma000\,0000 \longrightarrow 0000\,0000\,0\delta00\,0000\\
&\gamma000\,0000\,\gamma000\,0000 \longrightarrow 0000\,0000\,\delta000\,0000
\end{aligned}$$

Using the same approach as for impossible differentials (by considering the transpose of M instead of M), it is also possible to search for zero-correlation linear hulls over reduced-round versions of CRAFT. Again, the longest distinguisher that is found covers 13 rounds which are the following twelve (truncated) zero-correlation linear hulls:

$$\begin{aligned}
&0000\,000\gamma\,0000\,000\gamma \longrightarrow 0000\,00\delta0\,0000\,0000\\
&0000\,000\gamma\,0000\,000\gamma \longrightarrow 0000\,0\delta00\,0000\,0000\\
&0000\,000\gamma\,0000\,000\gamma \longrightarrow 0000\,\delta000\,0000\,0000\\
&0000\,00\gamma0\,0000\,00\gamma0 \longrightarrow 0000\,0\delta00\,0000\,0000\\
&0000\,00\gamma0\,0000\,00\gamma0 \longrightarrow 0000\,00\delta0\,0000\,0000\\
&0000\,00\gamma0\,0000\,00\gamma0 \longrightarrow 0000\,000\delta\,0000\,0000\\
&0000\,0\gamma00\,0000\,0\gamma00 \longrightarrow 0000\,\delta000\,0000\,0000\\
&0000\,0\gamma00\,0000\,0\gamma00 \longrightarrow 0000\,000\delta\,0000\,0000\\
&0000\,0\gamma00\,0000\,0\gamma00 \longrightarrow 0000\,00\delta0\,0000\,0000\\
&0000\,\gamma000\,0000\,\gamma000 \longrightarrow 0000\,000\delta\,0000\,0000\\
&0000\,\gamma000\,0000\,\gamma000 \longrightarrow 0000\,\delta000\,0000\,0000\\
&0000\,\gamma000\,0000\,\gamma000 \longrightarrow 0000\,0\delta00\,0000\,0000
\end{aligned}$$

where γ and δ can take any non-zero mask in $\mathbb{F}_2^4$.

By appending at most 14 rounds before and after the impossible differentials or zero-correlation linear hulls (the reason for 14 rounds is explained at the end of Section 6.1.3), the attacker can have successful key-recovery on a higher number of rounds. Actually, for the reported characteristics it is possible to only append 12 rounds. Thus, we conclude that the attacker may have a successful attack on *at most* $13 + 13 = 26$ rounds.

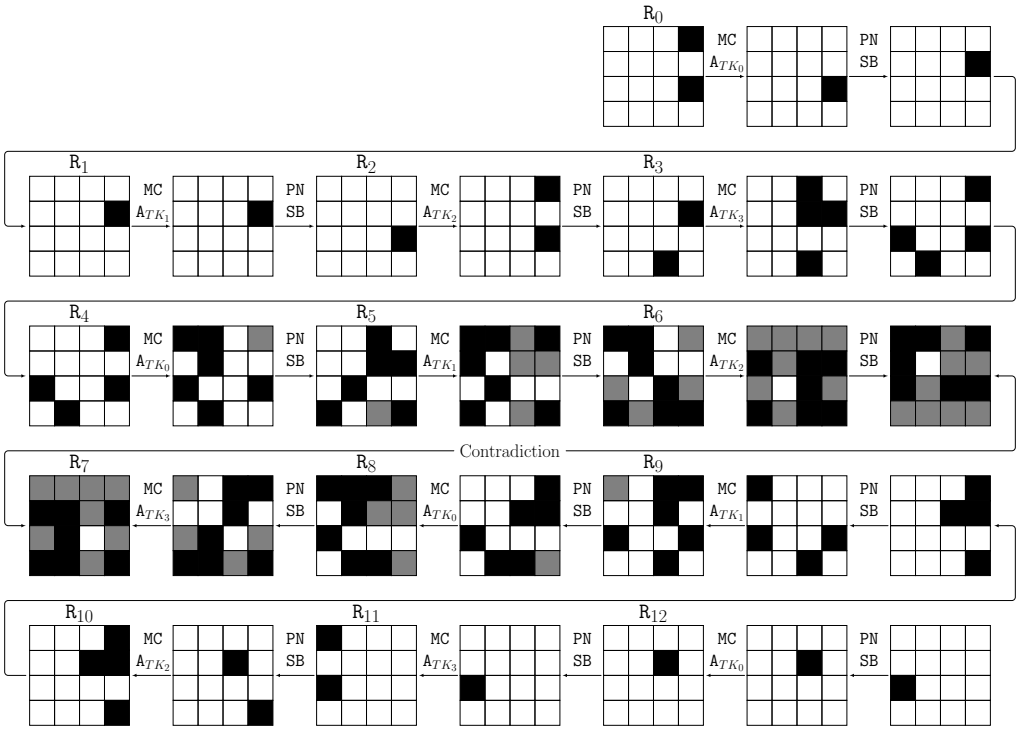


FIGURE 6.1: A 13-round Impossible Differential Characteristic for CRAFT. Note that active nibbles of the plaintext difference have the same value.

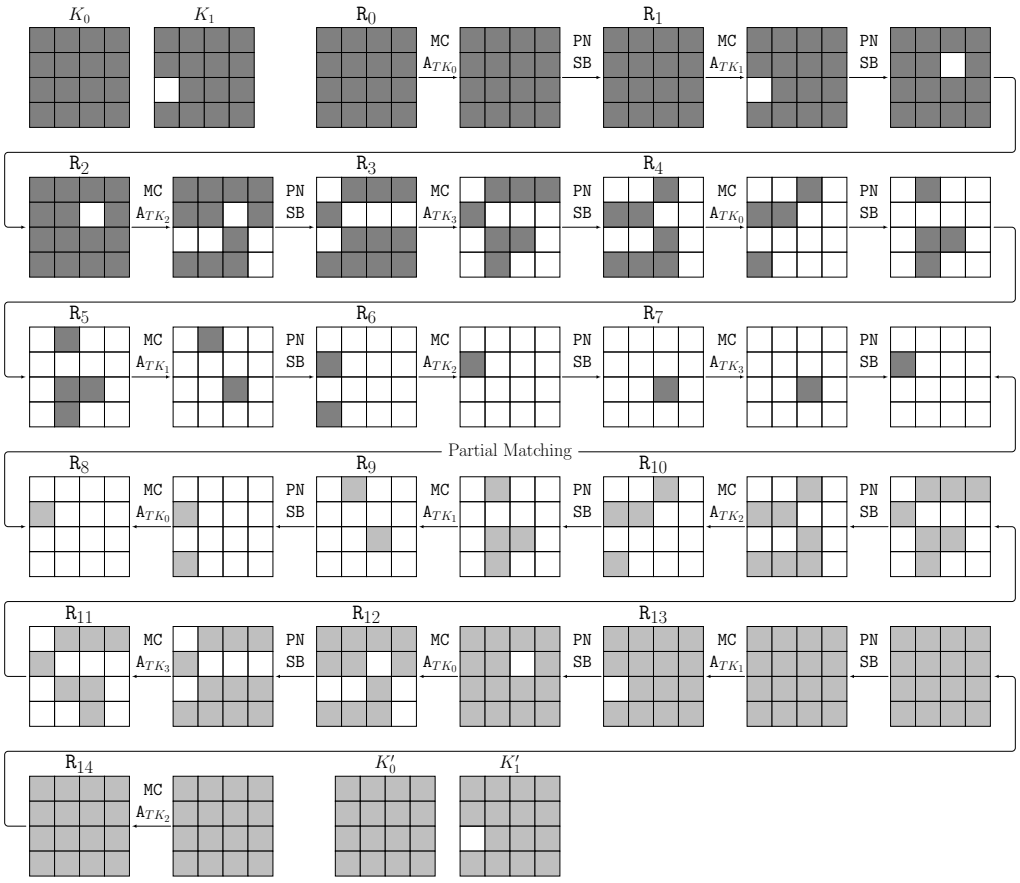


FIGURE 6.2: A 15-round Partial Matching MitM for CRAFT. Note that $K'_i = \mathtt{MC}(K_i)$.

6.1.5 *MitM*

[Sas11] Sasaki, "Meet-in-the-Middle Preimage Attacks on AES Hashing Modes and an Application to Whirlpool"

To discuss the resistance of CRAFT against MitM attacks, we use a similar approach as in its application to the SPN structure [Sas11] and the proposal of SKINNY and MIDORI. The maximum number of attacked rounds can be evaluated considering the maximum length of three features: partial-matching, initial structure, and splice-and-cut. For partial-matching, the number of rounds in both forward and backward directions cannot reach the full diffusion rounds (which for CRAFT is 7 rounds). Due to the key length being higher than the state size, we can add one more round in each direction. Also, because of the last linear half-round, partial-matching can work up to $2 \times (7-1+1)+1 = 15$ rounds. In fact, there are four partial-matching characteristics for 15 rounds of CRAFT and Figure 6.2 depicts one of them.

[SA09a] Sasaki and Aoki, "Finding Preimages in Full MD5 Faster Than Exhaustive Search"

The condition for the initial structure [SA09a] is that the key differential trails in both forward and backward directions do not share active non-linear components. As any key differential in CRAFT affects all 16 S-boxes after at least $7+1$ rounds in both directions, there is no such differential that shares active S-box(es) in more than 7 rounds. Therefore, it works up to 7 rounds for CRAFT.

Splice-and-cut may extend the number of attacked rounds up to the number of full diffusion rounds, i. e., 7. Thus, we conclude that *at most* a $(15+7+7) = 29$-round meet-in-the-middle attack might be feasible, but a higher number of rounds is secure. Note that the freedom of the tweak is already considered in the number of rounds for which there is a matching characteristic.

6.1.6 *Integral and Division Property*

Using the technique explained in Section 2.3.7, the longest integral characteristics that can be found cover 13 rounds, one of which is shown in Figure 6.3. The existence of integral distinguishers based on the division property using the MILP approach described in [Xia+16] is also checked. With that, distinguishers for more than 13 rounds are not found. By appending *at most* 7 rounds for key recovery to the rest of the characteristic, the attacker might have a successful attack on *at most* 20 rounds.

[Xia+16] Xiang, Zhang, Bao, and Lin, "Applying MILP Method to Searching Integral Distinguishers Based on Division Property for 6 Lightweight Block Ciphers"

6.1.7 *Invariant*

[Bei+17] Beierle, Canteaut, Leander, and Rotella "Proving Resistance Against Invariant Attacks: How to Choose the Round Constants"

Beierle, Canteaut, Leander, and Rotella [Bei+17] presented a security argument on the resistance against invariant attacks based on the round constants of the SPN ciphers with a simple key schedule which can easily be applied to CRAFT. In particular, let $\mathtt{R}_i$ and $\mathtt{R}_j$ be two rounds in which the same round tweakey is added and let $d_{i,j} := \mathtt{PN} \circ \mathtt{MC}(RC_i \oplus RC_j)$. If the smallest $(\mathtt{PN} \circ \mathtt{MC})$-invariant subspace that contains $d_{i,j}$ has a dimension of at least $n-1 = 63$, then any Boolean function that is an invariant for both $\mathtt{PN} \circ \mathtt{MC} \circ \mathtt{A}_{RC_i} \circ \mathtt{A}_{TK'_i}$ and $\mathtt{PN} \circ \mathtt{MC} \circ \mathtt{A}_{RC_j} \circ \mathtt{A}_{TK'_j}$ must be trivial or affine. Note that here for the simplicity, the round operations are re-ordered and considered an equivalent tweakey that is used before the MixColumn operation. However, as the S-box

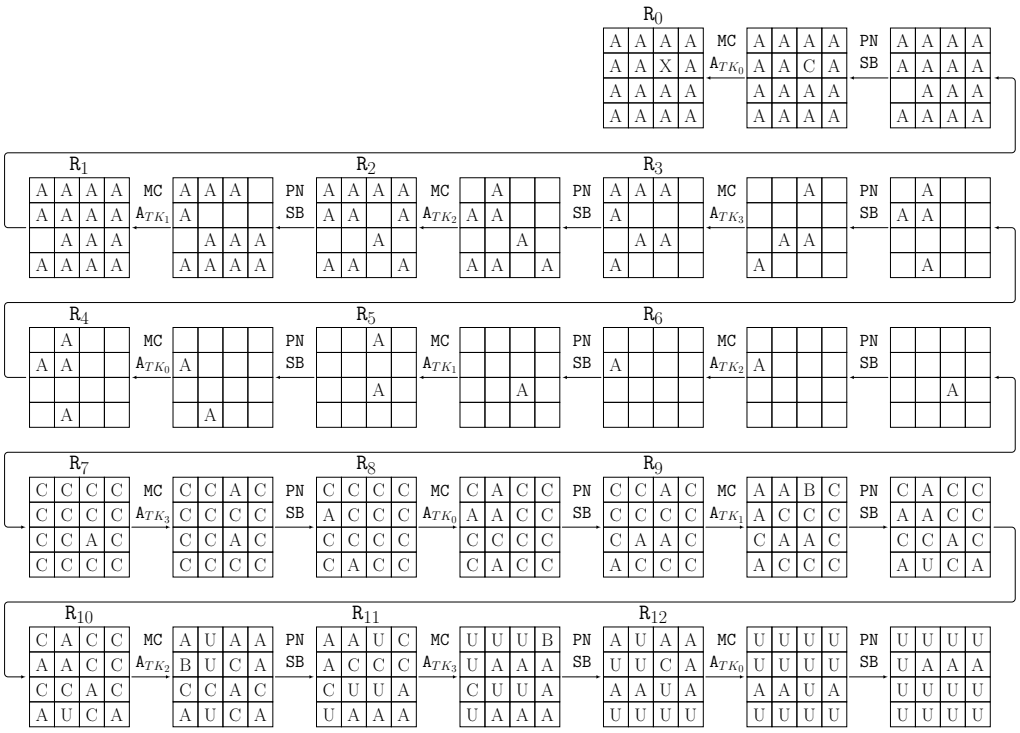


FIGURE 6.3: A 13-round Integral Distinguisher for CRAFT.

has no component of algebraic degree one, such an invariant would be useless to attack the cipher. For CRAFT, the smallest PN ∘ MC-invariant subspace that contains $\{d_{0,4}, d_{1,5}, d_{2,6}, d_{3,7}\}$ is $\mathbb{F}_2^{64}$, i. e., it has dimension 64. Therefore, we already get a sound security argument on the resistance against invariant attacks after 8 rounds.

6.1.8 *Follow Up Analyses*

The security claim for CRAFT is resistance against related-tweak attacks with time complexity of less than 2^{124}. In the design paper, several analyses on CRAFT were already conducted by the designers. After the publication of the design, some other follow-up cryptanalysis has been published [MA19; EY19; Had+19]. In the following, these analyses are briefly explained.

Hadipour, Sadeghi, Niknam, Song, and Bagheri [Had+19] presented several detailed analysis on CRAFT. First, using the technique introduced in [Ank+19], they presented 14-round related-tweak zero-correlation linear hull distinguishers. Using the same distinguishers and following the connection between zero-correlation and integral distinguisher [Sun+15], they also presented a 14-round related-tweak integral distinguishers. Secondly, using the automated search model based on CRYPTOSMT [Köl14] and MILP tool, they found the mistake reported on the differential probability and on the maximum number of rounds for single-tweak differential distinguishers. More precisely, in the proposal article of CRAFT [Bei+19], the reported differential probabilities were wrong because of an overflow problem in the 64-bit integers used in the corresponding C++ programs. Using the erroneous program, it was reported that the 10-round single-tweak differential distinguishers with a minimum number of active S-boxes

[Had+19] Hadipour, Sadeghi, Niknam, Song, and Bagheri "Comprehensive security analysis of CRAFT"

[Ank+19] Ankele, Dobraunig, Guo, Lambooij, Leander, and Todo, "Zero-Correlation Attacks on Tweakable Block Ciphers"

[Sun+15] Sun, Liu, Rijmen, Li, Cheng, Wang, AlKhzaimi, and Li, "Links Among Impossible Differential, Integral and Zero Correlation Linear Cryptanalysis"

has a probability of $2^{-62.61}$ and it is the longest single-tweak differential distinguisher. The authors of [Had+19] presented 10, 11, 12, 13, and 14 rounds of the differential in single-tweak model with the probabilities of $2^{-44.89}, 2^{-49.79}, 2^{-54.48}, 2^{-59.13}$, and $2^{-63.80}$, resp. Note that these probabilities are smaller than the ones reported in Section 5.3 and it is because CRYPTOSMT needs to consider all the single characteristics following the given differential activity pattern while as it is mention in [Had+19], since it was time-consuming, the authors stopped their program after a time.

[MA19] Moghaddam and Ahmadian *New Automatic search method for Truncated-differential characteristics: Application to Midori, SKINNY and CRAFT*

Moghaddam and Ahmadian [MA19] used a MILP-based tool to find truncated differentials distinguishers for CRAFT, MIDORI, and SKINNY block ciphers. For CRAFT, they reported a 12-round distinguisher that its comparison is explained in Section 5.3 in more detail.

[EY19] ElSheikh and Youssef "Related-Key Differential Cryptanalysis of Full Round CRAFT"

Even though we did not claim any security in related-key adversary model, ElSheikh and Youssef [EY19] presented a related-key differential attack that recovers the whole 128-bit key in a full-round CRAFT with querying the corresponding ciphertext for about 2^{36} chosen plaintexts and time complexity of about 2^{36} encryptions using a negligible memory.

[AA20] Ahmadi and Aref "Generalized Meet in the Middle Cryptanalysis of Block Ciphers With an Automated Search Algorithm"

Recently, Ahmadi and Aref [AA20] presented a generalized MitM analysis of several block ciphers, including CRAFT which is an accelerated exhaustive search kind of analysis. Applying the same related-key related-tweak property of CRAFT explained in Section 6.1.1, using 2^{12} blocks of memory and 2^{56} chosen plaintexts and querying the corresponding ciphertexts, they can recover the key with time complexity of $2^{123.25}$ encryption. They also presented two single-tweak analysis that both are using 2^{28} blocks of memory and one with 2^{48} and $2^{126.53}$ data and time complexity, resp., and the other one with 2^{64} and $2^{126.37}$.

6.2 GENERAL SECURITY ANALYSIS OF PRINCEV2

In this section, the security of PRINCEV2 building on the previously published analysis of PRINCE is analyzed. Most dedicated attacks on PRINCEV2 are comparable to PRINCE, while PRINCEV2 offers significantly better resistance to generic attacks. It is shown that several attacks successful against PRINCE, such as certain accelerated exhaustive search and MitM attacks, do not apply to PRINCEV2. Note that the analyses of variants with modified operations or related-key attacks are not considered, but a discussion for the latter is provided at the end of this section.

Since PRINCEV2 is designed to provide a higher security level, it is also necessary to consider attacks with higher complexity than for PRINCE. For several dedicated attack strategies, it is observed that the attacks that cover 1 or 2 more rounds need significantly higher time complexity T or data complexity D. This higher complexity is above the bound $D \cdot T < 2^{126}$ claimed for PRINCE, but below the generic bounds of $D < 2^{64}$, $T < 2^{128}$ for PRINCEV2, and thus relevant to judge the security margin of PRINCEV2. We note that the security claim for PRINCEV2 limits the attacker to $D < 2^{47}$ and $T < 2^{112}$, while most of the results we propose for round-reduced PRINCEV2 require more data than permitted by this bound.

Additionally, several new results are provided that include a linear attack,

TABLE 6.2: Overview of the analysis results on round-reduced PRINCEv2, where 12 is the full number of rounds. Time complexity is given in unit of encryption equivalents, data complexity in known plaintexts (KP) or chosen plaintexts (CP). For most attacks in this table except MitM (†), the attacks on PRINCE apply to PRINCEv2 with similar complexity and vice-versa; however, the complexity of the attacks listed for PRINCEv2 is higher than permitted by the PRINCE security claim. For other results, refer to details in Section 6.2.

Attack	Target		Complexity		Reference
	Version	Rounds	Time	Data	
Differential	PRINCE	10	2^{61}	2^{58} CP	[Can+15]
	PRINCEv2	11	$< 2^{92}$	2^{59} CP	New
Imp. Diff.	PRINCE	7	2^{54}	2^{56} CP	[Din+17]
	PRINCE	9	$e^{-\alpha} \cdot 2^{128}$	$\alpha \cdot 2^{65}$ CP	New
	PRINCEv2	9	$e^{-\alpha} \cdot 2^{128}$	$\alpha \cdot 2^{65}$ CP	New
Integral	PRINCE	7	2^{57}	2^{60} CP	[Mor17]
	PRINCE	8	$2^{107.4}$	2^{36} CP	New
	PRINCEv2	8	$2^{107.4}$	2^{36} CP	New
MitM	PRINCE †	10	2^{122}	2 KP	[RR16b]
	PRINCE †	10	2^{96}	2^{32} CP	New
	PRINCEv2 †	10	2^{112}	2^{48} CP	New

a 6-round integral distinguisher based on the division property, a more precise evaluation of boomerang attack using recently published techniques, and a new 10-round Demirci-Selçuk MitM attack.

Section 6.2 provides an overview of the highlights of this section, including the best attacks on PRINCEv2 and noteworthy new results. In summary, PRINCEv2 is at least as secure as PRINCE against various dedicated attacks and provides better generic security.

6.2.1 *Differential*

In [ALL12; Can+15; DP15; GR16], security of PRINCE is analyzed against differential attack. The truncated differential used in [GR16] applies the subspace trail technique and attacks at most 6 rounds of PRINCE, which is the same for PRINCEv2. The inside-out differential in [ALL12] took advantage of the key-less middle round to attack at most 6 rounds of PRINCE and is thus not applicable to PRINCEv2.

The most powerful differential attack against PRINCE was the one introduced in [Can+15] that covers 10 rounds using $2^{57.94}$ chosen plaintexts, $2^{60.62}$ computations and $2^{61.52}$ blocks of memory. The distinguisher of this attack covers 6-round and it appends 2 rounds before and 2 rounds after which needs to guess 66 key bits. Using the same distinguisher and attack for PRINCEv2, we need to guess 64 key bits. The complexities of both attacks are roughly the same. The probabilitiy of the corresponding differentials are summarized in Table 6.3.

This attack can be extended by one round with the new key schedule at the cost of significantly higher time complexity, as illustrated in Figure 6.4.

1. Query N_s structures of 2^{32} chosen plaintexts P with second and forth

[ALL12] Abed, List, and Lucks, *On the Security of the Core of PRINCE Against Biclique and Differential Cryptanalysis*

[Can+15] Canteaut, Fuhr, Gilbert, Naya-Plasencia, and Reinhard, "Multiple Differential Cryptanalysis of Round-Reduced PRINCE"

[DP15] Derbez and Perrin, "Meet-in-the-Middle Attacks and Structural Analysis of Round-Reduced PRINCE"

[GR16] Grassi and Rechberger, "Practical Low Data-Complexity Subspace-Trail Cryptanalysis of Round-Reduced PRINCE"

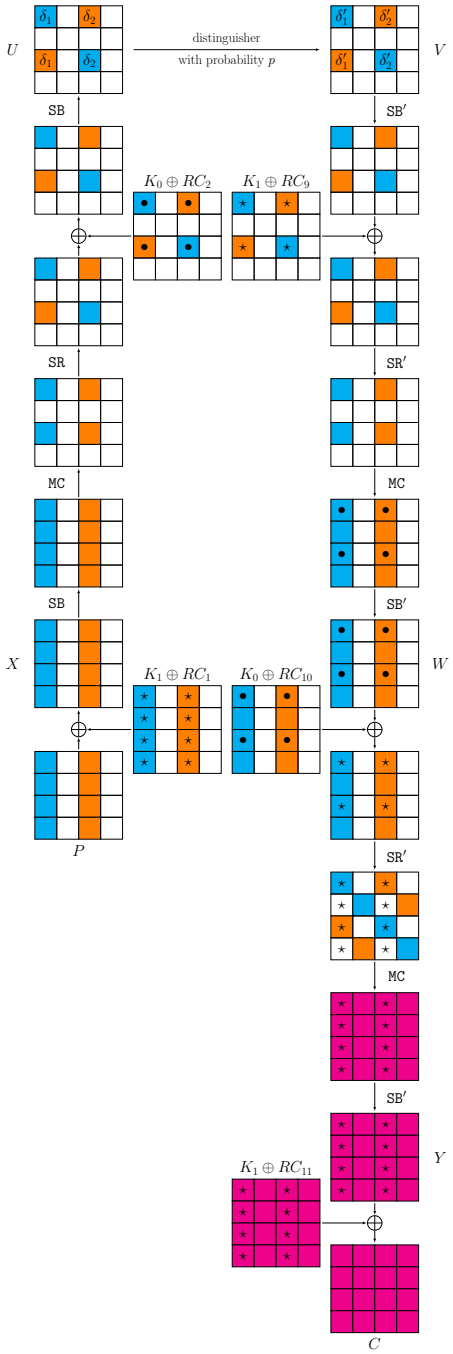


FIGURE 6.4: Differential Attack on 11-round PRINCEv2 by Extending [Can+15] by One Round.

columns fixed within each structure. This yields $N_s \cdot 2^{31} \cdot (2^{32}-1) \approx N_s \cdot 2^{63}$ candidate pairs as in Figure 6.4.

2. For each of the $N_s \cdot 2^{63}$ candidate pairs (P, P'), we expect 1 candidate for the 96-bit key (all of K_1 together with first and third columns of K_0) which can be determined by a small number of table lookups:

   a) We expect 1 key candidate for the first and third columns of K_1 (denoted by $\star$) that satisfies the pattern through SB, MC in rounds 1 and 11. This costs one lookup in a precomputed table $(\Delta X_0, \Delta Y_0, X_0 \oplus Y_0) \to X_0$ per pair and per column. Note that X_0 and Y_0 are denoting the first column of X and Y states.

   b) For any fixed difference ΔU, we also get 1 key candidate for the 4 nibbles of K_0 involved in round 2, which determines 4 nibbles of W in round 10 (denoted by $\bullet$).

   c) Due to the pattern for MC in round 10, there are only $2^8 \times 2^8$ possible differences ΔW. The key bits found so far determine 16 bits of this difference and thus on average fully determine ΔW.

   d) For any fixed difference ΔV, knowing the first and third columns of K_1, we get on average 1 candidate for the value of the two active columns of W. This determines the difference ΔZ and thus the second and fourth columns of K_1 and the rest of K_0.

3. Rank the obtained $N_s \cdot 2^{63}$ candidates for the 96-bit key (all of K_1 together with first and third columns of K_0).

Using $N_s = 5 \cdot p^{-1} \cdot 2^{-31} = 2^{29}$ structures for the best differential $(\delta_0, \delta_1, \delta'_0, \delta'_1) = (1, 2, 1, 2)$ from Table 6.3, we expect about 5 right pairs, which should be easily sufficient for distinguishing. The remaining 32 key bits (the second and fourth columns of K_0) can be recovered by brute force search. The overall complexity is $N_s \cdot 2^{32} = 2^{61}$ chosen plaintexts and the time corresponding to $N_s \cdot 2^{63} = 2^{92}$ repetitions of a few table lookups and arithmetic operations, which can be roughly approximated by one encryption equivalent.

The attack can be slightly improved using multiple differentials from Table 6.3, but fewer structures. For example, we can use the 2^2 permutations of the best differential with the same p and decrease N_s by a factor of 2^2, obtaining an attack with the same expected number of valid pairs and key candidates, while the data complexity is reduced to 2^{59} and the time complexity is slightly lower than before.

6.2.2 *Linear*

Even though there is no published linear analysis for PRINCE, we find out that the activity patterns used in [Can+15] for differential analysis are also useful for the linear one. This is because of the MC operation that uses an involutive and self-transpose matrix, i. e., $M'^T = M'^{-1} = M'$. We compute the potential (ELP) of all 6-round linear hulls which follow the given activity patterns similarly as in [Can+15] for both PRINCE and PRINCEv2. These values are summarized in Table 6.3.

TABLE 6.3: Differentials and Linear Hulls Fitting to the 6-round Activity Patterns Given in [Can+15] for Both PRINCE and PRINCEv2.

Differential Probability (EDP) (divided by $2^{-72} \cdot 2^3$)				Potential (ELP) (divided by $2^{-91} \cdot 2^3$)			
(δ_0, δ_1)	(δ'_0, δ'_1)	PRINCE	PRINCEv2	(δ_0, δ_1)	(δ'_0, δ'_1)	PRINCE	PRINCEv2
(1,2)	(1,2)	6144	2560	(4,2)	(4,2)	3563313280	2701826048
(1,2)	(1,8)	3328	1344	(4,2)	(4,8)	243931552	177559040
(1,2)	(1,a)	1664	672	(4,2)	(4,a)	215632256	165965824
(1,2)	(4,2)	1536	640	(4,2)	(5,2)	44716840	22956032
(1,2)	(4,8)	1664	928	(4,2)	(5,8)	23525008	14792960
(1,2)	(4,a)	832	336	(4,2)	(5,a)	3662080	2491520
(1,8)	(1,8)	2112	784	(4,8)	(4,8)	16864144	11669888
(1,8)	(1,a)	1056	392	(4,8)	(4,a)	14706248	10906624
(1,8)	(4,2)	832	336	(4,8)	(5,2)	3338620	1510400
(1,8)	(4,8)	1056	520	(4,8)	(5,8)	1723948	972992
(1,8)	(4,a)	528	196	(4,8)	(5,a)	256192	163616
(1,a)	(1,a)	528	196	(4,a)	(4,a)	13067272	10194944
(1,a)	(4,2)	416	168	(4,a)	(5,2)	2613530	1409536
(1,a)	(4,8)	528	260	(4,a)	(5,8)	1385768	908416
(1,a)	(4,a)	264	98	(4,a)	(5,a)	219776	153088
(4,2)	(4,2)	384	160	(5,2)	(5,2)	1221068	203976
(4,2)	(4,8)	416	232	(5,2)	(5,8)	698555	133008
(4,2)	(4,a)	208	84	(5,2)	(5,a)	54472	20960
(4,8)	(4,8)	656	388	(5,8)	(5,8)	466562	87600
(4,8)	(4,a)	264	130	(5,8)	(5,a)	27160	13544
(4,a)	(4,a)	132	49	(5,a)	(5,a)	4024	2312

One can use these linear hulls to analyze 10-round PRINCE by guessing 66 key bits and on PRINCEv2 with 64 key bits guesses (similar to differential attack). Data complexity for both of these attacks is a factor of 2^{57} known plaintexts.

6.2.3 *Impossible Differential*

The best previously known impossible-differential based attack was discovered by Ding, Zhao, Li, and Yu [Din+17], based on a 4-round distinguisher and extended to an attack up to 7 rounds with 2^{56} data, $2^{53.8}$ time and 2^{43} bytes of memory.

[Din+17] Ding, Zhao, Li, and Yu "Impossible Differential Analysis on Round-Reduced PRINCE"

Sasaki and Todo proposed a new way to search for impossible differentials [ST17] based on MILP, leading to much more sophisticated distinguishers than previously known. By implementing this algorithm and several new impossible distinguishers, over 5 rounds are found, which are given below. Note that there are two different configurations for our distinguishers: either $1+2+2$, which means one forward round, the two middles rounds, and 2 backward rounds, or $2+2+1$, i. e., two forward rounds, two middle rounds, and one backward round.

[ST17] Sasaki and Todo, "New Impossible Differential Search Tool from Design and Cryptanalysis Aspects - Revealing Structural Properties of Several Ciphers"

$$\texttt{0010 0000 0000 0000} \xrightarrow{2+2+1} \texttt{0000 0000 4000 0000}$$
$$\texttt{0000 0000 0010 0000} \xrightarrow{2+2+1} \texttt{0000 0000 0100 0000}$$
$$\texttt{0000 0000 0000 4000} \xrightarrow{2+2+1} \texttt{0400 0000 0000 0000}$$
$$\texttt{0000 0000 0000 0010} \xrightarrow{2+2+1} \texttt{0010 0000 0000 0000}$$

$$0400\,0000\,0000\,0000 \xrightarrow{1+2+2} 0000\,0000\,0000\,4000$$
$$0010\,0000\,0000\,0000 \xrightarrow{1+2+2} 0000\,0000\,0000\,0010$$
$$0000\,0000\,4000\,0000 \xrightarrow{1+2+2} 0010\,0000\,0000\,0000$$
$$0000\,0000\,0100\,0000 \xrightarrow{1+2+2} 0000\,0000\,0010\,0000$$

Due to the specific shape of these impossible differentials (only one active bit in the input and output), we can take any one of them and use it to mount an attack up to 9 rounds. Using [BNS14], we were able to estimate that the resulting attack would need $\alpha \cdot 2^{65}$ data and memory, and $2^{128} \cdot e^{-\alpha}$ time, where α is a parameter allowing for a trade-off between data/memory and time complexities (the higher is α, the higher is the data/memory complexity and the lower is the time complexity). Note that the results are the same for both PRINCE and PRINCEv2.

[BNS14] Boura, Naya-Plasencia, and Suder, "Scrutinizing and Improving Impossible Differential Attacks: Applications to CLEFIA, Camellia, LBlock and Simon"

6.2.4 *Integral, Higher-Order Differential and Division Property*

Several integral or higher-order differential attacks are introduced to analyze PRINCE [Jea+14; Mor17; PN15; RR16c]. The longest known distinguisher of these types is a higher-order differential introduced in [Mor17]. This distinguisher includes 5 nonlinear layers and needs a data set of size 2^{32}. For key recovery, it is possible to append at most 3 rounds to the end of the distinguisher and attack 8 round PRINCEv2 by guessing 80 key bits. The complexity of this attack is about 2^{112} computations (equivalent to $2^{107.4}$ 8-round PRINCEv2 encryption), 2^{36} chosen plaintexts and 2^{36} blocks of memory. Note that this attack is applicable to PRINCE with the same complexity.

To search bit-based division property distinguishers, we implement the MILP algorithm introduced in [Xia+16] and we ended up finding such a distinguisher over 6 rounds. This distinguisher requires 2^{62} chosen plaintexts, and due to this high data complexity, we do not expect it to be used to mount an attack over more than 8 rounds while also having better complexities than the above-mentioned attack.

[Jea+14] Jean, Nikolic, Peyrin, Wang, and Wu, "Security Analysis of PRINCE"

[Mor17] Morawiecki, "Practical attacks on the round-reduced PRINCE"

[PN15] Posteuca and Negara, "Integral Cryptanalysis of Round-Reduced PRINCE Cipher"

[RR16c] Rasoolzadeh and Raddum, "Faster Key Recovery Attack on Round-Reduced PRINCE"

[Xia+16] Xiang, Zhang, Bao, and Lin, "Applying MILP Method to Searching Integral Distinguishers Based on Division Property for 6 Lightweight Block Ciphers"

6.2.5 *Boomerang Attacks*

The boomerang attack is applied to PRINCE in [PDN15], but there are some flaws in the estimation of the probability, e. g., the effect of the boomerang switching [BK09] is not taken in consideration. The so-called sandwich attack [DKS10] is an experimental approach to estimate more rigorously this probability. We estimated the probability for a boomerang distinguisher with 4-round plus the middle layer. The probability of this 6-round distinguisher is about 2^{-34}, but the 7-round distinguisher is clearly worse than the classical differential attack because it involves 9 additional active S-boxes.

[PDN15] Posteuca, Duta, and Negara, "New Approaches For Round-Reduced PRINCE Cipher Cryptanalysis"

[BK09] Biryukov and Khovratovich, "Related-Key Cryptanalysis of the Full AES-192 and AES-256"

[DKS10] Dunkelman, Keller, and Shamir, "A Practical-Time Related-Key Attack on the KASUMI Cryptosystem Used in GSM and 3G Telephony"

6.2.6 *Accelerated Exhaustive Search*

In [Jea+14; RR16b; RR16a], some techniques are applied to accelerate the exhaustive key search in PRINCE block cipher. These attacks on PRINCE either used the α-reflection and FX-construction property or the key-less middle rounds of the cipher to accelerate the exhaustive search. For PRINCEv2, these

[RR16b] Rasoolzadeh and Raddum, "Cryptanalysis of PRINCE with Minimal Data"

[RR16a] Rasoolzadeh and Raddum, *Cryptanalysis of 6-round PRINCE using 2 Known Plaintexts*

properties do not hold anymore, and the only possible attack of this type can be the one used in [RR16b]. Using this technique and starting from the middle of the cipher, attacking 4 round or more requires guessing all the key bits. Starting from the plaintext/ciphertext side, attacking 6 rounds or more requires guessing all the key bits.

BICLIQUE ATTACK The Biclique attacks in [ALL12; YPO15] could accelerate an exhaustive search maximally by a factor of 2, exploiting the FX-construction in PRINCE. Since PRINCEv2 is not an FX-construction anymore and this attack does not improve the exhaustive search generally, we expect this attack to be not applicable.

[ALL12] Abed, List, and Lucks, *On the Security of the Core of PRINCE Against Biclique and Differential Cryptanalysis*

[YPO15] Yuan, Peng, and Ou, *Two Kinds of Biclique Attacks on Lightweight Block Cipher PRINCE*

6.2.7 *MitM*

Due to the special structure of the PRINCE, several MitM attacks are applied on this cipher [CNV13; DP15; LJW13; RR16b]. MitM attacks used in [LJW13; RR16b] took advantage of key-less middle rounds and use super-sboxes in the middle of the cipher to attack at most 10 rounds. These attacks do not work on PRINCEv2.

The sieve-in-the-middle attack, introduced in [CNV13], applies to 8 rounds of PRINCE which includes 6 rounds for the sieve-in-the-middle and 2 rounds for the biclique part. Applying it to PRINCEv2, the sieve-in-the-middle part will be more complicated. The Super S-box used there will be key-dependent, which increases the time and memory complexity.

The meet-in-the-middle attack used in [DP15] reaches 10 rounds of PRINCE. Applying the tool given in [Der19] by Patrick Derbez, which uses the same technique, we find out that it is possible to attack at most 6 rounds of PRINCEv2 with the complexity of either 2^{96} computations and 2^{26} memory blocks or 2^{112} computations and 2^{6} memory blocks.

We also analyzed the security of PRINCE and PRINCEv2 against Demirci-Selçuk meet-in-the-middle attacks using the same tool by Derbez. This attack can reach at most 10 rounds using 2^{48} chosen plaintexts, 2^{112} computations and 2^{70} memory blocks, while for PRINCE on the same number of rounds the complexity is 2^{32} chosen plaintexts, 2^{96} computations and 2^{70} memory blocks.

[CNV13] Canteaut, Naya-Plasencia, and Vayssière, "Sieve-in-the-Middle: Improved MITM Attacks"

[DP15] Derbez and Perrin, "Meet-in-the-Middle Attacks and Structural Analysis of Round-Reduced PRINCE"

[LJW13] Li, Jia, and Wang, *Improved Meet-in-the-Middle Attacks on AES-192 and PRINCE*

[Der19] Derbez, *AES Automatic Tool*

6.2.8 *TDM Trade-offs and Collisions*

There are two published TDM trade-offs against PRINCE in [Din15; Jea+14] Excluding trivial Diffie-Hellman TDM trade-off, all of these attacks used FX-construction property of the PRINCE and do not work on the PRINCEv2. Besides, the FX-construction of PRINCE allows a collision-based attack, using 2^{32} data, 2^{96} off-line and 2^{32} on-line computations, to recover the key [FJM14]. But again, it does not apply to PRINCEv2.

[Din15] Dinur, "Cryptanalytic Time-Memory-Data Tradeoffs for FX-Constructions with Applications to PRINCE and PRIDE"

[FJM14] Fouque, Joux, and Mavromati, "Multi-user Collisions: Applications to Discrete Logarithm, Even-Mansour and PRINCE"

6.2.9 *Remarks about Related-Key Attacks*

It is important to emphasize that the security claim of PRINCEv2 does not cover any security under related-key attacks, but it is also important to understand the impact on PRINCEv2 when attackers can use related-key attacks.

First of all, when both rotational and XOR-difference relations are allowed, the trivial related-key distinguishing attack is possible by exploiting the convertible property from encryption to decryption. Even if the relationship is restricted to XOR-difference, attackers can still attack PRINCEv2 by using related-key boomerang attacks. The related-key boomerang attack is applied to PRINCE without whitening keys in [Jea+14]. Inducing differences with the key allows attackers to construct iterative related-key differential characteristics whose number of active S-boxes is only one in each round. This iterative property is lost in the middle round R', but attackers can overcome R' by using a related-key boomerang attack, where two iterative related-key differential characteristics are connected. The new key schedule for PRINCEv2 is not designed to avoid this related-key boomerang attack, and there are related-key boomerang characteristics with 12 active S-boxes. Similar to the single-key boomerang attack, evaluating the probability in detail requires analyzing the effect of the boomerang switching. However, we can estimate the probability is roughly $2^{-12\times2\times2} = 2^{-48}$ from the number of active S-boxes, and it implies that PRINCEv2 is not secure against the related-key attack.

Again, we never claim any security under related-key attacks, and we believe that a related-key attack never happens in the environment that PRINCEv2 is demanded.

6.3 DIFFERENTIAL ANALYSIS OF PRIDE

PRIDE is a lightweight block cipher proposed at Crypto 2014 by Albrecht, Driessen, Kavun, Leander, Paar, and Yalçin [Alb+14] that is software-oriented and reaches notable performance figures when implemented on 8-bit microcontrollers.

[Alb+14] Albrecht, Driessen, Kavun, Leander, Paar, and Yalçin "Block Ciphers - Focus on the Linear Layer (feat. PRIDE)"

Previous analyses on PRIDE include a new cryptanalytic TDM data tradeoff by Dinur [Din15] presented at Eurocrypt 2015, while Bhaumik, Dutta, Guo, Jean, Mouha, and Nikolić [Bha+15] gave observations on the impact of increasing the number of rounds of the cipher. The more powerful attacks published to date are a related-key differential attack of the full cipher by Dai and Chen [DC14], and two differential attacks on 18-round by Zhao, Wang, Wang, and Dong [Zha+14] and 19-round by Yang, Hu, Sun, Qiao, Song, Shan, and Ma [Yan+15] out of 20 rounds. Quoting from the specification document [Alb+14, Section 5.5], the related-key attack is out of scope: *"* PRIDE *does not claim any resistance against related-key attacks (and actually can be distinguished trivially in this setting)"*, so the best type of attack appears to be the single-key differential attack.

[Din15] Dinur "Cryptanalytic Time-Memory-Data Tradeoffs for FX-Constructions with Applications to PRINCE and PRIDE"

[Bha+15] Bhaumik, Dutta, Guo, Jean, Mouha, and Nikolić *More Rounds, Less Security?*

[DC14] Dai and Chen *Cryptanalysis of Full PRIDE Block Cipher*

[Zha+14] Zhao, Wang, Wang, and Dong *Differential Analysis on Block Cipher PRIDE*

[Yan+15] Yang, Hu, Sun, Qiao, Song, Shan, and Ma "Improved Differential Analysis of Block Cipher PRIDE"

In this section, an insight on the resistance of PRIDE against differential attack is provided that gives a twofold contribution: first, it is shown that the two previous attacks ([Zha+14] and [Yan+15]) are erroneous – even when taking into account the corrections proposed by Tezcan, Okan, Senol, Dogan, Yücebas, and Baykal [Tez+16] – due to a miscomputation of the known bits, and second, it is shown how to correctly mount a differential cryptanalysis to attack 18 rounds of PRIDE.

[Tez+16] Tezcan, Okan, Senol, Dogan, Yücebas, and Baykal "Differential Attacks on Lightweight Block Ciphers PRESENT, PRIDE, and RECTANGLE Revisited"

The attack presented in this section requires 2^{61} chosen plaintexts and

the equivalent of $2^{63.3}$ encryptions. Since the security claim of the designers is that the product of data and time complexity cannot be smaller than 2^{127}, this proposal is a valid attack of the cipher reduced to 18 rounds.

6.3.1 *Specification*

The cipher follows an SPN structure and benefits from an extensive analysis of secure and efficient linear layers presented in the same article. It is software-oriented and reaches notable performance figures when implemented on 8-bit micro-controllers.

Round Function

PRIDE uses 64-bit blocks and 128-bit keys and makes use of the FX construction in the following way: The first 64 bits of the master key K, denoted by k_0, is used as both pre- and post-whitening keys, while the other half k_1 is used to compute the round keys. In the following, the whitening key is denoted by K_0 and the round key of round i by K_i, $1 \leq i \leq 20$ (see Section 6.3.1).

PRIDE encryption routine is made of 20 rounds. The first 19 rounds are identical and denoted by R, while the last one does not contain the linear layer and is denoted by R′. The cipher ends (resp. starts) with the application of a bit-permutation (resp. its inverse) for bit-sliced implementation reasons. Since these operations can easily be inverted, what we call in the following *plaintext* and *ciphertext* are the states before (resp. after) the first (resp. last) whitening operation. The cipher is based on the following operations, combined as depicted in Figure 6.6:

- The key addition layer, $\mathtt{A}_{K_i}$.
- An S-box layer, which consists of applying the same 4-bit S-box S (given in Table 6.4) to each nibble of the state.
- A linear layer, combining the bit permutations P and $\mathtt{P}^{-1}$ that apply P and P^{-1} bit permutations to the position of the state bits, resp., and the application of matrices, more precisely the multiplication of matrix L_i with $i \in \mathbb{Z}_4$ to the i-th 16-bit word of the state (see Figure 6.6).

TABLE 6.4: The S-box of PRIDE.

x	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
$S(x)$	0	4	8	f	1	5	e	9	2	7	a	c	b	d	6	3

About details of P permutation and L_i matrices, we refer to PRIDE proposal paper [Alb+14].

Actually, the cipher can be seen as an LS-design [Gro+15] that if the state is shown by 2 dimensional 4×16 array of bits that the S-box layer is applied in the first dimension on each 4-bit and in the linear layer, LSB of nibbles are multiplied to matrix L_3, second LSB of nibbles are multiplied to matrix L_2, second MSB of nibbles are multiplied to matrix L_1, and finally, MSB of nibbles are multiplied to matrix L_0.

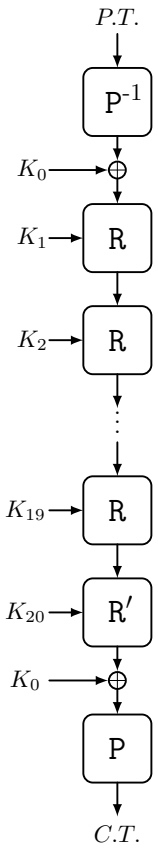


FIGURE 6.5: Overall Structure of PRIDE Block Cipher.

[Gro+15] Grosso, Leurent, Standaert, and Varici, "LS-Designs: Bitslice Encryption for Efficient Masked Software Implementations"

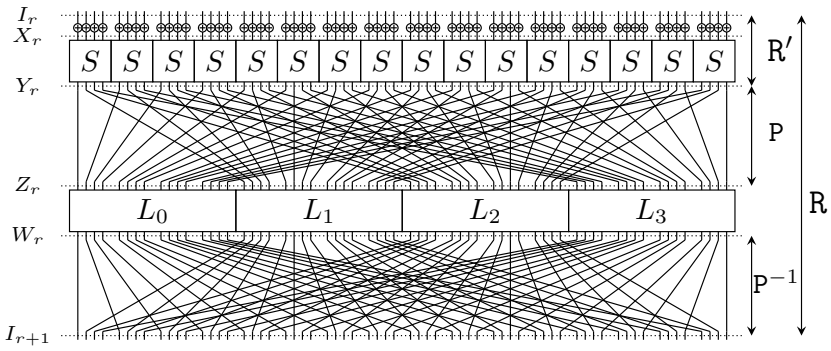


FIGURE 6.6: R and R′ Round Functions of PRIDE.

Key Schedule

In the key schedule of the PRIDE, the key k_1 is considered by its bytes, i. e., $(k_1[0], \ldots, k_1[7]) := \phi_{8,8}^{-1}(k_1)$. Thereby, for each $1 \leq i \leq 20$, the 64-bit round keys are given by $K_i = \mathrm{P}^{-1} \circ f_i(k_1)$ where $f_i(k_1)$ is:

$$f_i(k_1) = \phi_{8,8}\Big(\big(k_1[0], g_i^{(0)}(k_1[1]), k_1[2], g_i^{(1)}(k_1[3]), k_1[4], g_i^{(2)}(k_1[5]), k_1[6], g_i^{(3)}(k_1[7])\big)\Big),$$

and the g_i functions are given by:

$$g_i^{(0)}(x) = (x + 193i) \bmod 256, \qquad g_i^{(1)}(x) = (x + 165i) \bmod 256,$$
$$g_i^{(2)}(x) = (x + 81i) \bmod 256, \qquad g_i^{(3)}(x) = (x + 197i) \bmod 256.$$

Notation of the States

To ease comprehension of the remainder of the section, we use the same notation as in the two previous differential attacks on PRIDE([Zha+14; Yan+15]). These notations are also shown in Figure 6.6, that I_i, X_i, Y_i, Z_i and W_i denote the input state, the state after key addition, the state after the S-box layer, the state after the application of bit permutation P, and the state after the matrices layer of i-th round, resp. In order to remain consistent with it, we also start counting rounds from 1, i. e., I_1 is the plaintext.

6.3.2 *Properties of* PRIDE *Components*

In this section, important properties of the S-box and of the Key Schedule is presented that impact a differential attack of PRIDE. These properties are crucial for understanding the mistakes made in the previous differential cryptanalyses as well as to get the techniques used in the new attack.

S-box Properties

We start by recalling the coordinate functions of the S-box:

Definition 88 (Coordinate Functions of PRIDE S-box)**.** If $x = (x_0, x_1, x_2, x_3)$ is the input nibble of the S-box, then the ANF expressions of the correspond-

ing output nibble $S(x) = y = (y_0, y_1, y_2, y_3)$ is given by:

$$\begin{cases} y_0 = x_0x_1 \oplus x_2 \\ y_1 = x_1x_2 \oplus x_3 \\ y_2 = x_0x_1x_2 \oplus x_0x_1x_3 \oplus x_1x_2 \oplus x_2x_3 \oplus x_0 \\ y_3 = x_0x_1x_3 \oplus x_0x_3 \oplus x_1x_2 \oplus x_2x_3 \oplus x_1 \end{cases}$$

We can remark that y_0 and y_1 depend only on 3 bits out of 4 of the input and that only two of the input bits are involved in the degree 2 monomials. This remark turns useful in the new attack since it implies that instead of requiring a complete nibble to get the value of these two bits, we only need the value of 3 bits. Note that since PRIDE S-box is an involution, these properties also hold for its inverse.

What's more, this observation impacts the possible differential transitions of the S-box, a property formalized by Tezcan [Tez14] and applied to PRIDE in [Tez+16].

[Tez14] Tezcan "Improbable differential attacks on Present using undisturbed bits"

Definition 89 (Undisturbed Bit [Tez14])**.** For a specific input difference of an S-box, if some bits of the output difference remain invariant, we call such bits undisturbed.

For instance, if the input difference of PRIDE S-box is equal to 8 (`1000`), its output difference is of the form ?0??, where '?' represents an unknown bit (see [Tez+16]).

In [Tez14], Tezcan introduced the notion of differential factor, that plays a role in the number of key bits one can recover and on the time complexity:

Definition 90 (Differential Factor [Tez+16])**.** Let S be a function from $\mathbb{F}_2^n$ to $\mathbb{F}_2^m$. For all $x, y \in \mathbb{F}_2^n$ that satisfy $S(x) \oplus S(y) = \mu$, if we also have $S(x \oplus \lambda) \oplus S(y \oplus \lambda) = \mu$, then we say that the S-box has a differential factor λ for the output difference μ, i. e., μ remains invariant for λ)

Key Schedule Properties

Here a property of the key schedule is described that allows us to reduce the number of key-guesses required in our attack:

Lemma 91. *Difference Between Round Keys of* PRIDE *The binary difference between two round keys of* PRIDE K_i *and* K_j *for* i *and* j *of different oddity is given by the following expression, where ? represents an unknown bit:*

$$K_i \oplus K_j = (\texttt{0000 0000 0000 0000 0000 0000 0000 0000} \\ \texttt{???? ???? ???? ???? ???? ???? ???? 1111}).$$

Also, the difference between K_i *and* K_ℓ *for* i *and* ℓ *of the same oddity parity*[2] *is given by:*

$$K_i \oplus K_\ell = (\texttt{0000 0000 0000 0000 0000 0000 0000 0000} \\ \texttt{???? ???? ???? ???? ???? ???? ???? 0000}).$$

[2] Note that simple relations can also be found between other keys; K_i and K_{i+16} for instance.

Proof. The first relation results from the definition of the round key, from which we obtain that

TABLE 6.5: Hexadecimal Value of All the 1- and 2-round Iterative Differential Characteristics of PRIDE.

$I-a$	8000 0000 0000 0000
	0000 8000 0000 0000
	0000 0000 8000 0000
	0000 0000 0000 8000
$I-b$	0800 0000 0000 0000
	0000 0800 0000 0000
	0000 0000 0800 0000
	0000 0000 0000 0800
$I-c$	0080 0000 0000 0000
	0000 0080 0000 0000
	0000 0000 0080 0000
	0000 0000 0000 0080
$I-d$	0008 0000 0000 0000
	0000 0008 0000 0000
	0000 0000 0008 0000
	0000 0000 0000 0008
$II-a$	8000 8000 0000 0000
	8000 0000 8000 0000
	8000 0000 0000 8000
	0000 8000 8000 0000
	0000 8000 0000 8000
	0000 0000 8000 8000
$II-b$	0800 0800 0000 0000
	0800 0000 0800 0000
	0800 0000 0000 0800
	0000 0800 0800 0000
	0000 0800 0000 0800
	0000 0000 0800 0800
$II-c$	0080 0080 0000 0000
	0080 0000 0080 0000
	0080 0000 0000 0080
	0000 0080 0080 0000
	0000 0080 0000 0080
	0000 0000 0080 0080
$II-d$	0008 0008 0000 0000
	0008 0000 0008 0000
	0008 0000 0000 0008
	0000 0008 0008 0000
	0000 0008 0000 0008
	0000 0000 0008 0008
$III-a$	0000 8000 8000 8000
	8000 0000 8000 8000
	8000 8000 0000 8000
	8000 8000 8000 0000
$III-b$	0000 0800 0800 0800
	0800 0000 0800 0800
	0800 0800 0000 0800
	0800 0800 0800 0000
$III-c$	0000 0080 0080 0080
	0080 0000 0080 0080
	0080 0080 0000 0080
	0080 0080 0080 0000
$III-d$	0000 0008 0008 0008
	0008 0000 0008 0008
	0008 0008 0000 0008
	0008 0008 0008 0000

$$K_i \oplus K_j = \mathrm{P}^{-1}\big((\texttt{0000 0000 ???? ???1 0000 0000 ???? ???1}$$
$$\texttt{0000 0000 ???? ???1 0000 0000 ???? ???1})\big)$$

where the differences of 1 in bit 15, 31, 47 and 63 of $P(K_i \oplus K_{i+1})$ are easily explained by the fact that i and j have different oddity and that the values added to k_1 in g functions are odd integers. The second relation results from the fact that i and ℓ have the same oddity. □

As described later, we select our characteristic so that when checking the active S-boxes, we have common bits so less guesses to make.

6.3.3 *Differential Characteristics for* PRIDE

1 and 2-Round Iterative Differential Characteristics

As already shown in [Zha+14; Yan+15], there are 56 high-probability iterative characteristics on 1 and 2 rounds of PRIDE, each activating only 4 S-boxes on 2 rounds whose both input and output differences are equal to 8. Hence, the probability of any of these iterative characteristics is equal to $(2^{-2})^4 = 2^{-8}$. The 56 possible input/output differences are given in Table 6.5, where they are grouped according to the number of active S-boxes in the first round (I, II or III) and to the index of the first active S-box in the input difference (a, b, c and d). Note that all type II characteristics are iterative on 1 round while the others are iterative on 2 rounds.

14-Round Differential Characteristics

Repeating any of the iterative characteristics of Table 6.5 gives a 14-round characteristic of probability 2^{-56}. To find out if there are other 14-round characteristics with similar or better probability, we searched for characteristics with up to 3 active S-boxes in each round. Our program returned 168 (new) 14-round characteristics of probability 2^{-56}. Unfortunately, these characteristics are less advantageous than the iterative ones since when we propagate them with probability 1 in the forward and backward direction, they activate more S-boxes.

Assume then that we use a 14-round characteristic (built from one of Table 6.5) between round 3 and 17 of the cipher. By inverting the linear layer, we compute ΔY_2 from ΔI_3, thus capturing 56 pairs $(\Delta Y_2\ , \Delta X_{17})$ that hold with probability 2^{-56}. In addition to that, a 14-round characteristic defines a limited number of possible differences for ΔI_2 and ΔY_{18} that can be computed from the distribution table of the S-box. Namely, each active S-box of ΔI_2 and ΔY_{17} can only take 4 values, so we obtain that ΔI_2 and ΔY_{17} can respectively take 4^{n_2} and $4^{n_{17}}$ values, where n_i represents the number of active S-boxes in round i.

6.3.4 *Mistake in Previous Analyses*

Two single key differential attacks [Zha+14; Yan+15] have been published prior to our attack. In [Tez+16], Tezcan *et al.* show that the complexities of these attacks are miss-computed due to the oversight of the impact of

differential factor and propose a correction. Their patch mainly increases of the final time complexity.

In this section, we show that there are more problems in [Zha+14; Yan+15] than the ones reported in [Tez+16] and that consequently, the proposed patches are insufficient. The problem we disclose and that is common to both attacks is that the attacker misses information to compute the required internal state bits.

18-Round Differential Attack of Zhao *et al.*

In [Zha+14], Zhao, Wang, Wang, and Dong proposed an attack on 18-round PRIDE. They use a 15-round characteristic[3] of probability 2^{-58} and add one round to the top and two rounds to the bottom. Their attack procedure starts by eliminating some wrong pairs by looking at the ciphertext difference. Then, they guess 10 nibbles of the whitening key K_0 – namely $K_0[0,1,2,4,5,6,9,10,13,14]$ – in order to be able to check that the differences at the input of the corresponding S-boxes of round 18 have the right form (see Table 6.6).

[3] It corresponds to the first characteristic of type $I-a$ in Table 6.5.

They next introduce K'_{18}, a key that is equivalent to the last round key K_{18} and is given by $\mathtt{L}^{-1}(K_{18})$ with $\mathtt{L}$ the linear whole linear layer. They guess on $K'_{18}[5,9,13]$ in order to be able to compute the difference entering S-box number 5, 9, and 13 of penultimate round and access the corresponding sieve.

This attack suffers from several problems: first, as noted in [Yan+15] and later in [Tez+16], the authors omitted to take into account the undisturbed bits. In addition to that, [Tez+16] reveals that the 6 S-box differences involved in the attack are differential factors (namely $\lambda = \mu = 8$), which implies that the attacker cannot obtain information on 6 key bits. Quoting [Tez+16], this error results in the fact that "the correct time complexity of this attack is 2^{70} 18-round PRIDE encryptions, not 2^{66}".

The new problem we spotted is a miscomputation of the known bits of the internal states. Namely: to compute the difference entering S-box number 5, 9, and 13 of the penultimate round, we need the value of $Y_{17}[5,9,13]$, but it is impossible to determine the two middle bits of any of these 3 nibbles. This phenomenon appears clearly if we look at Table 6.6, where we have depicted the bits of rounds 17 that can be computed from the ciphertext given the key guesses on K_0. We see that the 3 highlighted nibbles $Y_{17}[5], Y_{17}[9]$ and $Y_{17}[13]$ are not completely determined.

As it is, the sieve offered by these 3 S-boxes cannot be accessed, so each possible value for the 52 bits of key would be suggested 2^7 times in average, and the right value would not be distinguishable. Consequently, the attack fails.

The attack of Zhao *et al.* has high requirements (2^{60} messages and 2^{66} encryptions), so taking into account the correction from the differential factors already leads to an attack that does not break the security claim ($2^{60} \times 2^{70} > 2^{127}$). In addition to that, correcting the problem we spotted in a straightforward manner would require making more guesses on K_0, that is, to guess 22 bits of the nibbles $K_0[3,7,8,11,12,15]$, so clearly fixing Zhao *et al.* paper does not lead to an attack that threatens the cipher.

TABLE 6.6: Analysis of 18-round differential attack of PRIDE by Zhao *et al.* [Zha+14]. All the bit values that are computable are depicted with 1, while other bits are shown by 0.

$P = I_1$	1111 1111 1111 1111 1111 1111 1111 1111 1111 1111 1111 1111 1111 1111 1111 1111
$K_0 \oplus K_1$	0000 0000 0000 0000 0000 1111 0000 0000 0000 1111 0000 0000 0000 1111 0000 0000
X_1	0000 0000 0000 0000 0000 1111 0000 0000 0000 1111 0000 0000 0000 1111 0000 0000
Y_1	0000 0000 0000 0000 0000 1111 0000 0000 0000 1111 0000 0000 0000 1111 0000 0000
...	...
X_{17}	0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
K'_{18}	0000 0000 0000 0000 0000 1111 0000 0000 0000 1111 0000 0000 0000 1111 0000 0000
Y_{17}	0000 1001 1001 0110 0000 **1001** 1001 0110 0000 **1001** 1111 0000 0000 **1001** 1111 0000
Z_{17}	0110 0110 0110 0110 0001 0001 0010 0010 0001 0001 0010 0010 0110 0110 0110 0110
W_{17}	1110 1110 0110 0110 1110 1110 0110 0110 1110 1110 0110 0110 1110 1110 0110 0110
X_{18}	1111 1111 1111 0000 1111 1111 1111 0000 0000 1111 1111 0000 0000 1111 1111 0000
Y_{18}	1111 1111 1111 0000 1111 1111 1111 0000 0000 1111 1111 0000 0000 1111 1111 0000
K_0	1111 1111 1111 0000 1111 1111 1111 0000 0000 1111 1111 0000 0000 1111 1111 0000
C	1111 1111 1111 1111 1111 1111 1111 1111 1111 1111 1111 1111 1111 1111 1111 1111

The authors' confusion comes probably from the fact that the linear layer is not an involution. In the case it was, knowing the bits they name would have been enough to access the active S-boxes in round 17. Unfortunately, L_1 and L_2 do not define involutions, so more bits are required to find the output of the 3 active S-boxes of round 17.

19-Round Differential Attack by Yang *et al.*

There is a similar mistake in the 19-round attack described by Yang, Hu, Sun, Qiao, Song, Shan, and Ma in [Yan+15]. They use a 15-round characteristic[4] and expand it two rounds to the plaintext side and two rounds to the ciphertext side. After discarding pairs that for sure do not follow the characteristic, they guess nine nibbles of key in plaintext side, namely $(K_0 \oplus K_1)[0, 1, 2, 4, 6, 8, 9, 12, 13]$, and partially encrypt the plaintext pairs through the first S-box layer. On the ciphertext side, they guess the seven nibbles $0, 1, 4, 7, 8, 9$ and 12 of K_0 and partially decrypt the ciphertext pairs through the last S-box layer. They next claim that they can also recover nibbles number 4 and 8 of K_2 and nibbles 4 and 8 of $K'_{19} = \mathrm{L}^{-1}(K_{19})$.

[4]The fourth characteristic of type $II - a$ given in Table 6.6.

Similarly to the 18-round attack discussed previously and as described in [Tez+16], this attack uses difference transitions that are differential factors, meaning that it fails to recover 4 bits of the key (each most significant bit of nibbles number 4 and 8 of K_2 and nibbles 4 and 8 of K'_{19}).

Moreover, as in the previous section, we note that there is a problem upstream to that one. Namely, the authors cannot compute the desired S-box transitions given their guesses: They lack information to compute the two middle bits of $I_2[4, 8]$ and $Y_{18}[4, 8]$. Consequently, they cannot obtain information on these four nibbles, and the right key cannot be identified (even if the correction given by Tezcan *et al.* is applied).

The detail of which bits are computable in the first and last two rounds is provided in Table 6.7.

The initial attack requires 2^{62} chosen plaintexts and 2^{63} 19-round encryptions. To correct the errors we spotted, it is necessary to significantly increase the number of key guesses. We need the value of 24 bits of nibbles number 3, 4, 7, 11, 12, 15, 16 of K_0 to be able to treat the 2 S-boxes of the penultimate round, while we need the value of $(K_0 \oplus K_1)[11, 15]$ and 3 bits of $(K_0 \oplus K_1)[5]$ to access the 2 S-boxes of round 2. Clearly, the straightfor-

TABLE 6.7: Analysis of 19-round Differential Attack of PRIDE by Yang *et al.* [Yan+15]. We use the same notation as before and depict known bits with 1 and 0.

$P = I_1$	1111	1111	1111	1111	1111	1111	1111	1111	1111	1111	1111	1111	1111	1111	1111	1111
$K_0 \oplus K_1$	1111	1111	1111	0000	1111	0000	1111	0000	1111	1111	0000	0000	1111	1111	0000	0000
X_1	1111	1111	1111	0000	1111	0000	1111	0000	1111	1111	0000	0000	1111	1111	0000	0000
Y_1	1111	1111	1111	0000	1111	0000	1111	0000	1111	1111	0000	0000	1111	1111	0000	0000
Z_1	1110	1010	1100	1100	1110	1010	1100	1100	1110	1010	1100	1100	1110	1010	1100	1100
W_1	1000	1100	1000	1000	0100	0000	0100	0000	0000	0100	0000	0100	1000	1000	1000	1100
I_2	1001	0100	0000	0000	1001	1010	0000	0000	1001	0100	0000	0000	1001	0011	0000	0000
K_2	0000	0000	0000	0000	1111	0000	0000	0000	1111	0000	0000	0000	0000	0000	0000	0000
X_2	0000	0000	0000	0000	**1001**	0000	0000	0000	**1001**	0000	0000	0000	0000	0000	0000	0000
Y_2	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
...									...							
X_{18}	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
K'_{19}	0000	0000	0000	0000	1111	0000	0000	0000	1111	0000	0000	0000	0000	0000	0000	0000
Y_{18}	1001	0000	0100	0000	**1001**	0000	0010	0000	**1001**	0010	0000	0000	1001	0100	0000	0000
Z_{18}	1000	1000	1000	1000	0010	0000	0000	0100	0000	0010	0100	0000	1000	1000	1000	1000
W_{18}	1100	1001	1100	1000	1100	1001	1100	1000	1100	1001	1100	1000	1100	1001	1100	1000
X_{19}	1111	1111	0000	0000	1111	0000	0000	1111	1111	1111	0000	0000	1111	0000	0000	0000
Y_{19}	1111	1111	0000	0000	1111	0000	0000	1111	1111	1111	0000	0000	1111	0000	0000	0000
K_0	1111	1111	0000	0000	1111	0000	0000	1111	1111	1111	0000	0000	1111	0000	0000	0000
C	1111	1111	1111	1111	1111	1111	1111	1111	1111	1111	1111	1111	1111	1111	1111	1111

ward correction does not lead to a correct attack since the time complexity explodes.

6.3.5 *Key Recovery Attacks*

This section describes our 18-round differential cryptanalysis of PRIDE. We start by exposing a differential property of PRIDES-box and then show how to use it in an attack to find information on key bits. We then detail the complexities of our attack.

PRIDES-box Properties for our Differential Characteristics

As discussed in Section 6.3.3, the difference transitions made by the S-boxes of round 2 and 17 are either from 8 to 2, 3, 8 or a or from 2, 3, 8 or a to 8. For these configurations, the following property holds:

Lemma 92. *If two inputs of* PRIDE *S-box differ by 2 (resp. 3, 8 or a) and lead to an output difference of 8 then the following relations hold:*

$$\begin{aligned}
S(x) \oplus S(x \oplus 2) = S(x_0x_1x_2x_3) \oplus S(x_0x_1\bar{x}_2x_3) = 8 \quad &\Rightarrow \quad x_1 = 0, x_3 = 0 \\
S(x) \oplus S(x \oplus 3) = S(x_0x_1x_2x_3) \oplus S(x_0x_1\bar{x}_2\bar{x}_3) = 8 \quad &\Rightarrow \quad x_1 = 1, x_2 = x_3 \\
S(x) \oplus S(x \oplus 8) = S(x_0x_1x_2x_3) \oplus S(\bar{x}_0x_1x_2x_3) = 8 \quad &\Rightarrow \quad x_1 = 1, x_2 = \bar{x}_3 \\
S(x) \oplus S(x \oplus a) = S(x_0x_1x_2x_3) \oplus S(\bar{x}_0x_1\bar{x}_2x_3) = 8 \quad &\Rightarrow \quad x_1 = 0, x_3 = 1
\end{aligned}$$

Proof. The property results from using the ANF of the coordinate functions given in Definition 88. □

This means if we can check that the input difference of an active S-box is 2, 3, 8 or a and if we expect its output difference to be equal to 8 then we can deduce information on the value of the state entering this S-box. Namely, we obtain the value of x_1 together with either the value of x_3 or a relation between x_3 and x_2.

This observation implies that to check if an S-box executes the right transition (so to have access to the corresponding filter of probability 2^{-2}), we only require information on (at most) 3 bits (bits 1, 2, and 3). This can be seen as a reinterpretation of the undisturbed bits of [Tez14].

Overview of the Attack Procedure

In our attack, we use the 14-round characteristic of type $II-d$ given in the fifth row the same type in of Table 6.5,[5] and extend it 2 rounds to the plaintext and 2 rounds to the ciphertext. This extension defines 4 rounds of key recovery that will allow us to recover 10 bits of key on the plaintext side and 10 bits of key on the ciphertext side.

[5] Note that this characteristic is iterative on 1 round.

The reasons why we decided to use this particular characteristic is described in the following.

- Probability of the 14-round differential path: 2^{-p}
- Number of active S-boxes in round i: n_i
- Probability that a pair of messages of the same structure leads to the correct difference: 2^{-4n_1}
- Number of pairs needed to get around a right pairs: $a \cdot 2^{p+4n_1}$
- Number of encryptions needed to obtain a full structure: 2^{4n_1}
- Number of pairs that can be made with one structure: 2^{8n_1-1}
- Number of structures required to get enough pairs: $a \cdot 2^{p-4n_1+1}$
- Number of encryptions required to get enough pairs: $a \cdot 2^{p+1}$

From this, we can see that the number of encryptions needed to conduct the attack only depends on p, so all the 14-round characteristics with maximal probability (with $p = 56$) lead to the same data complexity.

Among these, we want to select the characteristic that leads to the minimal time complexity. The parameters that impact it are first the number of active S-boxes on the plaintext and ciphertext side (n_P and n_C) but also the number of possible differences in plaintext and ciphertext ($N_{\Delta P}$ and $N_{\Delta C}$), and the number of key bits that we need to guess ($|K_P|$ and $|K_c|$). Table 6.8 gives a comparison of the best probability characteristics of minimal $n_P + n_C$.

To find out which one is the best for our attack, we first make the following observations: First, among the 224 (168 new and 56 previously found) characteristics, we prefer the ones that imply less active S-boxes on the plaintext and ciphertext side and, consequently, require fewer guesses and computations checking that first and last round transitions are correct. This downsizes the set of candidate characteristics to 24 (8 of each type), each activating a total of 14 S-boxes on the plaintext and ciphertext side. As can be seen in Table 6.8, type I characteristics activate 9 S-boxes on plaintext side and 5 S-boxes on ciphertext side while for type II we have 7 active S-boxes on each side, and for type III the distribution is of 6 active S-boxes on plaintext side and 8 on ciphertext side.

Then, among the 24 possible characteristics, we prefer the ones that lead to a smaller amount of possible differences when extending the characteristic with probability 1 in plaintext and ciphertext. We also take into account the number of key bits that we need to guess.

When looking for minimizing these parameters, both of the characteristics of type I and type II seem good. Type I characteristics require 1 more

TABLE 6.8: Comparison of 14-round Characteristics with Minimal $n_P + n_C$.

Type	$\Delta I_t = \Delta I_{t+14}$	n_P	n_C	$N_{\Delta P}$	$N_{\Delta C}$	$\lvert K_P\rvert$	$\lvert K_C\rvert$
I	8000 0000 0000 0000	9	5	$2^{24.86}$	$2^{10.97}$	53	27
	0800 0000 0000 0000					53	
	0080 0000 0000 0000					53	
	0008 0000 0000 0000					50	
	0000 8000 0000 0000					53	
	0000 0800 0000 0000					53	
	0000 0080 0000 0000					53	
	0000 0008 0000 0000					50	
II	8000 0000 8000 0000	7	7	$2^{17.38}$	$2^{19.93}$	40	38
	0800 0000 0800 0000					40	
	0080 0000 0080 0000					40	
	0008 0000 0008 0000					40	
	0000 8000 0000 8000					40	
	0000 0800 0000 0800					40	
	0000 0080 0000 0080					40	
	0000 0008 0000 0008					38	
III	8000 8000 0000 8000	6	8	$2^{14.10}$	$2^{24.33}$	35	45
	0800 0800 0000 0800					35	
	0080 0080 0000 0080					35	
	0008 0008 0000 0008					35	
	8000 8000 8000 0000					35	
	0800 0800 0800 0000					35	
	0080 0080 0080 0000					35	
	0008 0008 0008 0000					33	

key bit guess but lead to fewer possible differences in plaintext and ciphertext. Eventually, we prefer characteristics of type II since the memory size required to store a full structure is more reasonable.

For the selected characteristic, and as explained in Section 6.3.3, ΔX_2 can take $4^2 = 16$ possible values, implying that there are also 16 possible values for ΔY_1 (these values are given in Table 6.10).

As shown in Table 6.9, we have 7 active nibbles for the 16 possible differences of ΔY_1 (nibbles 3, 4, 5, 7, 8, 11, and 15) while the other 9 nibbles are always inactive. We use this property to reduce the necessary amount of data by building data structures: The messages we ask for encryption are organized as sets of $2^{4 \cdot 7} = 2^{28}$ plaintexts that are all equal in the corresponding 9 inactive nibbles and take all possible values in the above mentioned 7 nibbles. So in each structure, there is about $2^{8 \cdot 7 - 1} = 2^{55}$ plaintext pairs that differ in the 7 nibbles of interest.

TABLE 6.9: Extension of the 14-round Characteristic used in Our Attack.

ΔP	0000	0000	0000	????	????	????	0000	????	????	0000	0000	????	0000	0000	0000	????
ΔY_1	0000	0000	0000	?00?	00?0	00?0	0000	?00?	00?0	0000	0000	?00?	0000	0000	0000	?0??
ΔZ_1	000?	000?	000?	000?	0000	0000	0000	0000	0000	??00	?000	000?	000?	000?	000?	000?
ΔW_1	0000	000?	0000	000?	0000	0000	0000	0000	0000	000?	0000	000?	0000	000?	0000	000?
ΔX_2	0000	0000	0000	0000	0000	0000	0000	?0??	0000	0000	0000	0000	0000	0000	0000	?0??
ΔY_2	0000	0000	0000	0000	0000	0000	0000	1000	0000	0000	0000	0000	0000	0000	0000	1000
...								...								
ΔX_{17}	0000	0000	0000	0000	0000	0000	0000	1000	0000	0000	0000	0000	0000	0000	0000	1000
ΔY_{17}	0000	0000	0000	0000	0000	0000	0000	?0??	0000	0000	0000	0000	0000	0000	0000	?0??
ΔZ_{17}	0000	000?	0000	000?	0000	0000	0000	0000	0000	000?	0000	000?	0000	000?	0000	000?
ΔW_{17}	000?	000?	000?	000?	0000	0000	0000	0000	?0??	0000	?00?	000?	000?	000?	000?	000?
ΔX_{18}	00?0	0000	00?0	?0??	0000	0000	0000	?00?	00?0	0000	0000	?0??	0000	0000	0000	?0??
ΔC	????	0000	????	????	0000	0000	0000	????	????	0000	0000	????	0000	0000	0000	????

TABLE 6.10: Possible ΔY_1 and ΔX_{18} Values.

$\Delta X_2[7,15]$	ΔY_1
22	0000 2000 0000 0002
23	0001 2000 0001 0003
28	0008 2208 2008 0000
2a	0008 2008 0008 0002
32	0001 2001 0001 0002
33	0000 2001 0000 0003
38	0009 2209 2009 0000
3a	0009 2009 0009 0002
82	0008 0200 2008 000a
83	0009 0200 2009 000b
88	0000 0008 0000 0008
8a	0000 0208 2000 000a
a2	0008 2000 0008 000a
a3	0009 2000 0009 000b
a8	0000 2208 2000 0008
aa	0000 2008 0000 000a

$\Delta Y_{17}[7,15]$	ΔX_{18}
22	2022 0000 2002 0002
23	2023 0000 2003 0003
28	002a 0008 000a 0000
2a	202a 0008 200a 0002
32	2023 0001 2003 0002
33	2022 0001 2002 0003
38	002b 0009 000b 0000
3a	202b 0009 200b 0002
82	2008 0000 2008 000a
83	2009 0000 2009 000b
88	0000 0008 0000 0008
8a	2000 0008 2000 000a
a2	202a 0000 200a 000a
a3	202b 0000 200b 000b
a8	0022 0008 0002 0008
aa	2022 0008 2002 000a

A pair from this structure has the correct difference in ΔY_2 if it reaches one of the 16 targeted values for ΔY_1 and then makes happen the correct transitions of S-box 8 and 16 of the S-box layer of round 2. The probability of this event can be computed as follows.

PROBABILITY THAT A PAIR TAKES ONE OF THE 16 TARGETED VALUES IN ΔY_1: Consider a complete structure that is the set made by all the messages with the same – fixed – value on our 9 inactive nibbles and taking all possible values on the other 7 nibbles. There are exactly 2^{28} such plaintexts. Since the key addition step defines a permutation, we still have 2^{28} messages differing on 7 nibbles after the addition of the whitening key K_0 and the first round key K_1. The same reasoning applies to the S-box layer: The messages in our structure take all possible values at the input of 7 S-boxes, so since PRIDE S-box is a permutation, the images of these messages still correspond to 2^{28} messages differing on the same positions. Consequently, when forming pairs of these messages, every possible non-null difference on the 7 nibbles appears 2^{27} times. So in one structure, exactly 16×2^{27} pairs out of 2^{55} are useful for our attack (i. e., a ratio of 1 out of 2^{24}).

PROBABILITY THAT ONE OF THE 16 DIFFERENCES IN ΔY_1 LEADS TO THE CORRECT ΔY_2: If ΔY_1 is as required, the probability that the second round leads to the desired characteristic is equal to $(2^{-2})^2 = 2^{-4}$, which corresponds to the probability that the two active S-boxes of round 2 output a difference of 8 given an entering difference of 2, 3, 8 or a.

In sum, the total probability that one of our pairs follows the characteristic is equal to:

$$2^{-24} \cdot 2^{-4} \cdot 2^{-56} = 2^{-84}$$

which corresponds to realizing the correct transitions in rounds 1 and 2, following the 14-round characteristic[6] and finally propagating with probability 1 in the last 2 rounds.

[6] Our experiments for up to 7 rounds showed that the probability of the differential matches the one of the characteristic.

This indicates that we need to encrypt about $a \cdot 2^{84}$ plaintext pairs in order to obtain a pairs that follow the characteristic (also called *right pairs*). This amount can be obtained with $a \cdot 2^{84} \cdot 2^{-55} = a \cdot 2^{29}$ data structures i. e., with $a \cdot 2^{57}$ chosen plaintexts.

In the forward extension, there are $4^2 = 16$ possible values for ΔY_{17}, so there are 16 possible values for ΔX_{18} (see Table 6.10). As shown in Table 6.9, these 16 possible values for ΔX_{16} define at most 7 active nibbles (nibbles 0, 2, 3, 7, 8, 11 and 15) while other 9 nibbles are always inactive. A common technique to filter out wrong pairs consists in discarding pairs that have active S-boxes at any of these 9 positions. Given our harsh restrictions in terms of time complexity, we prefer considering a stronger filter which consists of checking that the difference observed in both plaintext and ciphertext differences are consistent with the 16 possible differences that can take Y_1 and X_{18}.

Once we generated enough messages and filtered them according to plaintext and ciphertext differences, we start making key guesses: we test a pair together with a possible value for the key by making partial encryptions and checking that the necessary conditions are fulfilled. We discard all the candidates that do not follow the characteristic.

We start by considering the ciphertext side and make a guess on the seven nibbles $K_0[0,2,3,7,8,11,15]$. We partially decrypt each of the pairs through the matching seven nibbles of the last S-box layer and look at the difference that we obtain: any candidate which difference is not one of the previously computed 16 possible values for ΔX_{18} is discarded.

We follow a similar procedure in the plaintext side: we guess the 28 key bits that intervene in the computation of the 7 active S-boxes ($(K_0 \oplus K_1)[3,4,5,7,8,11,15]$) and partially encrypt the corresponding nibbles. If the obtained difference is one of the 16 precomputed ones, we keep the candidate as possible; otherwise we discard it.

At this point, each pair is associated with $28 + 28 = 56$ bits of key corresponding to $(K_0 \oplus K_1)[3,4,5,7,8,11,15]$ and $K_0[0,2,3,7,8,11,15]$. From these possible values for parts of $(K_0 \oplus K_1)$ and of K_0, we deduce possible values for $K_1[3,7,8,11,15]$. In addition to that, Lemma 91 implies that we can deduce nibble 3, 7, and 15 of any round key K_i.

We now have a look at the S-box layer of round 2. We know the value of the differences entering S-box 7 and 15, together with the value of $K_2[7,15]$. To check if the S-boxes execute the right transitions, we lack the value of the two middle bits of nibble $I_2[7]$ and $I_2[15]$. By inverting the linear layer, we can see that these values depend on the values of 3 unknown bits, which are bit 1, 41, and 58 of state Y_1 (see Figure 6.7). The ANF description of the S-box (see Definition 88) indicates that the values of these 3 bits depend on 10 bits of the plaintext (which is known), together with 10 key bits: $(K_0 \oplus K_1)[0][1,2,3]$, $(K_0 \oplus K_1)[10][1,2,3]$ and $(K_0 \oplus K_1)[14]$, resp. Consequently, the idea would be to guess these 10 key bits, deduce the value of $X_2[7]$ and $X_2[15]$, and check whether the transitions are satisfied or not. The probability that a guess passes this test is 2^{-4}.

We follow a similar procedure to handle the last 2 rounds. Let us recall here that our 14-round characteristic ends at round 16 and that the difference spreads freely in rounds 17 and 18, which are of type R and R′, resp. Our goal here is to check the transitions of S-box 7 and 15 of round 17 by using Lemma 92. To limit the complexity of this step, we only check that the value of x_1 is correct instead of checking both relations, so we are only interested in $Y_{17}[7][1]$ and $Y_{17}[15][1]$ (denoted by c_1 and d_1 in Figure 6.8). By referring to the linear layer, we obtain that their expressions in the function of I_{18} are:

$$\begin{cases} Y_{17}[7][1] = I_{18}[5][1] \oplus I_{18}[6][1] \oplus I18[13][1], \\ Y_{17}[15][1] = I_{18}[2][1] \oplus I_{18}[10][1] \oplus I18[11][1]. \end{cases}$$

The value of I_{18} depends on ciphertext bits (state C) together with bits of K_0 and K_{18}. For instance, $Y_{17}[7][1]$ can be rewritten as:

$$\begin{aligned}
&= K_{18}[5][1] \oplus X_{18}[5][1] \oplus K_{18}[6][1] \oplus X_{18}[6][1] \oplus K_{18}[13][1] \oplus X_{18}[13][1] \\
&= K_{18}[5][1] \oplus K_{18}[6][1] \oplus K_{18}[13][1] \oplus Y_{18}[5][3] \oplus Y_{18}[5][2] Y_{18}[5][1] \oplus \\
&\quad Y_{18}[6][3] \oplus Y_{18}[6][2] \cdot Y_{18}[6][1] \oplus Y_{18}[13][3] \oplus Y_{18}[13][2] \cdot Y_{18}[13][1] \\
&= K_{18}[5][1] \oplus K_{18}[6][1] \oplus K_{18}[13][1] \oplus (C[5][3] \oplus K_0[5][3]) \oplus \\
&\quad (C[5][2] \oplus K_0[5][2]) \cdot (C[5][1] \oplus K_0[5][1]) \oplus (C[6][3] \oplus K_0[6][3]) \oplus \\
&\quad (C[6][2] \oplus K_0[6][2]) \cdot (C[6][1] \oplus K_0[6][1]) \oplus (C[13][3] \oplus K_0[13][3]) \oplus \\
&\quad (C[13][2] \oplus K_0[13][2]) \cdot (C[13][1] \oplus K_0[13][1])).
\end{aligned}$$

This indicates that we need to make a guess on 7 bits of $K_0[5,6,13][1,2]$ and $K_{18}[5][1] \oplus K_{18}[6][1] \oplus K_{18}[13][1] \oplus K_0[5][3] \oplus K_0[6][3] \oplus K_0[13][3]$. We follow a similar procedure for $Y_{17}[15][1]$ and conclude that we need to guess another 3 bits, namely $K_0[10][3] \oplus K_{18}[2][1] \oplus K_{18}[10][1] \oplus K_{18}[11][1]$ and $K_0[10][1,2]$. To sum up the key guessing process, we started from a set of $a \cdot 2^{84}$ possible pairs, we guessed $28+28+10+10 = 76$ key bits and we had access to a filter of $2^{-36} \cdot 2^{-24} \cdot 2^{-24} \cdot 2^{-4} \cdot 2^{-2} = 2^{-54}$ (which corresponds to filtering on the ciphertext difference, checking last and first round and finally second and seventeenth rounds, resp.). The number of candidates remaining in the last step is then equal to $a \cdot 2^{70}$. So, on average, each of the 76-bit key candidates will be counted $a \cdot 2^{70} \cdot 2^{-76} = a \cdot 2^{-6}$ times, while as we expect to have a right pairs, the right key candidate will be counted a times. The *signal to noise ratio* (S/N) will then be equal to 2^6, which ensures that we can distinguish the right key candidate from the wrong ones.

The last step of the attack consists of doing an exhaustive search to find the correct value for the remaining $128-76 = 52$ key bits.

Detailed Description of the Attack and of its Complexities

Here, we detail the time, data, and memory complexities of our attack. We show that a naive implementation of the attack procedure described in Section 6.3.5 would lead to a time complexity overrun and show how to deal with this issue.

DATA COMPLEXITY: As detailed previously, we need about $a \cdot 2^{57}$ chosen plaintexts in order to successfully achieve the attack. We choose $a = 2^4$, which means that the data complexity of our attack is equal to 2^{61}. We recall that the security provided by PRIDE when the attacker has access to 2^d messages is equal to 2^{127-d}. Since our attack requires 2^{61} messages, we are limited to a number of operations lower than 2^{66} encryptions.

TIME COMPLEXITY: To summarize the process described in Section 6.3.5, the attack is made of 6 main steps:

1. Encrypt 2^{33} structures and filter wrong pairs by looking at their input and output differences.
2. Make a guess on 28 bits of K_0 and check the transitions of the active S-boxes of last round. Eliminate wrong candidates (that are associations of a pair with a key value that do not satisfy the transitions).
3. Guess 28 bits of $K_0 \oplus K_1$ and check the transitions of the active S-boxes of first round. Eliminate wrong candidates.
4. Guess 10 bits of $K_0 \oplus K_1$ to access the value entering S-boxes number 7 and 15 of round 2 and check their transitions.
5. Guess 10 bits of K_0 and K_{18} to access the value outputting S-boxes number 7 and 15 of round 17 and check their transitions.
6. The key guess that is suggested the most is the correct one. Guess the remaining 52 key bits and do trial encryptions to recover the 128-bit master key.

A naive implementation of this process would lead to several problems. First, the attack involves many key bit guesses and uses many pairs, which would make the time complexity exceed our upper bound of 2^{66} PRIDE encryptions as soon as step 3. Second, detecting which key candidate is the most frequent would require keeping track of 2^{76} counters, which is clearly not reasonable.

As described next, we solve those two problems by making small guesses at the time and by studying each possible key guess for all possible pairs instead of studying each pair one after the other with all the possible key candidates.

FIRST 3 STEPS OF THE ATTACK As briefly mentioned in Section 6.3.5, the first step of the attack consists of filtering the 2^{88} pairs of messages by looking at their plaintext and ciphertext differences.

Starting from the known 16 possible differences in ΔY_1 and ΔX_{18}, we refer to the difference distribution table and precompute the possible differences in P and C. A search returns that ΔP can take $170\,164 = 2^{17.38}$ values while ΔC can take $999\,448 = 2^{19.93}$ values. This implies that out of the 2^{88} initial pairs of messages only $2^{88} \cdot (2^{17.38} \cdot 2^{-28}) \cdot (2^{19.93} \cdot 2^{-64}) = 2^{33.31}$ pairs will remain.

In practice, we start by filtering pairs according to the truncated difference in the ciphertext. We are left with 2^{52} pairs whose differences on the plaintext and ciphertext sides are only on (at most) 7 nibbles. We then build two tables of 2^{28} bits each: the first table indicates if a difference on 28 bits is possible in the plaintext side (so contains a 1 at the position corresponding to the $2^{17.38}$ possible ΔP), while the second indicates which 28-bit differences are valid on the ciphertext side. Each of the 2^{52} remaining pairs then requires at most two table lookup to be filtered.

Then, we store all these $2^{33.31}$ pairs and evaluate them with all possible key values. This change implies that instead of needing 2^{76} counters, we have to save the $2^{33.31}$ pairs (so we require $2^{35.31}$ blocks of 64 bits).

In step 2, we start by guessing 7 nibbles of K_0 ($K_0[0,2,3,7,8,11,15]$). For each possible value, we study the $2^{33.31}$ pairs and cancel the ones that do not fulfill the required conditions. More precisely, the key guess is used to invert last round S-box layer and compute the difference ΔX_{18}. We only keep pairs that lead to one of the 16 possible differences given in Table 6.10. Since there are 2^{28} possible values for the 7 key nibbles and that we repeat these operations for each of the $2^{33.31}$ pairs, this step is made $2^{61.31}$ times. To express this complexity in terms of PRIDE encryptions, we can see that it consists in computing two times 7 S-boxes (for a pair), while one full encryption with the cipher requires $18 \cdot 16 = 288 = 2^{8.17}$ S-box computations. Step 2 is then roughly equivalent to $2^{56.95}$ PRIDE encryptions.

Step 3 consists of the same operations as step 2 but in the plaintext side. We guess the 7 nibbles $(K_0 \oplus K_1)[3,4,5,7,8,11,15]$ and compute the corresponding 7 S-boxes of round 1 for all the remaining pairs. The pairs that are processed in this step correspond to the pairs that remain after step 2, that is, on average $2^{33.31} \cdot 16 \cdot 2^{-19.93} = 2^{17.38}$ pairs associated to each possible value for the 28 bits of key. In its naive form, the number of PRIDE encryptions made in this step would then be equal to $2^{28} \cdot 2^{28} \cdot 2^{17.38} \cdot 7 \cdot 2 \cdot 2^{-8.17} = 2^{69.02}$, which

exceeds our limit of 2^{66} PRIDE encryptions. To solve this problem, we encrypt one S-box after the other and immediately check if the conditions are fulfilled. We start with S-box number 5, for which according to Table 6.10 the targeted output difference is 2. Given one of the 2^4 possible values for $(K_0 \oplus K_1)[4]$ fixed, we compute $Y_1[4]$ (this requires a total of $2^{28} \cdot 2^4 \cdot 2^{17.38} \cdot 2 = 2^{50.38}$ S-box operations) and check that the obtained difference is equal to 2. To compute the number of pairs that pass this test, we need to take into account the proportion of pairs for which S-box number 5 is active (equal to $\frac{152\,004}{170\,164} = 2^{-0.16}$) together with the probability that an active S-box of our pre-filtered set leads to a difference of 2 (which is $\frac{1}{6}$). We obtain the following estimate:

$$2^{17.38} \cdot \left(\frac{152\,004}{170\,164} \cdot \frac{1}{6} + \frac{18\,160}{170\,164} \cdot 1\right) = 2^{17.38} \cdot 2^{-1.97} = 2^{15.41}.$$

In the same way, we make a guess on the 4 bits of $(K_0 \oplus K_1)[5]$, which requires $2^{28} \cdot 2^4 \cdot 2^4 \cdot 2^{15.41} \cdot 2 = 2^{52.41}$ S-box encryptions. We then filter out wrong pairs by checking that active S-boxes give a difference of 2. The number of remaining pairs is then equal to $2^{13.37}$. We then process S-box number 8, which requires $2^{28} \cdot (2^4)^3 \cdot 2^{13.37} \cdot 2 = 2^{54.37}$ S-box encryptions and leaves us with an average of $2^{12.33}$ pairs for each partial key guess. Next, we treat S-boxes number 3 and 11, taking advantage of the fact that they must have the same output difference. The number of S-box encryptions is equal to $2^{61.33}$ and $2^{8.73}$ pairs remain on average. We finally handle the last 2 S-boxes together, which requires $2^{65.73}$ S-box encryptions and discard all but 2^4 pairs in average for each key candidate ($2^{17.38} \cdot (16 \cdot 2^{-17.38}) = 2^4$). To sum up, total time complexity of this step is $2^{50.38} + 2^{52.41} + 2^{54.37} + 2^{61.33} + 2^{65.73} = 2^{65.80}$ which is equivalent to $2^{66.80-8.17} = 2^{57.63}$ PRIDE encryptions.

STEP 4 AND 5 The next operations (step 4 and 5) consist in checking the transitions of S-box number 7 and 15 in round 2 and in round 17. As explained before, we look at the value of only 3 out of their 4 input bits in round 2 and only 1 out of 4 output bits in round 17. In the following, we name these bits a_i, b_i, $(1 \leq i \leq 3)$ and c_1, d_1, resp. (see Figures 6.7 and 6.8).

We start by explaining how to check the 2 active S-boxes of round 2. We remark here that if a_1 (resp. b_1) defines a condition on its own, the condition that a_3 (resp. b_3) must fulfill sometimes depends on a_2 (resp. b_2).

As briefly mentioned in Section 6.3.5 and as illustrated in Figure 6.7, a_1 and b_1 are given by the following two expressions:

$$\begin{aligned} a_1 &= K_2[7][1] \oplus Y_1[0][1] \oplus Y_1[7][1] \oplus Y_1[10][1], \\ b_1 &= K_2[15][1] \oplus Y_1[7][1] \oplus Y_1[10][1] \oplus Y_1[11][1], \end{aligned}$$

that when referring to Definition 88 can be rewritten as:

$$\begin{aligned} a_1 = {} & P[0][3] \oplus P[10][3] \oplus Y_1[7][1] \oplus \\ & K_2[7][1] \oplus (K_0 \oplus K_1)[0][3] \oplus (K_0 \oplus K_1)[10][3] \oplus \\ & (P[0][2] \oplus (K_0 \oplus K_1)[0][2]) \cdot (P[0][1] \oplus (K_0 \oplus K_1)[0][1]) \oplus \\ & \big(P[10][2] \oplus (K_0 \oplus K_1)[10][2]\big) \cdot \big(P[10][1] \oplus (K_0 \oplus K_1)[10][1]\big), \end{aligned} \tag{6.1}$$

$$b_1 = K_2[15][1] \oplus (K_0 \oplus K_1)[10][3] \oplus P[10][3] \oplus Y_1[7][1] \oplus Y_1[11][1] \oplus \\ \big(P[10][2] \oplus (K_0 \oplus K_1)[10][2]\big) \cdot \big(P[10][1] \oplus (K_0 \oplus K_1)[10][1]\big), \tag{6.2}$$

for which the only unknown bits are $(K_0 \oplus K_1)[0, 10][1, 2, 3]$. Indeed, the plaintext bits are known ,and $K_2[7, 15][1]$ are deduced from key-schedule properties, while a_1 and b_1 are determined by the difference observed in X_2 together with the relations given by Lemma 92.

Consequently, we guess these 6 key bits and check that the relations given by Eqs. (6.1) and (6.2) hold, which happens with probability 2^{-2}.

Since we have an average of 2^4 candidates for each possibility for the 56 bits of key guessed so far, this step is repeated $2^4 \cdot 2^{56} \cdot 2^6 = 2^{66}$ times. We expect that $2^4 \cdot 2^{56} \cdot 2^6 \cdot 2^{-2} = 2^{64}$ candidates remain after it. Since computing a_1 and b_1 requires less operations than for an S-box encryption, the time complexity of this step is less than $2 \cdot 2^{66-8.17} = 2^{58.83}$ full cipher encryptions.

We then look at a_2, a_3, b_2 and b_3. As can be seen in Figure 6.7, the bits that are necessary to compute a_3 and b_3 are $Y_1[3, 7, 11, 15][3]$ and $K_2[7, 15][3]$. Since all these bits are known from previous computations, we can obtain a_3 and b_3 and deduce from the value of ΔY_1 and Lemma 92 the conditions that they must fulfill on their own or with respect to a_2 and b_2.

To simplify the explanation, we consider that a_2 and b_2 are always necessary to check the S-boxes. Note that this simplification is at the disadvantage of the attacker and results in an overestimation of the time complexity.

Bits a_2 and b_2 are given by the following expressions (see also Figure 6.7):

$$a_2 = K_2[7][2] \oplus Y_1[3][2] \oplus Y_1[4][2] \oplus Y_1[14][2]$$
$$b_2 = K_2[15][2] \oplus Y_1[3][2] \oplus Y_1[14][2] \oplus Y_1[15][2]$$

in which the only unknown bit is $Y_1[14][2]$. Since this term appears linearly in both a_2 and b_2, we can obtain a relation relying only on known bits by XORing the two expressions:

$$a_2 \oplus b_2 = K_2[7][2] \oplus K_2[15][2] \oplus Y_1[4][2] \oplus Y_1[15][2].$$

Therefore without any key guessing we can filter our candidates and reduce their number by a factor of 2^{-1}: As a result, 2^{63} candidates remain at

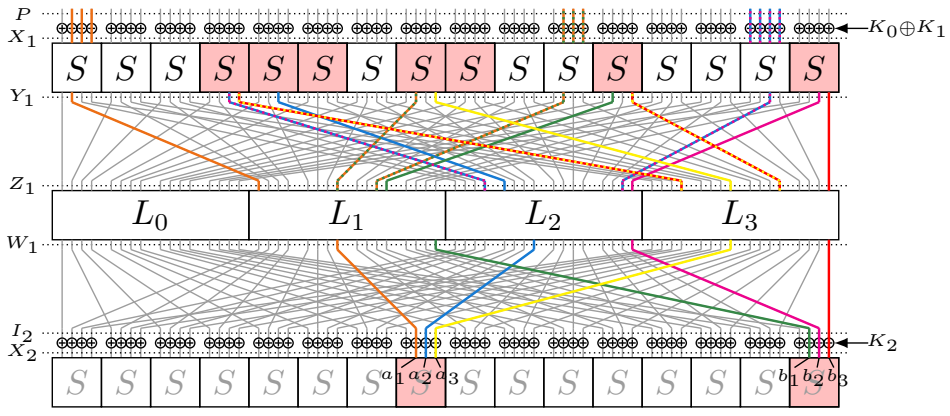


FIGURE 6.7: Bits Involved in the Computation of a_1, a_2, a_3 and b_1, b_2, b_3.

this point, while the complexity of this step is lower than $2^{64-8.17} = 2^{55.83}$ encryptions.

For the remaining candidates, we guess $(K_0 \oplus K_1)[14]$ to be able to compute $Y_1[14][2]$ and we check that a_2 takes the right value. This requires a guess of 4 bits and leads to a reduction of the set of possible candidates by a factor of 2^{-1}. With 2^{67} simple computations (each roughly equal to one S-box computation, so with a time complexity that is less than $2^{58.83}$ PRIDE encryptions), we reduce the number of candidates to 2^{66}.

At this point, we have 2^{66} candidates made of a pair of plaintext/ciphertext associated with a guessed value for 66 key bits. The average count for a wrong key is expected to be 1, while we built our messages so that the right key appears around $a = 2^4$ times.

The distribution of keys in the candidates follows a *Binomial* distribution of parameters $n = 2^{66}$ and $p = 2^{-66}$, i. e., $\mathrm{B}(2^{66}, 2^{-66})$ that can be approximated by a *Poisson* distribution of parameter $\lambda = n \cdot p = 1$ so the probability that a wrong key appears strictly more than t times in our set of candidates is given by:

$$P_t = 1 - \sum_{k=0}^{t} e^{-1} \cdot \frac{1^k}{k!} = 1 - e^{-1} \cdot \sum_{k=0}^{t} \frac{1}{k!}$$

The idea here is to do an additional filtering step (that is, checking the 2 active S-boxes of round 17) only for candidates that are associated with a key that is suggested $t+1$ times or more. Doing so, the ratio of candidates that we have to study is equal to P_t.

We choose $t = 13$, meaning that we are now looking at $2^{66} \cdot 2^{-37.7} = 2^{28.3}$ candidates. For these, we start by computing the value of c_1 and d_1, that as can be seen in Figure 6.8 depend on the following unknown bits:

- For c_1: $K_0[5,6,13][1,2]$ and $K_{18}[5][1] \oplus K_{18}[6][1] \oplus K_{18}[13][1] \oplus K_0[5][3] \oplus K_0[6][3] \oplus K_0[13][3]$.
- For d_1: $K_0[10][1,2]$ and $K_0[10][3] \oplus K_{18}[2][1] \oplus K_{18}[10][1] \oplus K_{18}[11][1]$.

We start by guessing the 3 key bits required to compute d_1, and we filter our guesses by confronting the obtained value with the value given by

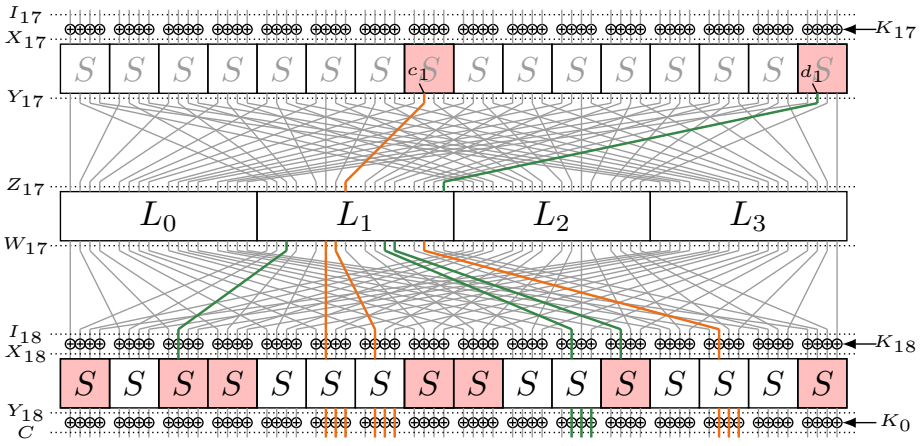


FIGURE 6.8: Bits Involved in the Computation of c_1 and d_1.

Lemma 92. The filtering ratio is of 2^{-1}, so the number of candidates after this step is: $2^{28.3} \times 2^3 \times 2^{-1} = 2^{30.3}$.

Next, we repeat the same process by guessing the 7 unknown key bits that are necessary to compute c_1. The number of candidates obtained at this point is $2^{30.3} \times 2^7 \times 2^{-1} = 2^{36.3}$, and the time complexity of these two steps is negligible in comparison to previous ones.

For all the key candidates that are (still) suggested 14 times or more, we do an exhaustive search to find the value of the $128-76 = 52$ unknown key bits and check them by doing a trial encryption. Since we expect $2^{11.3}$ such key candidates, this step will at most require $2^{11.3} \cdot 2^{52} = 2^{63.3}$ encryptions.

To sum up, the total data complexity of our attack is 2^{61} chosen plaintexts, its time complexity is less than $2^{63.3}$ 18-round PRIDE encryption, and its memory complexity is of 2^{35} 64-bit blocks.

Part IV

EPILOGUE

7

Conclusion and Future Works

به پایان آمد این دفتر حکایت همچنان باقی

سعدی

"At an end this book almost is, remains though the tale." —Saadi

7.1 CONCLUSION

In this thesis, we focus on two main topics in the direction of the design and the security of hardware-oriented SPN block ciphers: countermeasures against fault injection attacks, and low-latency designs. In the following, we briefly explain the results of the research in this thesis.

Countermeasures Against Fault Injection Attacks

In Chapter 3, we presented a comprehensive methodology for the correct implementation of EDC-based CED schemes. We defined an adversary model restricted to injecting a limited number of faults at any location of the circuit, including the FSM and the control signals. To the best of our knowledge, we presented the first secure and efficient EDC-based design methodology that guarantees the detection of up to a certain number of faults.

We also introduced a design methodology to apply ECCs to protect against SIFA. The proposed CEC scheme guarantees the correction of faults up to a number of faults, again in any location of the circuit, i. e., the CEC scheme protects against SIFA as long as the number of injected faults do not exceed the number of fault corrections.

We presented the tweakable block cipher CRAFT, for which the resistance of its implementations against DFA attacks was taken into account during the design phase. For the unprotected implementation and the one equipped with fault-detection mechanisms, the corresponding results show a clear distance between CRAFT and state of the art. To the best of our knowledge, it is a unique construction with a 128-bit key whose round-based implementation (requiring 32 clock cycles to encrypt a 64-bit message) needs less than 1000 GE. Further, it offers two other interesting features by (a) supporting a 64-bit tweak which adds around 245 GE area, and (b) being able to turn into a decryption function with a very low area overhead of around 140 GE.

Low-Latency Designs

In Chapter 4, we modified the block cipher PRINCE to reach the required security level set by the NIST with minimal overhead, especially in a situation

where PRINCE is already deployed. To reach our target, we carefully studied the key-schedule of the cipher while (almost) all of the remaining design is kept untouched, and we proposed the block cipher PRINCEv2. We provided various experiments to show that PRINCEv2 only has a very small overhead compared to PRINCE. Moreover, the fact that the PRINCE and PRINCEv2 designs are very similar allows one to implement both PRINCE and PRINCEv2 in the same environment (e. g., for backward compatibility) with a very small overhead. We thus believe that PRINCEv2 is a significant improvement over PRINCE, and we expect it to be widely adopted in the near future. Moreover, our work shows that one can improve the security level of some lightweight primitives with minimal downsides.

We also studied the latency of the (vectorial) Boolean functions and bijective S-boxes. We introduced the *latency complexity* metric to measure the latency of Boolean functions mathematically. Based on the latency complexity, we presented an efficient algorithm to find all the Boolean functions with low-latency complexity. We also presented another efficient algorithm to build bijective low-latency S-boxes. As a result, we found 5- and 6-bit low-latency S-boxes with cryptographically good properties.

Computing EDP of Differentials and ELP of Linear Hulls

In Chapter 5, we showed that the previous main method for approximating the EDP of a truncated differential is not reliable and may produce estimations very different from the correct value. We presented new methods that, based only on Assumption 16, allow us to practically compute the EDP of (truncated) differentials and the ELP of (multidimensional) linear hulls. We applied these methods to SPN block ciphers CRAFT, MIDORI, SKINNY, KLEIN, and PRINCE. Several observations have shown that an important one is that in the differential or the linear activity patterns with the same number of rounds, having less active S-boxes or fewer conditions in the linear layers does not promise a better EDP or ELP.

CRAFT, PRINCEv2, and PRIDE Cryptanalysis

In Chapter 6, we analyzed the security of the CRAFT and PRINCEv2 block ciphers in detail.

Since the general structure of CRAFT is similar to that of AES, MIDORI, SKINNY, and MANTIS, the security analysis of CRAFT is also more or less similar to that of those primitives. We provided the argument to show the security claim for CRAFT block cipher considering the security of the cipher against accelerated exhaustive search, TDM, (impossible) differential, (zero-correlation) linear, MitM, integral, division property, and nonlinear invariant attacks.

We analyzed the security of PRINCEv2 based on the previously published analysis of PRINCE. We showed that several attacks successful against PRINCE, such as certain accelerated exhaustive search and MitM attacks, do not apply to PRINCEv2. Additionally, we provided several new analyses that include a linear attack, a 6-round integral distinguisher based on the division property, a more precise evaluation of boomerang attack using recently published techniques, and a new 10-round Demirci-Selçuk MitM attack.

We also provided new insights and a better understanding of differential attacks of PRIDE block cipher. We showed that two previous differential attacks are incorrect and described (new and old) properties of the cipher that make such attacks intricate. Based on this understanding, we showed how to mount a differential attack properly.

7.2 FUTURE WORKS

Our research in the topics of this thesis leave some possibilities for future research that we point out some of them in the following:

Countermeasures Against Fault Injection Attacks

- We applied $[s+p, s, d]^m$ codes as the underlying EDCs and ECCs for the CED and CEC schemes. Since the S-box layers in an SPN cipher apply m parallel S-boxes, this choice of codes intuitively leads to an efficient implementation of the CED and CEC schemes. One other possibility is to apply $[a \cdot s + p', a \cdot s, d]^{m/a}$ codes (with a being a divisor of m), i. e., the codes with a rank that is a multiplication of the S-box size. The parity size in the second application ($p' \cdot m/a$) can be smaller than the parity size in our application ($p \cdot m$), and therefore the redundant part A' may have a smaller overhead.

- In our CED schemes, we always considered that the encryption part A and the predictor part A' are separated (see Figures 3.5 and 3.6), and there is no interaction between their circuits. This consideration necessitates separating the control signals and the FSM, which causes a large implementation overhead. A clear example of such overhead is the implementation of multiplexers in the A' part. While the multiplexers in the A part are controlled with a single-bit control signal s_i, the corresponding multiplexers in the A' part must be controlled with the control signal s'_i that its size is more than 1-bit. This causes using several layers of multiplexers that increase the area and the delay of the implementation.

  Applying systematic codes makes it possible to combine both the A and A' parts to a single part A^* without using extra decoders for the output. This technique of combining cannot increase the area or the delay of the implementation but, on the other hand, allows for some simplifications in the implementation that can lead to smaller cost than before. Such a simplification can be using the same control signal s_i for both multiplexers in the A and A' parts while still the control signal s'_i is used in the check point. Another simplification that may occur is in combining the S and S' S-boxes. While they are separate, they use the same s bit input and output s and p bits, resp. In the combined S-box S^*, both the input and the output are $s+p$ bits, but only s bits from the input or the output are independent. The linear dependency between the input or the output bits can cause simplifications in the circuit for the S-box S^*. For example, such a simplification in combining the A and A' parts to A^* is in the implementation for the permutation FRIT

that the separate implementation of A and A' uses more area than the implementation for A^*.

- In the CEC schemes with injective F (see Figure 3.21), it is also possible to combine A and A' parts. Note that in the CEC schemes with non-injective F, the two parts are already combined (see Figure 3.22). Combining A and A' parts in the structure for injective F simplifies the structure to a similar one for the non-injective F. By experience, we already know that the overhead of using the structure in Figure 3.22 is significantly less than the one for the structure in Figure 3.21.

- Even though we presented schemes that are either able to detect the injected faults or able to correct them, it is an interesting idea to investigate the possibilities for designing a scheme with the ability to both detect and correct the faults, i. e., a scheme which can correct up to t_1 number of faults and detect up to t_2 faults with $t_2 > t_1$. In error-detection and error-correction schemes, we know that using an $[n, k, d]$-code with $d > t_2 + t_1$, it is possible to correct the errors e with $\mathrm{hw}(e) \leq t_1$ and detect the ones with $t_1 < \mathrm{hw}(e) \leq t_2$. But, this may be different in the case of fault injections which needs to be studied precisely. For instance, using the Extended Hamming code $[8, 4, 4]$, it is possible to correct the errors with Hamming weight 1 and detect the ones with Hamming weight 2. Is it also possible to correct 1-bit fault injections and detect 2-bit ones?

Low-Latency Designs

- An open question is to see if similar improvements to PRINCE could be made for other microcontroller-targeted block ciphers such as MIDORI, MANTIS, and QARMA; that is to see if minimal modifications in the (twea)key schedule of these block ciphers can improve their security level.

- Since determining the latency complexity of a Boolean function is an NP-hard problem, presenting an efficient algorithm to find an upper-bound for the latency complexity is useful. Such an algorithm would present a low-latency implementation of the Boolean function (not necessarily with the minimum latency as of the latency complexity), which is interesting for a designer to reduce the latency of the implementation.

- A clear future work for our research in this chapter is finding the low-latency Boolean functions of 7- and 8-bits, i. e., building $\mathcal{F}_{7,4}$ and $\mathcal{F}_{8,5}$ sets. Then we can build low-latency bijective 7- and 8-bit S-boxes.

- Our algorithm for building bijective S-boxes is not only for building low-latency S-boxes. With simple modifications, this algorithm is applicable to build S-boxes with other properties. An interesting question in this area is to find all the 6-bit bijective S-boxes with minimum linearity, 16, up to affine equivalence.

Computing EDP of Differentials and ELP of Linear Hulls

- Given a truncated differential characteristic, how can we determine the differential(s) within the truncated differential characteristic with the maximum EDP.
- In both of our advanced word-by-word methods, we used a greedy method for the order of the variables. Presenting a solution to find the optimum solution for the order of the variables is interesting.

Bibliography

[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. http://eprint.iacr.org/2012/712. 2012 (cit. on pp. 153, 157).

[Agh+18] Anita Aghaie, Amir Moradi, Shahram Rasoolzadeh, Aein Rezaei Shahmirzadi, Falk Schellenberg, and Tobias Schneider. *Impeccable Circuits*. Cryptology ePrint Archive, Report 2018/203. https://eprint.iacr.org/2018/203. 2018 (cit. on pp. 51, 60, 64, 65).

[Agh+20] Anita Aghaie, Amir Moradi, Shahram Rasoolzadeh, Aein Rezaei Shahmirzadi, Falk Schellenberg, and Tobias Schneider. "Impeccable Circuits". In: *IEEE Trans. Computers* 69.3 (2020), pp. 361–376. DOI: 10.1109/TC.2019.2948617. URL: https://doi.org/10.1109/TC.2019.2948617 (cit. on pp. 6, 51, 60, 64, 65).

[Ago+10a] Michel Agoyan, Jean-Max Dutertre, Amir-Pasha Mirbaha, David Naccache, Anne-Lise Ribotta, and Assia Tria. "How to flip a bit?" In: *16th IEEE International On-Line Testing Symposium (IOLTS 2010), 5-7 July, 2010, Corfu, Greece*. IEEE Computer Society, 2010, pp. 235–239. DOI: 10.1109/IOLTS.2010.5560194. URL: https://doi.org/10.1109/IOLTS.2010.5560194 (cit. on p. 41).

[Ago+10b] Michel Agoyan, Jean-Max Dutertre, David Naccache, Bruno Robisson, and Assia Tria. "When Clocks Fail: On Critical Paths and Clock Faults". In: *Smart Card Research and Advanced Application, 9th IFIP WG 8.8/11.2 International Conference, CARDIS 2010, Passau, Germany, April 14-16, 2010. Proceedings*. Ed. by Dieter Gollmann, Jean-Louis Lanet, and Julien Iguchi-Cartigny. Vol. 6035. Lecture Notes in Computer Science. Springer, 2010, pp. 182–193. DOI: 10.1007/978-3-642-12510-2_13. URL: https://doi.org/10.1007/978-3-642-12510-2_13 (cit. on p. 41).

[AA20] Siavash Ahmadi and Mohammad Reza Aref. "Generalized Meet in the Middle Cryptanalysis of Block Ciphers With an Automated Search Algorithm". In: *IEEE Access* 8 (2020), pp. 2284–2301. DOI: 10.1109/ACCESS.2019.2962101. URL: https://doi.org/10.1109/ACCESS.2019.2962101 (cit. on p. 152).

[Akd+12] Kahraman D. Akdemir, Zhen Wang, Mark G. Karpovsky, and Berk Sunar. "Design of Cryptographic Devices Resilient to Fault Injection Attacks Using Nonlinear Robust Codes". In: ed. by Marc Joye and Michael Tunstall. ISC. Springer, Heidelberg, 2012, pp. 171–199. ISBN: 978-3-642-29655-0. DOI: 10.1007/978-3-642-29656-7 (cit. on p. 47).

[Alb+14] Martin R. Albrecht, Benedikt Driessen, Elif Bilge Kavun, Gregor Leander, Christof Paar, and Tolga Yalçin. "Block Ciphers - Focus on the Linear Layer (feat. PRIDE)". In: *CRYPTO 2014, Part I*. Ed. by Juan A. Garay and Rosario Gennaro. Vol. 8616. LNCS. Springer, Heidelberg, Aug. 2014, pp. 57–76. DOI: 10.1007/978-3-662-44371-2_4 (cit. on pp. 17, 21, 38, 158, 159).

[AL13] Martin R. Albrecht and Gregor Leander. "An All-In-One Approach to Differential Cryptanalysis for Small Block Ciphers". In: *SAC 2012*. Ed. by Lars R. Knudsen and Huapeng Wu. Vol. 7707. LNCS. Springer, Heidelberg, Aug. 2013, pp. 1–15. DOI: 10.1007/978-3-642-35999-6_1 (cit. on pp. 115, 116).

[Ana+16] Charalampos Ananiadis, Athanasios Papadimitriou, David Hély, Vincent Beroulle, Paolo Maistri, and Régis Leveugle. "On the Development of a New Countermeasure Based on a Laser Attack RTL Fault Model". In: *2016 Design, Automation & Test in Europe Conference & Exhibition, DATE 2016, Dresden, Germany, March 14-18, 2016*. Ed. by Luca Fanucci and Jürgen Teich. IEEE, 2016, pp. 445–450. URL: http://ieeexplore.ieee.org/document/7459352/ (cit. on p. 42).

[Ank+17] Ralph Ankele, Subhadeep Banik, Avik Chakraborti, Eik List, Florian Mendel, Siang Meng Sim, and Gaoli Wang. "Related-Key Impossible-Differential Attack on Reduced-Round Skinny". In: *ACNS 17*. Ed. by Dieter Gollmann, Atsuko Miyaji, and Hiroaki Kikuchi. Vol. 10355. LNCS. Springer, Heidelberg, July 2017, pp. 208–228. DOI: 10.1007/978-3-319-61204-1_11 (cit. on p. 147).

[Ank+19] Ralph Ankele, Christoph Dobraunig, Jian Guo, Eran Lambooij, Gregor Leander, and Yosuke Todo. "Zero-Correlation Attacks on Tweakable Block Ciphers". In: *IACR Trans. Symm. Cryptol.* 2019.1 (2019), pp. 192–235. ISSN: 2519-173X. DOI: 10.13154/tosc.v2019.i1.192-235 (cit. on p. 151).

[AK19] Ralph Ankele and Stefan Kölbl. "Mind the Gap - A Closer Look at the Security of Block Ciphers against Differential Cryptanalysis". In: *SAC 2018*. Ed. by Carlos Cid and Michael J. Jacobson Jr: vol. 11349. LNCS. Springer, Heidelberg, Aug. 2019, pp. 163–190. DOI: 10.1007/978-3-030-10970-7_8 (cit. on pp. 115, 116, 140).

[Ava17] Roberto Avanzi. "The QARMA Block Cipher Family". In: *IACR Trans. Symm. Cryptol.* 2017.1 (2017), pp. 4–44. ISSN: 2519-173X. DOI: 10.13154/tosc.v2017.i1.4-44 (cit. on pp. 18, 38, 39, 96, 99).

[Ban+15] Subhadeep Banik, Andrey Bogdanov, Takanori Isobe, Kyoji Shibutani, Harunaga Hiwatari, Toru Akishita, and Francesco Regazzoni. "Midori: A Block Cipher for Low Energy". In: *ASIACRYPT 2015, Part II*. Ed. by Tetsu Iwata and Jung Hee Cheon. Vol. 9453. LNCS. Springer, Heidelberg, 2015, pp. 411–436. DOI: 10.1007/978-3-662-48800-3_17 (cit. on pp. 16, 38, 39, 67, 69, 79, 99, 143, 144).

[Ban+17] Subhadeep Banik, Sumit Kumar Pandey, Thomas Peyrin, Yu Sasaki, Siang Meng Sim, and Yosuke Todo. "GIFT: A Small Present - Towards Reaching the Limit of Lightweight Encryption". In: *CHES 2017*. Ed. by Wieland Fischer and Naofumi Homma. Vol. 10529. LNCS. Springer, Heidelberg, Sept. 2017, pp. 321–345. DOI: 10.1007/978-3-319-66787-4_16 (cit. on p. 38).

[Bar+06] Hagai Bar-El, Hamid Choukri, David Naccache, Michael Tunstall, and Claire Whelan. "The Sorcerer's Apprentice Guide to Fault Attacks". In: *Proceedings of the IEEE* 94.2 (2006), pp. 370–382. DOI: 10.1109/JPROC.2005.862424. URL: https://doi.org/10.1109/JPROC.2005.862424 (cit. on p. 42).

[BR15] Elaine Barker and Allen Roginsky. "Recommendation for Transitioning the Use of Cryptographic Algorithms and Key Lengths Report on Lightweight Cryptography". In: *National Institute of Standards and Technology* (2015). Available online at https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-131Ar1.pdf (cit. on pp. 13, 20).

[Bea+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. http://eprint.iacr.org/2013/404. 2013 (cit. on p. 38).

[Bea+15] Ray Beaulieu, Douglas Shors, Jason Smith, Stefan Treatman-Clark, Bryan Weeks, and Louis Wingers. "The SIMON and SPECK lightweight block ciphers". In: *Proceedings of the 52nd Annual Design Automation Conference, San Francisco, CA, USA, June 7-11, 2015*. ACM, 2015, 175:1–175:6. DOI: 10.1145/2744769.2747946. URL: https://doi.org/10.1145/2744769.2747946 (cit. on pp. 38, 67, 79, 80).

[Bei+17] Christof Beierle, Anne Canteaut, Gregor Leander, and Yann Rotella. "Proving Resistance Against Invariant Attacks: How to Choose the Round Constants". In: *CRYPTO 2017, Part II*. Ed. by Jonathan Katz and Hovav Shacham. Vol. 10402. LNCS. Springer, Heidelberg, Aug. 2017, pp. 647–678. DOI: 10.1007/978-3-319-63715-0_22 (cit. on p. 150).

[Bei+16] Christof Beierle, Jérémy Jean, Stefan Kölbl, Gregor Leander, Amir Moradi, Thomas Peyrin, Yu Sasaki, Pascal Sasdrich, and Siang Meng Sim. "The SKINNY Family of Block Ciphers and Its Low-Latency Variant MANTIS". In: *CRYPTO 2016, Part II*. Ed. by Matthew Robshaw and Jonathan Katz. Vol. 9815. LNCS. Springer, Heidelberg, Aug. 2016, pp. 123–153. DOI: 10.1007/978-3-662-53008-5_5 (cit. on pp. 16, 18, 38, 39, 67, 96, 99, 143, 144, 147).

[Bei+19] Christof Beierle, Gregor Leander, Amir Moradi, and Shahram Rasoolzadeh. "CRAFT: Lightweight Tweakable Block Cipher with Efficient Protection Against DFA Attacks". In: *IACR Trans. Symm. Cryptol.* 2019.1 (2019), pp. 5–45. ISSN: 2519-173X. DOI: 10.13154/tosc.v2019.i1.5-45 (cit. on pp. 7, 51, 143, 151).

[Ber+03] Guido Bertoni, Luca Breveglieri, Israel Koren, Paolo Maistri, and Vincenzo Piuri. "Error Analysis and Detection Procedures for a Hardware Implementation of the Advanced Encryption Standard". In: *IEEE Trans. Computers* 52.4 (2003), pp. 492–505. DOI: 10.1109/TC.2003.1190590. URL: https://doi.org/10.1109/TC.2003.1190590 (cit. on pp. 41–44, 52).

[Bha+15] Ritam Bhaumik, Avijit Dutta, Jian Guo, Jérémy Jean, Nicky Mouha, and Ivica Nikolić. *More Rounds, Less Security?* Cryptology ePrint Archive, Report 2015/484. http://eprint.iacr.org/2015/484. 2015 (cit. on p. 158).

[BBS99a] Eli Biham, Alex Biryukov, and Adi Shamir. "Cryptanalysis of Skipjack Reduced to 31 Rounds Using Impossible Differentials". In: *EUROCRYPT'99*. Ed. by Jacques Stern. Vol. 1592. LNCS. Springer, Heidelberg, May 1999, pp. 12–23. DOI: 10.1007/3-540-48910-X_2 (cit. on p. 28).

[BBS99b] Eli Biham, Alex Biryukov, and Adi Shamir. "Miss in the Middle Attacks on IDEA and Khufu". In: *FSE'99*. Ed. by Lars R. Knudsen. Vol. 1636. LNCS. Springer, Heidelberg, Mar. 1999, pp. 124–138. DOI: 10.1007/3-540-48519-8_10 (cit. on p. 28).

[BS91] Eli Biham and Adi Shamir. "Differential Cryptanalysis of DES-like Cryptosystems". In: *CRYPTO'90*. Ed. by Alfred J. Menezes and Scott A. Vanstone. Vol. 537. LNCS. Springer, Heidelberg, Aug. 1991, pp. 2–21. DOI: 10.1007/3-540-38424-3_1 (cit. on pp. 22, 23).

[BS97] Eli Biham and Adi Shamir. "Differential Fault Analysis of Secret Key Cryptosystems". In: *CRYPTO'97*. Ed. by Burton S. Kaliski Jr. Vol. 1294. LNCS. Springer, Heidelberg, Aug. 1997, pp. 513–525. DOI: 10.1007/BFb0052259 (cit. on pp. 40, 41).

[Bil+13] Begül Bilgin, Andrey Bogdanov, Miroslav Knežević, Florian Mendel, and Qingju Wang. "Fides: Lightweight Authenticated Cipher with Side-Channel Resistance for Constrained Hardware". In: *CHES 2013*. Ed. by Guido Bertoni and Jean-Sébastien Coron. Vol. 8086. LNCS. Springer, Heidelberg, Aug. 2013, pp. 142–158. DOI: 10.1007/978-3-642-40349-1_9 (cit. on p. 38).

[Bil+12] Begül Bilgin, Svetla Nikova, Ventzislav Nikov, Vincent Rijmen, and Georg Stütz. "Threshold Implementations of All 3 × 3 and 4 × 4 S-Boxes". In: *CHES 2012*. Ed. by Emmanuel Prouff and Patrick Schaumont. Vol. 7428. LNCS. Springer, Heidelberg, Sept. 2012, pp. 76–91. DOI: 10.1007/978-3-642-33027-8_5 (cit. on p. 73).

[BK09] Alex Biryukov and Dmitry Khovratovich. "Related-Key Cryptanalysis of the Full AES-192 and AES-256". In: *ASIACRYPT 2009*. Ed. by Mitsuru Matsui. Vol. 5912. LNCS. Springer, Heidelberg, Dec. 2009, pp. 1–18. DOI: 10.1007/978-3-642-10366-7_1 (cit. on p. 156).

[BW99] Alex Biryukov and David Wagner. "Slide Attacks". In: *FSE'99*. Ed. by Lars R. Knudsen. Vol. 1636. LNCS. Springer, Heidelberg, Mar. 1999, pp. 245–259. DOI: 10.1007/3-540-48519-8_18 (cit. on p. 17).

[Bla03] Richard E. Blahut. *Algebraic Codes for Data Transmission*. Cambridge: Cambridge Univ. Press, 2003. URL: https://cds.cern.ch/record/634022 (cit. on p. 46).

[Bog+07] Andrey Bogdanov, Lars R. Knudsen, Gregor Leander, Christof Paar, Axel Poschmann, Matthew J. B. Robshaw, Yannick Seurin, and C. Vikkelsoe. "PRESENT: An Ultra-Lightweight Block Cipher". In: *CHES 2007*. Ed. by Pascal Paillier and Ingrid Verbauwhede. Vol. 4727. LNCS. Springer, Heidelberg, Sept. 2007, pp. 450–466. DOI: 10.1007/978-3-540-74735-2_31 (cit. on p. 38).

[BR11] Andrey Bogdanov and Christian Rechberger. "A 3-Subset Meet-in-the-Middle Attack: Cryptanalysis of the Lightweight Block Cipher KTANTAN". In: *SAC 2010*. Ed. by Alex Biryukov, Guang Gong, and Douglas R. Stinson. Vol. 6544. LNCS. Springer, Heidelberg, Aug. 2011, pp. 229–240. DOI: 10.1007/978-3-642-19574-7_16 (cit. on pp. 16, 21).

[BR14] Andrey Bogdanov and Vincent Rijmen. "Linear hulls with correlation zero and linear cryptanalysis of block ciphers". In: *Des. Codes Cryptogr.* 70.3 (2014), pp. 369–383. DOI: 10.1007/s10623-012-9697-z. URL: https://doi.org/10.1007/s10623-012-9697-z (cit. on p. 34).

[BDL97] Dan Boneh, Richard A. DeMillo, and Richard J. Lipton. "On the Importance of Checking Cryptographic Protocols for Faults (Extended Abstract)". In: *EUROCRYPT'97*. Ed. by Walter Fumy. Vol. 1233. LNCS. Springer, Heidelberg, May 1997, pp. 37–51. DOI: 10.1007/3-540-69053-0_4 (cit. on pp. 40, 41).

[Bor+12] Julia Borghoff et al. "PRINCE - A Low-Latency Block Cipher for Pervasive Computing Applications - Extended Abstract". In: *ASIACRYPT 2012*. Ed. by Xiaoyun Wang and Kazue Sako. Vol. 7658. LNCS. Springer, Heidelberg, Dec. 2012, pp. 208–225. DOI: 10.1007/978-3-642-34961-4_14 (cit. on pp. 17, 20, 21, 38, 39, 89–91, 96, 100).

[BNS14] Christina Boura, María Naya-Plasencia, and Valentin Suder. "Scrutinizing and Improving Impossible Differential Attacks: Applications to CLEFIA, Camellia, LBlock and Simon". In: *ASIACRYPT 2014, Part I*. Ed. by Palash Sarkar and Tetsu Iwata. Vol. 8873. LNCS. Springer, Heidelberg, Dec. 2014, pp. 179–199. DOI: 10.1007/978-3-662-45611-8_10 (cit. on p. 156).

[BMP08] Joan Boyar, Philip Matthews, and René Peralta. "On the Shortest Linear Straight-Line Program for Computing Linear Forms". In: *Mathematical Foundations of Computer Science 2008, 33rd International Symposium, MFCS 2008, Torun, Poland, August 25-29, 2008, Proceedings*. Ed. by Edward Ochmanski and Jerzy Tyszkiewicz. Vol. 5162. Lecture Notes in Computer Science. Springer, 2008, pp. 168–179. DOI: 10.1007/978-3-540-85238-4_13. URL: https://doi.org/10.1007/978-3-540-85238-4_13 (cit. on p. 103).

[BPP00] Joan Boyar, René Peralta, and Denis Pochuev. "On the Multiplicative Complexity of Boolean Functions over the Basis (∧, ⊕, 1)". In: *Theor. Comput. Sci.* 235.1 (2000), pp. 43–57. DOI: 10.1016/S0304-3975(99)00182-6. URL: https://doi.org/10.1016/S0304-3975(99)00182-6 (cit. on p. 37).

[Bož+20] Dušan Božilov, Maria Eichlseder, Miroslav Knežević, Baptiste Lambin, Gregor Leander, Thorben Moos, Ventzislav Nikov, Shahram Rasoolzadeh, Yosuke Todo, and Friedrich Wiemer. "PRINCEv2: More Security for (Almost) No Overhead". In: *SAC 2020*. Ed. by Orr Dunkelman, Michael J. Jacobson Jr., and Colin O'Flynn. Vol. x. LNCS. Springer, Heidelberg, Dec. 2020, pp. x–x (cit. on pp. 7, 89, 97, 143).

[Bre+20] Jakub Breier, Mustafa Khairallah, Xiaolu Hou, and Yang Liu. "A Countermeasure Against Statistical Ineffective Fault Analysis". In: *IEEE Transactions on Circuits and Systems II: Express Briefs* (2020), pp. 1–1 (cit. on p. 84).

[Bri+14] Julien Bringer, Claude Carlet, Hervé Chabanne, Sylvain Guilley, and Houssem Maghrebi. "Orthogonal Direct Sum Masking - A Smartcard Friendly Computation Paradigm in a Code, with Builtin Protection against Side-Channel and Fault Attacks". In: *Information Security Theory and Practice. Securing the Internet of Things - 8th IFIP WG 11.2 International Workshop, WISTP 2014, Heraklion, Crete, Greece, June 30 - July 2, 2014. Proceedings*. Ed. by David Naccache and Damien Sauveron. Vol. 8501. Lecture Notes in Computer Science. Springer, 2014, pp. 40–56. DOI: 10.1007/978-3-662-43826-8_4. URL: https://doi.org/10.1007/978-3-662-43826-8_4 (cit. on p. 42).

[BPS90] Lawrence Brown, Josef Pieprzyk, and Jennifer Seberry. "LOKI - A Cryptographic Primitive for Authentication and Secrecy Applications". In: *AUSCRYPT'90*. Ed. by Jennifer Seberry and Josef Pieprzyk. Vol. 453. LNCS. Springer, Heidelberg, Jan. 1990, pp. 229–236. DOI: 10.1007/BFb0030364 (cit. on p. 20).

[Cop] *COPACOBANA: A Codebreaker for DES and other Ciphers*. http://www.sciengines.com/copacobana/. 2006 (cit. on p. 20).

[Can+15] Anne Canteaut, Thomas Fuhr, Henri Gilbert, María Naya-Plasencia, and Jean-René Reinhard. "Multiple Differential Cryptanalysis of Round-Reduced PRINCE". In: *FSE 2014*. Ed. by Carlos Cid and Christian Rechberger. Vol. 8540. LNCS. Springer, Heidelberg, Mar. 2015, pp. 591–610. DOI: 10.1007/978-3-662-46706-0_30 (cit. on pp. 115, 117, 139, 153–155).

[Can+17] Anne Canteaut, Eran Lambooij, Samuel Neves, Shahram Rasoolzadeh, Yu Sasaki, and Marc Stevens. "Refined Probability of Differential Characteristics Including Dependency Between Multiple Rounds". In: *IACR Trans. Symm. Cryptol.* 2017.2 (2017), pp. 203–227. ISSN: 2519-173X. DOI: 10.13154/tosc.v2017.i2.203-227 (cit. on p. 7).

[CNV13] Anne Canteaut, María Naya-Plasencia, and Bastien Vayssière. "Sieve-in-the-Middle: Improved MITM Attacks". In: *CRYPTO 2013, Part I*. Ed. by Ran Canetti and Juan A. Garay. Vol. 8042. LNCS. Springer, Heidelberg, Aug. 2013, pp. 222–240. DOI: 10.1007/978-3-642-40041-4_13 (cit. on pp. 22, 157).

[Car07] Claude Carlet. "Boolean Functions for Cryptography and Error Correcting Codes". In: *Boolean Methods and Models*. Ed. by Yves Crama and Peter Hammer. Cambridge University Press, 2007 (cit. on pp. 11, 31).

[CG16] Claude Carlet and Sylvain Guilley. "Complementary Dual Codes for Counter-Measures to Side-Channel Attacks". In: *Adv. Math. Commun.* 10.1 (2016), pp. 131–150. URL: http://aimsciences.org/journals/displayArticlesnew.jsp?paperID=12289 (cit. on p. 42).

[Cla07] Christophe Clavier. "Secret External Encodings Do Not Prevent Transient Fault Analysis". In: *CHES 2007*. Ed. by Pascal Paillier and Ingrid Verbauwhede. Vol. 4727. LNCS. Springer, Heidelberg, Sept. 2007, pp. 181–194. DOI: 10.1007/978-3-540-74735-2_13 (cit. on pp. 41, 80).

[Cui+16] Tingting Cui, Keting Jia, Kai Fu, Shiyao Chen, and Meiqin Wang. *New Automatic Search Tool for Impossible Differentials and Zero-Correlation Linear Approximations*. Cryptology ePrint Archive, Report 2016/689. http://eprint.iacr.org/2016/689. 2016 (cit. on p. 147).

[Dae95] Joan Daemen. *Cipher and Hash Function Design, Strategies Based on Linear and Differential Cryptanalysis, PhD Thesis*. K.U.Leuven, 1995 (cit. on p. 38).

[Dae+19] Joan Daemen, Christoph Dobraunig, Maria Eichlseder, Hannes Gross, Florian Mendel, and Robert Primas. *Protecting against Statistical Ineffective Fault Attacks*. Cryptology ePrint Archive, Report 2019/536. https://eprint.iacr.org/2019/536. 2019 (cit. on p. 84).

[DGV95] Joan Daemen, René Govaerts, and Joos Vandewalle. "Correlation Matrices". In: *FSE'94*. Ed. by Bart Preneel. Vol. 1008. LNCS. Springer, Heidelberg, Dec. 1995, pp. 275–285. DOI: 10.1007/3-540-60590-8_21 (cit. on pp. 31, 32, 116).

[DKR97] Joan Daemen, Lars R. Knudsen, and Vincent Rijmen. "The Block Cipher Square". In: *FSE'97*. Ed. by Eli Biham. Vol. 1267. LNCS. Springer, Heidelberg, Jan. 1997, pp. 149–165. DOI: 10.1007/BFb0052343 (cit. on p. 34).

[Dae+00] Joan Daemen, Michaël Peeters, Gilles Van Assche, and Vincent Rijmen. "Nessie Proposal: NOEKEON". In: *First Open NESSIE Workshop*. 2000, pp. 213–230 (cit. on p. 38).

[DR02] Joan Daemen and Vincent Rijmen. *The Design of Rijndael: AES - The Advanced Encryption Standard*. Information Security and Cryptography. Springer, 2002. ISBN: 3-540-42580-2. DOI: 10.1007/978-3-662-04722-4. URL: https://doi.org/10.1007/978-3-662-04722-4 (cit. on pp. 3, 31).

[DC14] Yibin Dai and Shaozhen Chen. *Cryptanalysis of Full PRIDE Block Cipher*. Cryptology ePrint Archive, Report 2014/987. http://eprint.iacr.org/2014/987. 2014 (cit. on p. 158).

[DDK09] Christophe De Cannière, Orr Dunkelman, and Miroslav Knežević. "KATAN and KTANTAN - A Family of Small and Efficient Hardware-Oriented Block Ciphers". In: *CHES 2009*. Ed. by Christophe Clavier and Kris Gaj. Vol. 5747. LNCS. Springer, Heidelberg, Sept. 2009, pp. 272–288. DOI: 10.1007/978-3-642-04138-9_20 (cit. on pp. 16, 21, 38, 67, 79).

[Deh+12] Amine Dehbaoui, Jean-Max Dutertre, Bruno Robisson, and Assia Tria. "Electromagnetic Transient Faults Injection on a Hardware and a Software Implementations of AES". In: *2012 Workshop on Fault Diagnosis and Tolerance in Cryptography, Leuven, Belgium, September 9, 2012*. Ed. by Guido Bertoni and Benedikt Gierlichs. IEEE Computer Society, 2012, pp. 7–15. DOI: 10.1109/FDTC.2012.15. URL: https://doi.org/10.1109/FDTC.2012.15 (cit. on p. 41).

[Der19] Patrick Derbez. *AES Automatic Tool*. Available at https://seafile.cifex-dedibox.ovh/f/72be1bc96bf740d3a854/. 2019 (cit. on p. 157).

[DP15] Patrick Derbez and Léo Perrin. "Meet-in-the-Middle Attacks and Structural Analysis of Round-Reduced PRINCE". In: *FSE 2015*. Ed. by Gregor Leander. Vol. 9054. LNCS. Springer, Heidelberg, Mar. 2015, pp. 190–216. DOI: 10.1007/978-3-662-48116-5_10 (cit. on pp. 153, 157).

[DH77] Whitfield Diffie and Martin E. Hellman. “Special Feature Exhaustive Cryptanalysis of the NBS Data Encryption Standard”. In: *Computer* 10.6 (1977), pp. 74–84. DOI: 10.1109/C-M.1977.217750. URL: https://doi.org/10.1109/C-M.1977.217750 (cit. on p. 21).

[Din+17] Yao-Ling Ding, Jing-Yuan Zhao, Lei-Bo Li, and Hong-Bo Yu. “Impossible Differential Analysis on Round-Reduced PRINCE”. In: *J. Inf. Sci. Eng.* 33.4 (2017), pp. 1041–1053. URL: http://jise.iis.sinica.edu.tw/JISESearch/pages/View/PaperView.jsf?keyId=157_2080 (cit. on pp. 153, 155).

[Din15] Itai Dinur. “Cryptanalytic Time-Memory-Data Tradeoffs for FX-Constructions with Applications to PRINCE and PRIDE”. In: *EUROCRYPT 2015, Part I*. Ed. by Elisabeth Oswald and Marc Fischlin. Vol. 9056. LNCS. Springer, Heidelberg, Apr. 2015, pp. 231–253. DOI: 10.1007/978-3-662-46800-5_10 (cit. on pp. 21, 157, 158).

[Dob+18] Christoph Dobraunig, Maria Eichlseder, Thomas Korak, Stefan Mangard, Florian Mendel, and Robert Primas. “SIFA: Exploiting Ineffective Fault Inductions on Symmetric Cryptography”. In: *IACR TCHES* 2018.3 (2018). https://tches.iacr.org/index.php/TCHES/article/view/7286, pp. 547–572. ISSN: 2569-2925. DOI: 10.13154/tches.v2018.i3.547-572 (cit. on pp. 41, 42, 80, 84, 88).

[Dob+19] Christoph Dobraunig, Maria Eichlseder, Florian Mendel, and Markus Schofnegger. “Algebraic Cryptanalysis of Variants of Frit”. In: *SAC 2019*. Ed. by Kenneth G. Paterson and Douglas Stebila. Vol. 11959. LNCS. Springer, Heidelberg, Aug. 2019, pp. 149–170. DOI: 10.1007/978-3-030-38471-5_7 (cit. on p. 67).

[DKS10] Orr Dunkelman, Nathan Keller, and Adi Shamir. “A Practical-Time Related-Key Attack on the KASUMI Cryptosystem Used in GSM and 3G Telephony”. In: *CRYPTO 2010*. Ed. by Tal Rabin. Vol. 6223. LNCS. Springer, Heidelberg, Aug. 2010, pp. 393–410. DOI: 10.1007/978-3-642-14623-7_21 (cit. on p. 156).

[EFF98] EFF. *DES Cracker Machine*. https://web.archive.org/web/20170507231657/https://w2.eff.org/Privacy/Crypto/Crypto_misc/DESCracker/HTML/19980716_eff_des_faq.html. 1998 (cit. on p. 20).

[EK18] Maria Eichlseder and Daniel Kales. “Clustering Related-Tweak Characteristics: Application to MANTIS-6”. In: *IACR Trans. Symm. Cryptol.* 2018.2 (2018), pp. 111–132. ISSN: 2519-173X. DOI: 10.13154/tosc.v2018.i2.111-132 (cit. on pp. 115, 117).

[ELR20] Maria Eichlseder, Gregor Leander, and Shahram Rasoolzadeh. “Computing Expected Differential Probability of (Truncated) Differentials and Expected Linear Potential of (Multidimensional) Linear Hulls in SPN Block Ciphers”. In: *INDOCRYPT 2020*. Ed. by Karthikeyan Bhargavan, Elisabeth Oswald, and Manoj Prabhakaran. Vol. 12578. LNCS. Springer, Heidelberg, Dec. 2020, pp. 345–369. DOI: 10.1007/978-3-030-65277-7_16 (cit. on pp. 7, 115).

[EY19] Muhammad ElSheikh and Amr M. Youssef. “Related-Key Differential Cryptanalysis of Full Round CRAFT”. In: *Security, Privacy, and Applied Cryptography Engineering - 9th International Conference, SPACE 2019, Gandhinagar, India, December 3-7, 2019, Proceedings*. Ed. by Shivam Bhasin, Avi Mendelson, and Mridul Nandi. Vol. 11947. Lecture Notes in Computer Science. Springer, 2019, pp. 50–66. DOI: 10.1007/978-3-030-35869-3_6. URL: https://doi.org/10.1007/978-3-030-35869-3_6 (cit. on pp. 151, 152).

[FIP77] FIPS. “46: Data Encryption Standard”. In: *Federal Information Processing Standards Publication* (1977) (cit. on pp. 3, 16).

[FIP01] FIPS. “197: Advanced encryption standard (AES)”. In: *Federal Information Processing Standards Publication* (2001). Available online at http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf (cit. on p. 3).

[Fei71] Horst Feistel. *Block Cipher Cryptographic System*. US Patent 3,798,359. 1971 (cit. on pp. 3, 16).

[FJM14] Pierre-Alain Fouque, Antoine Joux, and Chrysanthi Mavromati. “Multi-user Collisions: Applications to Discrete Logarithm, Even-Mansour and PRINCE”. In: *ASIACRYPT 2014, Part I*. Ed. by Palash Sarkar and Tetsu Iwata. Vol. 8873. LNCS. Springer, Heidelberg, Dec. 2014, pp. 420–438. DOI: 10.1007/978-3-662-45611-8_22 (cit. on p. 157).

[Fuh+13] Thomas Fuhr, Éliane Jaulmes, Victor Lomné, and Adrian Thillard. “Fault Attacks on AES with Faulty Ciphertexts Only”. In: *2013 Workshop on Fault Diagnosis and Tolerance in Cryptography, Los Alamitos, CA, USA, August 20, 2013*. Ed. by Wieland Fischer and Jörn-Marc Schmidt. IEEE Computer Society, 2013, pp. 108–118. DOI: 10.1109/FDTC.2013.18. URL: https://doi.org/10.1109/FDTC.2013.18 (cit. on p. 41).

[GSK06] Gunnar Gaubatz, Berk Sunar, and Mark G. Karpovsky. “Non-linear Residue Codes for Robust Public-Key Arithmetic”. In: *Fault Diagnosis and Tolerance in Cryptography, Third International Workshop, FDTC 2006, Yokohama, Japan, October 10, 2006, Proceedings*. Ed. by Luca Breveglieri, Israel Koren, David Naccache, and Jean-Pierre Seifert. Vol. 4236. Lecture Notes in Computer Science. Springer, 2006, pp. 173–184. DOI: 10.1007/11889700_16. URL: https://doi.org/10.1007/11889700_16 (cit. on p. 47).

[Gér+13] Benoît Gérard, Vincent Grosso, María Naya-Plasencia, and François-Xavier Standaert. "Block Ciphers That Are Easier to Mask: How Far Can We Go?" In: *CHES 2013*. Ed. by Guido Bertoni and Jean-Sébastien Coron. Vol. 8086. LNCS. Springer, Heidelberg, Aug. 2013, pp. 383–399. DOI: 10.1007/978-3-642-40349-1_22 (cit. on p. 38).

[Gha+14] Nahid Farhady Ghalaty, Bilgiday Yuce, Mostafa M. I. Taha, and Patrick Schaumont. "Differential Fault Intensity Analysis". In: *2014 Workshop on Fault Diagnosis and Tolerance in Cryptography, FDTC 2014, Busan, South Korea, September 23, 2014*. Ed. by Assia Tria and Dooho Choi. IEEE Computer Society, 2014, pp. 49–58. DOI: 10.1109/FDTC.2014.15. URL: https://doi.org/10.1109/FDTC.2014.15 (cit. on p. 41).

[Goh19] Aron Gohr. "Improving Attacks on Round-Reduced Speck32/64 Using Deep Learning". In: *CRYPTO 2019, Part II*. Ed. by Alexandra Boldyreva and Daniele Micciancio. Vol. 11693. LNCS. Springer, Heidelberg, Aug. 2019, pp. 150–179. DOI: 10.1007/978-3-030-26951-7_6 (cit. on p. 116).

[GNL11] Zheng Gong, Svetla Nikova, and Yee Wei Law. "KLEIN: A New Family of Lightweight Block Ciphers". In: *RFID. Security and Privacy - 7th International Workshop, RFIDSec 2011, Amherst, USA, June 26-28, 2011, Revised Selected Papers*. Ed. by Ari Juels and Christof Paar. Vol. 7055. Lecture Notes in Computer Science. Springer, 2011, pp. 1–18. DOI: 10.1007/978-3-642-25286-0_1. URL: https://doi.org/10.1007/978-3-642-25286-0_1 (cit. on p. 137).

[GR16] Lorenzo Grassi and Christian Rechberger. "Practical Low Data-Complexity Subspace-Trail Cryptanalysis of Round-Reduced PRINCE". In: *INDOCRYPT 2016*. Ed. by Orr Dunkelman and Somitra Kumar Sanadhya. Vol. 10095. LNCS. Springer, Heidelberg, Dec. 2016, pp. 322–342. DOI: 10.1007/978-3-319-49890-4_18 (cit. on p. 153).

[Gro+15] Vincent Grosso, Gaëtan Leurent, François-Xavier Standaert, and Kerem Varici. "LS-Designs: Bitslice Encryption for Efficient Masked Software Implementations". In: *FSE 2014*. Ed. by Carlos Cid and Christian Rechberger. Vol. 8540. LNCS. Springer, Heidelberg, Mar. 2015, pp. 18–37. DOI: 10.1007/978-3-662-46706-0_2 (cit. on pp. 38, 159).

[Guo+11] Jian Guo, Thomas Peyrin, Axel Poschmann, and Matthew J. B. Robshaw. "The LED Block Cipher". In: *CHES 2011*. Ed. by Bart Preneel and Tsuyoshi Takagi. Vol. 6917. LNCS. Springer, Heidelberg, 2011, pp. 326–341. DOI: 10.1007/978-3-642-23951-9_22 (cit. on p. 38).

[Guo+15] Xiaofei Guo, Debdeep Mukhopadhyay, Chenglu Jin, and Ramesh Karri. "Security analysis of concurrent error detection against differential fault analysis". In: *Journal of Cryptographic Engineering* 5.3 (Sept. 2015), pp. 153–169. DOI: 10.1007/s13389-014-0092-8 (cit. on pp. 42, 46).

[Had+19] Hosein Hadipour, Sadegh Sadeghi, Majid M. Niknam, Ling Song, and Nasour Bagheri. "Comprehensive security analysis of CRAFT". In: *IACR Trans. Symm. Cryptol.* 2019.4 (2019), pp. 290–317. ISSN: 2519-173X. DOI: 10.13154/tosc.v2019.i4.290-317 (cit. on pp. 151, 152).

[Hel80] Martin E. Hellman. "A Cryptanalytic Time-Memory Trade-off". In: *IEEE Trans. Inf. Theory* 26.4 (1980), pp. 401–406. DOI: 10.1109/TIT.1980.1056220. URL: https://doi.org/10.1109/TIT.1980.1056220 (cit. on p. 20).

[HCN19] Miia Hermelin, Joo Yeon Cho, and Kaisa Nyberg. "Multidimensional Linear Cryptanalysis". In: *Journal of Cryptology* 32.1 (Jan. 2019), pp. 1–34. DOI: 10.1007/s00145-018-9308-x (cit. on p. 34).

[Ish+06] Yuval Ishai, Manoj Prabhakaran, Amit Sahai, and David Wagner. "Private Circuits II: Keeping Secrets in Tamperable Circuits". In: *EUROCRYPT 2006*. Ed. by Serge Vaudenay. Vol. 4004. LNCS. Springer, Heidelberg, 2006, pp. 308–327. DOI: 10.1007/11761679_19 (cit. on p. 53).

[JK97] Thomas Jakobsen and Lars R. Knudsen. "The Interpolation Attack on Block Ciphers". In: *FSE'97*. Ed. by Eli Biham. Vol. 1267. LNCS. Springer, Heidelberg, Jan. 1997, pp. 28–40. DOI: 10.1007/BFb0052332 (cit. on p. 28).

[Jea+17] Jérémy Jean, Amir Moradi, Thomas Peyrin, and Pascal Sasdrich. "Bit-Sliding: A Generic Technique for Bit-Serial Implementations of SPN-based Primitives - Applications to AES, PRESENT and SKINNY". In: *CHES 2017*. Ed. by Wieland Fischer and Naofumi Homma. Vol. 10529. LNCS. Springer, Heidelberg, Sept. 2017, pp. 687–707. DOI: 10.1007/978-3-319-66787-4_33 (cit. on p. 79).

[JNP14] Jérémy Jean, Ivica Nikolic, and Thomas Peyrin. "Tweaks and Keys for Block Ciphers: The TWEAKEY Framework". In: *ASIACRYPT 2014, Part II*. Ed. by Palash Sarkar and Tetsu Iwata. Vol. 8874. LNCS. Springer, Heidelberg, Dec. 2014, pp. 274–288. DOI: 10.1007/978-3-662-45608-8_15 (cit. on pp. 18, 38).

[Jea+14] Jérémy Jean, Ivica Nikolic, Thomas Peyrin, Lei Wang, and Shuang Wu. "Security Analysis of PRINCE". In: *FSE 2013*. Ed. by Shiho Moriai. Vol. 8424. LNCS. Springer, Heidelberg, Mar. 2014, pp. 92–111. DOI: 10.1007/978-3-662-43933-3_6 (cit. on pp. 156–158).

[KR94] Burton S. Kaliski Jr. and Matthew J. B. Robshaw. “Linear Cryptanalysis Using Multiple Approximations”. In: *CRYPTO’94*. Ed. by Yvo Desmedt. Vol. 839. LNCS. Springer, Heidelberg, Aug. 1994, pp. 26–39. DOI: 10.1007/3-540-48658-5_4 (cit. on p. 34).

[KKT04] Mark G. Karpovsky, Konrad J. Kulikowski, and Alexander Taubin. “Robust Protection against Fault-Injection Attacks on Smart Cards Implementing the Advanced Encryption Standard”. In: *2004 International Conference on Dependable Systems and Networks (DSN 2004), 28 June - 1 July 2004, Florence, Italy, Proceedings*. IEEE Computer Society, 2004, pp. 93–101. DOI: 10.1109/DSN.2004.1311880. URL: https://doi.org/10.1109/DSN.2004.1311880 (cit. on p. 47).

[Kar+02] Ramesh Karri, Kaijie Wu, Piyush Mishra, and Yongkook Kim. “Concurrent error detection schemes for fault-based side-channel cryptanalysis of symmetric block ciphers”. In: *IEEE Trans. on CAD of Integrated Circuits and Systems* 21.12 (2002), pp. 1509–1517. DOI: 10.1109/TCAD.2002.804378. URL: https://doi.org/10.1109/TCAD.2002.804378 (cit. on pp. 43, 46).

[KSW96] John Kelsey, Bruce Schneier, and David Wagner. “Key-Schedule Cryptanalysis of IDEA, G-DES, GOST, SAFER, and Triple-DES”. In: *CRYPTO’96*. Ed. by Neal Koblitz. Vol. 1109. LNCS. Springer, Heidelberg, Aug. 1996, pp. 237–251. DOI: 10.1007/3-540-68697-5_19 (cit. on p. 78).

[KR96] Joe Kilian and Phillip Rogaway. “How to Protect DES Against Exhaustive Key Search”. In: *CRYPTO’96*. Ed. by Neal Koblitz. Vol. 1109. LNCS. Springer, Heidelberg, Aug. 1996, pp. 252–267. DOI: 10.1007/3-540-68697-5_20 (cit. on p. 17).

[KNR12] Miroslav Knežević, Ventzislav Nikov, and Peter Rombouts. “Low-Latency Encryption - Is “Lightweight = Light + Wait”?” In: *CHES 2012*. Ed. by Emmanuel Prouff and Patrick Schaumont. Vol. 7428. LNCS. Springer, Heidelberg, Sept. 2012, pp. 426–446. DOI: 10.1007/978-3-642-33027-8_25 (cit. on p. 39).

[Knu93] Lars R. Knudsen. “Cryptanalysis of LOKI”. In: *ASIACRYPT’91*. Ed. by Hideki Imai, Ronald L. Rivest, and Tsutomu Matsumoto. Vol. 739. LNCS. Springer, Heidelberg, Nov. 1993, pp. 22–35. DOI: 10.1007/3-540-57332-1_2 (cit. on p. 20).

[Knu95] Lars R. Knudsen. “Truncated and Higher Order Differentials”. In: *FSE’94*. Ed. by Bart Preneel. Vol. 1008. LNCS. Springer, Heidelberg, Dec. 1995, pp. 196–211. DOI: 10.1007/3-540-60590-8_16 (cit. on pp. 27, 28).

[KB96] Lars R. Knudsen and Thomas A. Berson. “Truncated Differentials of SAFER”. In: *FSE’96*. Ed. by Dieter Gollmann. Vol. 1039. LNCS. Springer, Heidelberg, Feb. 1996, pp. 15–26. DOI: 10.1007/3-540-60865-6_38 (cit. on p. 115).

[KW02] Lars R. Knudsen and David Wagner. “Integral Cryptanalysis”. In: *FSE 2002*. Ed. by Joan Daemen and Vincent Rijmen. Vol. 2365. LNCS. Springer, Heidelberg, Feb. 2002, pp. 112–127. DOI: 10.1007/3-540-45661-9_9 (cit. on p. 34).

[Knu98] Lars Knudsen. “DEAL - A 128-bit Block Cipher”. In: *NIST AES Proposal* (1998) (cit. on p. 28).

[Köl14] Stefan Kölbl. *CryptoSMT: An Easy to Use Tool for Cryptanalysis of Symmetric Primitives*. Available at https://github.com/kste/cryptosmt. 2014 (cit. on pp. 115, 116, 151).

[KLW17] Thorsten Kranz, Gregor Leander, and Friedrich Wiemer. “Linear Cryptanalysis: Key Schedules and Tweakable Block Ciphers”. In: *IACR Trans. Symm. Cryptol.* 2017.1 (2017), pp. 474–505. ISSN: 2519-173X. DOI: 10.13154/tosc.v2017.i1.474-505 (cit. on p. 145).

[KKT07] Konrad J. Kulikowski, Mark G. Karpovsky, and Alexander Taubin. “Robust codes and robust, fault-tolerant architectures of the Advanced Encryption Standard”. In: *J. Syst. Archit.* 53.2-3 (2007), pp. 139–149. DOI: 10.1016/j.sysarc.2006.09.007. URL: https://doi.org/10.1016/j.sysarc.2006.09.007 (cit. on p. 47).

[KWK08] Konrad J. Kulikowski, Zhen Wang, and Mark G. Karpovsky. “Comparative Analysis of Robust Fault Attack Resistant Architectures for Public and Private Cryptosystems”. In: *Fifth International Workshop on Fault Diagnosis and Tolerance in Cryptography, 2008, FDTC 2008, Washington, DC, USA, 10 August 2008*. Ed. by Luca Breveglieri, Shay Gueron, Israel Koren, David Naccache, and Jean-Pierre Seifert. IEEE Computer Society, 2008, pp. 41–50. DOI: 10.1109/FDTC.2008.13. URL: https://doi.org/10.1109/FDTC.2008.13 (cit. on p. 47).

[Kum+06] Sandeep Kumar, Christof Paar, Jan Pelzl, Gerd Pfeiffer, and Manfred Schimmler. “Breaking Ciphers with COPACOBANA - A Cost-Optimized Parallel Code Breaker”. In: *CHES 2006*. Ed. by Louis Goubin and Mitsuru Matsui. Vol. 4249. LNCS. Springer, Heidelberg, Oct. 2006, pp. 101–118. DOI: 10.1007/11894063_9 (cit. on p. 20).

[Lai94] Xuejia Lai. "Higher Order Derivatives and Differential Cryptanalysis". In: *Communications and Cryptography*. Ed. by Richard E. Blahut, Daniel J. Costello, Ueli Maurer, and Thomas Mittelholzer. Vol. 276. Series in Engineering and Computer Science (Communications and Information Theory). Springer, 1994, pp. 227–233 (cit. on p. 28).

[LMM91] Xuejia Lai, James L. Massey, and Sean Murphy. "Markov Ciphers and Differential Cryptanalysis". In: *EUROCRYPT'91*. Ed. by Donald W. Davies. Vol. 547. LNCS. Springer, Heidelberg, Apr. 1991, pp. 17–38. DOI: 10.1007/3-540-46416-6_2 (cit. on pp. 23, 24).

[LN15] Virginie Lallemand and María Naya-Plasencia. "Cryptanalysis of KLEIN". In: *FSE 2014*. Ed. by Carlos Cid and Christian Rechberger. Vol. 8540. LNCS. Springer, Heidelberg, Mar. 2015, pp. 451–470. DOI: 10.1007/978-3-662-46706-0_23 (cit. on pp. 137, 138).

[LR17] Virginie Lallemand and Shahram Rasoolzadeh. "Differential Cryptanalysis of 18-Round PRIDE". In: *INDOCRYPT 2017*. Ed. by Arpita Patra and Nigel P. Smart. Vol. 10698. LNCS. Springer, Heidelberg, Dec. 2017, pp. 126–146 (cit. on pp. 7, 143).

[LP07] Gregor Leander and Axel Poschmann. "On the Classification of 4 Bit S-Boxes". In: *Arithmetic of Finite Fields, First International Workshop, WAIFI 2007, Madrid, Spain, June 21-22, 2007, Proceedings*. Ed. by Claude Carlet and Berk Sunar. Vol. 4547. Lecture Notes in Computer Science. Springer, 2007, pp. 159–176. DOI: 10.1007/978-3-540-73074-3_13. URL: https://doi.org/10.1007/978-3-540-73074-3_13 (cit. on pp. 105, 110, 120).

[Leu16] Gaëtan Leurent. "Differential Forgery Attack Against LAC". In: *SAC 2015*. Ed. by Orr Dunkelman and Liam Keliher. Vol. 9566. LNCS. Springer, Heidelberg, Aug. 2016, pp. 217–224. DOI: 10.1007/978-3-319-31301-6_13 (cit. on pp. 115, 117).

[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. http://eprint.iacr.org/2013/573. 2013 (cit. on p. 157).

[Li+10] Yang Li, Kazuo Sakiyama, Shigeto Gomisawa, Toshinori Fukunaga, Junko Takahashi, and Kazuo Ohta. "Fault Sensitivity Analysis". In: *CHES 2010*. Ed. by Stefan Mangard and François-Xavier Standaert. Vol. 6225. LNCS. Springer, Heidelberg, Aug. 2010, pp. 320–334. DOI: 10.1007/978-3-642-15031-9_22 (cit. on pp. 41, 80).

[LN96] Rudolf Lidl and Harald Niederreiter. *Finite Fields*. 2nd ed. Encyclopedia of Mathematics and its Applications. Cambridge: Cambridge University Press, 1996. DOI: 10.1017/CBO9780511525926 (cit. on p. 11).

[LRW02] Moses Liskov, Ronald L. Rivest, and David Wagner. "Tweakable Block Ciphers". In: *CRYPTO 2002*. Ed. by Moti Yung. Vol. 2442. LNCS. Springer, Heidelberg, Aug. 2002, pp. 31–46. DOI: 10.1007/3-540-45708-9_3 (cit. on pp. 17, 38).

[LGS17] Guozhen Liu, Mohona Ghosh, and Ling Song. "Security Analysis of SKINNY under Related-Tweakey Settings (Long Paper)". In: *IACR Trans. Symm. Cryptol.* 2017.3 (2017), pp. 37–72. ISSN: 2519-173X. DOI: 10.13154/tosc.v2017.i3.37-72 (cit. on p. 147).

[MS77] Florence Jessie MacWilliams and Neil James Alexander Sloane. *The Theory of Error Correcting Codes*. North-Holland mathematical library. North-Holland Pub. Co. New York, 1977 (cit. on pp. 44, 46).

[MSY06] Tal Malkin, François-Xavier Standaert, and Moti Yung. "A Comparative Cost/Security Analysis of Fault Attack Countermeasures". In: *Fault Diagnosis and Tolerance in Cryptography, Third International Workshop, FDTC 2006, Yokohama, Japan, October 10, 2006, Proceedings*. Ed. by Luca Breveglieri, Israel Koren, David Naccache, and Jean-Pierre Seifert. Vol. 4236. Lecture Notes in Computer Science. Springer, 2006, pp. 159–172. DOI: 10.1007/11889700_15. URL: https://doi.org/10.1007/11889700_15 (cit. on pp. 43, 47).

[Mat94] Mitsuru Matsui. "Linear Cryptanalysis Method for DES Cipher". In: *EUROCRYPT'93*. Ed. by Tor Helleseth. Vol. 765. LNCS. Springer, Heidelberg, May 1994, pp. 386–397. DOI: 10.1007/3-540-48285-7_33 (cit. on pp. 28–30).

[Mat95] Mitsuru Matsui. "On Correlation Between the Order of S-boxes and the Strength of DES". In: *EUROCRYPT'94*. Ed. by Alfredo De Santis. Vol. 950. LNCS. Springer, Heidelberg, May 1995, pp. 366–375. DOI: 10.1007/BFb0053451 (cit. on pp. 29, 145).

[MY93] Mitsuru Matsui and Atsuhiro Yamagishi. "A New Method for Known Plaintext Attack of FEAL Cipher". In: *EUROCRYPT'92*. Ed. by Rainer A. Rueppel. Vol. 658. LNCS. Springer, Heidelberg, May 1993, pp. 81–91. DOI: 10.1007/3-540-47555-9_7 (cit. on pp. 28, 29).

[MG01] Marine Minier and Henri Gilbert. "Stochastic Cryptanalysis of Crypton". In: *FSE 2000*. Ed. by Bruce Schneier. Vol. 1978. LNCS. Springer, Heidelberg, Apr. 2001, pp. 121–133. DOI: 10.1007/3-540-44706-7_9 (cit. on pp. 115, 117).

[MA19] AmirHossein E. Moghaddam and Zahra Ahmadian. *New Automatic search method for Truncated-differential characteristics: Application to Midori, SKINNY and CRAFT*. Cryptology ePrint Archive, Report 2019/126. https://eprint.iacr.org/2019/126. 2019 (cit. on pp. 119, 120, 151, 152).

[Mor17] Pawel Morawiecki. "Practical attacks on the round-reduced PRINCE". In: *IET Information Security* 11.3 (2017), pp. 146–151. DOI: 10.1049/iet-ifs.2015.0432. URL: https://doi.org/10.1049/iet-ifs.2015.0432 (cit. on pp. 153, 156).

[Mor+99] Shiho Moriai, Makoto Sugita, Kazumaro Aoki, and Masayuki Kanda. "Security of E2 against Truncated Differential Cryptanalysis". In: *SAC 1999*. Ed. by Howard M. Heys and Carlisle M. Adams. Vol. 1758. LNCS. Springer, Heidelberg, Aug. 1999, pp. 106–117. DOI: 10.1007/3-540-46513-8_8 (cit. on pp. 115, 118, 119).

[Mou+11] Nicky Mouha, Qingju Wang, Dawu Gu, and Bart Preneel. "Differential and Linear Cryptanalysis Using Mixed-Integer Linear Programming". In: *Information Security and Cryptology - 7th International Conference, Inscrypt 2011, Beijing, China, November 30 - December 3, 2011. Revised Selected Papers*. Ed. by Chuankun Wu, Moti Yung, and Dongdai Lin. Vol. 7537. Lecture Notes in Computer Science. Springer, 2011, pp. 57–76. DOI: 10.1007/978-3-642-34704-7_5. URL: https://doi.org/10.1007/978-3-642-34704-7_5 (cit. on p. 145).

[NIS98] NIST. "SKIPJACK and KEA Algorithm Specifications". In: *National Institute of Standards and Technology* (1998). Available online at https://csrc.nist.gov/CSRC/media//Projects/Cryptographic-Algorithm-Validation-Program/documents/skipjack/skipjack.pdf (cit. on p. 28).

[NFR07] Giorgio Di Natale, Marie-Lise Flottes, and Bruno Rouzeyre. "An On-Line Fault Detection Scheme for SBoxes in Secure Circuits". In: *13th IEEE International On-Line Testing Symposium (IOLTS 2007), 8-11 July 2007, Heraklion, Crete, Greece*. IEEE Computer Society, 2007, pp. 57–62. DOI: 10.1109/IOLTS.2007.16. URL: https://doi.org/10.1109/IOLTS.2007.16 (cit. on p. 42).

[Nyb95] Kaisa Nyberg. "Linear Approximation of Block Ciphers (Rump Session)". In: *EUROCRYPT'94*. Ed. by Alfredo De Santis. Vol. 950. LNCS. Springer, Heidelberg, May 1995, pp. 439–444. DOI: 10.1007/BFb0053460 (cit. on p. 31).

[Nyb01] Kaisa Nyberg. "Correlation Theorems in Cryptanalysis". In: *Discret. Appl. Math.* 111.1-2 (2001), pp. 177–188. DOI: 10.1016/S0166-218X(00)00351-6. URL: https://doi.org/10.1016/S0166-218X(00)00351-6 (cit. on p. 30).

[NK95] Kaisa Nyberg and Lars R. Knudsen. "Provable Security Against a Differential Attack". In: *Journal of Cryptology* 8.1 (Dec. 1995), pp. 27–37. DOI: 10.1007/BF00204800 (cit. on p. 28).

[Och+05] Vitalij Ocheretnij, G. Kouznetsov, Ramesh Karri, and Michael Gössel. "On-Line Error Detection and BIST for the AES Encryption Algorithm with Different S-Box Implementations". In: *11th IEEE International On-Line Testing Symposium (IOLTS 2005), 6-8 July 2005, Saint Raphael, France*. IEEE Computer Society, 2005, pp. 141–146. DOI: 10.1109/IOLTS.2005.51. URL: https://doi.org/10.1109/IOLTS.2005.51 (cit. on p. 42).

[Pat+15] Sikhar Patranabis, Abhishek Chakraborty, Phuong Ha Nguyen, and Debdeep Mukhopadhyay. "A Biased Fault Attack on the Time Redundancy Countermeasure for AES". In: *COSADE 2015*. Ed. by Stefan Mangard and Axel Y. Poschmann: vol. 9064. LNCS. Springer, Heidelberg, Apr. 2015, pp. 189–203. DOI: 10.1007/978-3-319-21476-4_13 (cit. on p. 52).

[PRC12] Gilles Piret, Thomas Roche, and Claude Carlet. "PICARO - A Block Cipher Allowing Efficient Higher-Order Side-Channel Resistance". In: *ACNS 12*. Ed. by Feng Bao, Pierangela Samarati, and Jianying Zhou. Vol. 7341. LNCS. Springer, Heidelberg, June 2012, pp. 311–328. DOI: 10.1007/978-3-642-31284-7_19 (cit. on p. 38).

[PDN15] Raluca Posteuca, Cristina-Loredana Duta, and Gabriel Negara. "New Approaches For Round-Reduced PRINCE Cipher Cryptanalysis". In: *Proceedings of the Romanian Academy, Series A* 16 (2015), pp. 253–264 (cit. on p. 156).

[PN15] Raluca Posteuca and Gabriel Negara. "Integral Cryptanalysis of Round-Reduced PRINCE Cipher". In: *Proceedings of the Romanian Academy, Series A* 16 (2015), pp. 265–270 (cit. on p. 156).

[RR16a] Shahram Rasoolzadeh and Håvard Raddum. *Cryptanalysis of 6-round PRINCE using 2 Known Plaintexts*. Cryptology ePrint Archive, Report 2016/132. http://eprint.iacr.org/2016/132. 2016 (cit. on p. 156).

[RR16b] Shahram Rasoolzadeh and Håvard Raddum. "Cryptanalysis of PRINCE with Minimal Data". In: *AFRICACRYPT 16*. Ed. by David Pointcheval, Abderrahmane Nitaj, and Tajjeeddine Rachidi. Vol. 9646. LNCS. Springer, Heidelberg, Apr. 2016, pp. 109–126. DOI: 10.1007/978-3-319-31517-1_6 (cit. on pp. 153, 156, 157).

[RR16c] Shahram Rasoolzadeh and Håvard Raddum. "Faster Key Recovery Attack on Round-Reduced PRINCE". In: *Lightweight Cryptography for Security and Privacy - 5th International Workshop, LightSec 2016, Aksaray, Turkey, September 21-22, 2016, Revised Selected Papers*. Ed. by Andrey Bogdanov. Vol. 10098. Lecture Notes in Computer Science. Springer, 2016, pp. 3–17. DOI: 10.1007/978-3-319-55714-4_1. URL: https://doi.org/10.1007/978-3-319-55714-4_1 (cit. on p. 156).

[Sah+19] Sayandeep Saha, Dirmanto Jap, Debapriya Basu Roy, Avik Chakraborti, Shivam Bhasin, and Debdeep Mukhopadhyay. *Transform-and-Encode: A Countermeasure Framework for Statistical Ineffective Fault Attacks on Block Ciphers*. Cryptology ePrint Archive, Report 2019/545. https://eprint.iacr.org/2019/545. 2019 (cit. on p. 84).

[Sas11] Yu Sasaki. "Meet-in-the-Middle Preimage Attacks on AES Hashing Modes and an Application to Whirlpool". In: *FSE 2011*. Ed. by Antoine Joux. Vol. 6733. LNCS. Springer, Heidelberg, Feb. 2011, pp. 378–396. DOI: 10.1007/978-3-642-21702-9_22 (cit. on p. 150).

[SA09a] Yu Sasaki and Kazumaro Aoki. "Finding Preimages in Full MD5 Faster Than Exhaustive Search". In: *EUROCRYPT 2009*. Ed. by Antoine Joux. Vol. 5479. LNCS. Springer, Heidelberg, Apr. 2009, pp. 134–152. DOI: 10.1007/978-3-642-01001-9_8 (cit. on p. 150).

[SA09b] Yu Sasaki and Kazumaro Aoki. "Meet-in-the-Middle Preimage Attacks on Double-Branch Hash Functions: Application to RIPEMD and Others". In: *Information Security and Privacy, 14th Australasian Conference, ACISP 2009, Brisbane, Australia, July 1-3, 2009, Proceedings*. Ed. by Colin Boyd and Juan Manuel González Nieto. Vol. 5594. Lecture Notes in Computer Science. Springer, 2009, pp. 214–231. DOI: 10.1007/978-3-642-02620-1_15. URL: https://doi.org/10.1007/978-3-642-02620-1_15 (cit. on p. 22).

[ST17] Yu Sasaki and Yosuke Todo. "New Impossible Differential Search Tool from Design and Cryptanalysis Aspects - Revealing Structural Properties of Several Ciphers". In: *EUROCRYPT 2017, Part III*. Ed. by Jean-Sébastien Coron and Jesper Buus Nielsen. Vol. 10212. LNCS. Springer, Heidelberg, 2017, pp. 185–215. DOI: 10.1007/978-3-319-56617-7_7 (cit. on pp. 147, 155).

[Sat+08] Akashi Satoh, Takeshi Sugawara, Naofumi Homma, and Takafumi Aoki. "High-Performance Concurrent Error Detection Scheme for AES Hardware". In: *CHES 2008*. Ed. by Elisabeth Oswald and Pankaj Rohatgi. Vol. 5154. LNCS. Springer, Heidelberg, Aug. 2008, pp. 100–112. DOI: 10.1007/978-3-540-85053-3_7 (cit. on p. 43).

[SGD08] Nidhal Selmane, Sylvain Guilley, and Jean-Luc Danger. "Practical Setup Time Violation Attacks on AES". In: *Seventh European Dependable Computing Conference, EDCC-7 2008, Kaunas, Lithuania, 7-9 May 2008*. IEEE Computer Society, 2008, pp. 91–96. DOI: 10.1109/EDCC-7.2008.11. URL: https://doi.org/10.1109/EDCC-7.2008.11 (cit. on p. 41).

[SHS16] Bodo Selmke, Johann Heyszl, and Georg Sigl. "Attack on a DFA Protected AES by Simultaneous Laser Fault Injections". In: *2016 Workshop on Fault Diagnosis and Tolerance in Cryptography, FDTC 2016, Santa Barbara, CA, USA, August 16, 2016*. IEEE Computer Society, 2016, pp. 36–46. DOI: 10.1109/FDTC.2016.16. URL: https://doi.org/10.1109/FDTC.2016.16 (cit. on p. 41).

[SRM19] Aein Rezaei Shahmirzadi, Shahram Rasoolzadeh, and Amir Moradi. *Impeccable Circuits II*. Cryptology ePrint Archive, Report 2019/1369. https://eprint.iacr.org/2019/1369. 2019 (cit. on p. 88).

[SRM20] Aein Rezaei Shahmirzadi, Shahram Rasoolzadeh, and Amir Moradi. "Impeccable Circuits II". In: *57th ACM/IEEE Design Automation Conference, DAC 2020, San Francisco, CA, USA, July 20-24, 2020*. IEEE, 2020, pp. 1–6. DOI: 10.1109/DAC18072.2020.9218615. URL: https://doi.org/10.1109/DAC18072.2020.9218615 (cit. on pp. 6, 52).

[Sha45] Claude E. Shannon. *A Mathematical Theory of Cryptography*. Available at https://www.iacr.org/museum/shannon/shannon45.pdf. 1945 (cit. on pp. 3, 13, 14).

[Shi+11] Kyoji Shibutani, Takanori Isobe, Harunaga Hiwatari, Atsushi Mitsuda, Toru Akishita, and Taizo Shirai. "Piccolo: An Ultra-Lightweight Blockcipher". In: *CHES 2011*. Ed. by Bart Preneel and Tsuyoshi Takagi. Vol. 6917. LNCS. Springer, Heidelberg, 2011, pp. 342–357. DOI: 10.1007/978-3-642-23951-9_23 (cit. on pp. 67, 79).

[SM88] Akihiro Shimizu and Shoji Miyaguchi. "Fast Data Encipherment Algorithm FEAL". In: *EUROCRYPT'87*. Ed. by David Chaum and Wyn L. Price. Vol. 304. LNCS. Springer, Heidelberg, Apr. 1988, pp. 267–278. DOI: 10.1007/3-540-39118-5_24 (cit. on p. 16).

[Sim+18] Thierry Simon, Lejla Batina, Joan Daemen, Vincent Grosso, Pedro Maat Costa Massolino, Kostas Papagiannopoulos, Francesco Regazzoni, and Niels Samwel. *Towards Lightweight Cryptographic Primitives with Built-in Fault-Detection*. Cryptology ePrint Archive, Report 2018/729. https://eprint.iacr.org/2018/729. 2018 (cit. on pp. 38, 67).

[Sor84] Arthur Sorkin. “Lucifer, A Cryptographic Algorithm”. In: *Cryptologia* 8.1 (1984), pp. 22–42. DOI: 10.1080/0161-118491858746. eprint: https://doi.org/10.1080/0161-118491858746. URL: https://doi.org/10.1080/0161-118491858746 (cit. on p. 16).

[Sta+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: *FSE 2004*. Ed. by Bimal K. Roy and Willi Meier. Vol. 3017. LNCS. Springer, Heidelberg, Feb. 2004, pp. 279–299. DOI: 10.1007/978-3-540-25937-4_18 (cit. on p. 38).

[Sto16] Ko Stoffelen. “Optimizing S-Box Implementations for Several Criteria Using SAT Solvers”. In: *FSE 2016*. Ed. by Thomas Peyrin. Vol. 9783. LNCS. Springer, Heidelberg, Mar. 2016, pp. 140–160. DOI: 10.1007/978-3-662-52993-5_8 (cit. on p. 103).

[Sun+15] Bing Sun, Zhiqiang Liu, Vincent Rijmen, Ruilin Li, Lei Cheng, Qingju Wang, Hoda AlKhzaimi, and Chao Li. “Links Among Impossible Differential, Integral and Zero Correlation Linear Cryptanalysis”. In: *CRYPTO 2015, Part I*. Ed. by Rosario Gennaro and Matthew J. B. Robshaw. Vol. 9215. LNCS. Springer, Heidelberg, Aug. 2015, pp. 95–115. DOI: 10.1007/978-3-662-47989-6_5 (cit. on p. 151).

[Sun+13] Siwei Sun, Lei Hu, Ling Song, Yonghong Xie, and Peng Wang. *Automatic Security Evaluation of Block Ciphers with S-bP Structures against Related-key Differential Attacks*. Cryptology ePrint Archive, Report 2013/547. http://eprint.iacr.org/2013/547. 2013 (cit. on p. 145).

[Tez14] Cihangir Tezcan. “Improbable differential attacks on Present using undisturbed bits”. In: *J. Comput. Appl. Math.* 259 (2014), pp. 503–511. DOI: 10.1016/j.cam.2013.06.023. URL: https://doi.org/10.1016/j.cam.2013.06.023 (cit. on pp. 161, 165).

[Tez+16] Cihangir Tezcan, Galip Oral Okan, Asuman Senol, Erol Dogan, Furkan Yücebas, and Nazife Baykal. “Differential Attacks on Lightweight Block Ciphers PRESENT, PRIDE, and RECTANGLE Revisited”. In: *Lightweight Cryptography for Security and Privacy - 5th International Workshop, LightSec 2016, Aksaray, Turkey, September 21-22, 2016, Revised Selected Papers*. Ed. by Andrey Bogdanov. Vol. 10098. Lecture Notes in Computer Science. Springer, 2016, pp. 18–32. DOI: 10.1007/978-3-319-55714-4_2. URL: https://doi.org/10.1007/978-3-319-55714-4_2 (cit. on pp. 158, 161–164).

[Tod15] Yosuke Todo. “Structural Evaluation by Generalized Integral Property”. In: *EUROCRYPT 2015, Part I*. Ed. by Elisabeth Oswald and Marc Fischlin. Vol. 9056. LNCS. Springer, Heidelberg, Apr. 2015, pp. 287–314. DOI: 10.1007/978-3-662-46800-5_12 (cit. on p. 35).

[TM16] Yosuke Todo and Masakatu Morii. “Bit-Based Division Property and Application to Simon Family”. In: *FSE 2016*. Ed. by Thomas Peyrin. Vol. 9783. LNCS. Springer, Heidelberg, Mar. 2016, pp. 357–377. DOI: 10.1007/978-3-662-52993-5_18 (cit. on p. 35).

[Xia+16] Zejun Xiang, Wentao Zhang, Zhenzhen Bao, and Dongdai Lin. “Applying MILP Method to Searching Integral Distinguishers Based on Division Property for 6 Lightweight Block Ciphers”. In: *ASIACRYPT 2016, Part I*. Ed. by Jung Hee Cheon and Tsuyoshi Takagi. Vol. 10031. LNCS. Springer, Heidelberg, Dec. 2016, pp. 648–678. DOI: 10.1007/978-3-662-53887-6_24 (cit. on pp. 150, 156).

[Yan+15] Qianqian Yang, Lei Hu, Siwei Sun, Kexin Qiao, Ling Song, Jinyong Shan, and Xiaoshuang Ma. “Improved Differential Analysis of Block Cipher PRIDE”. In: *Information Security Practice and Experience - 11th International Conference, ISPEC 2015, Beijing, China, May 5-8, 2015. Proceedings*. Ed. by Javier López and Yongdong Wu. Vol. 9065. Lecture Notes in Computer Science. Springer, 2015, pp. 209–219. DOI: 10.1007/978-3-319-17533-1_15. URL: https://doi.org/10.1007/978-3-319-17533-1_15 (cit. on pp. 158, 160, 162–165).

[YJ00] Sung-Ming Yen and Marc Joye. “Checking Before Output May Not Be Enough Against Fault-Based Cryptanalysis”. In: *IEEE Trans. Computers* 49.9 (2000), pp. 967–970. DOI: 10.1109/12.869328. URL: https://doi.org/10.1109/12.869328 (cit. on p. 80).

[YPO15] Zheng Yuan, Zhen Peng, and Haiwen Ou. *Two Kinds of Biclique Attacks on Lightweight Block Cipher PRINCE*. Cryptology ePrint Archive, Report 2015/1208. http://eprint.iacr.org/2015/1208. 2015 (cit. on p. 157).

[Zha+14] Jingyuan Zhao, Xiaoyun Wang, Meiqin Wang, and Xiaoyang Dong. *Differential Analysis on Block Cipher PRIDE*. Cryptology ePrint Archive, Report 2014/525. http://eprint.iacr.org/2014/525. 2014 (cit. on pp. 158, 160, 162–164).

[ZG11] Bo Zhu and Guang Gong. *Multidimensional Meet-in-the-Middle Attack and Its Applications to KATAN32/48/64*. Cryptology ePrint Archive, Report 2011/619. http://eprint.iacr.org/2011/619. 2011 (cit. on p. 22).

Affirmation in Lieu of Oath

I hereby declare under oath that I have prepared the thesis submitted independently and without inadmissible third-party assistance, that I have not used any literature except that cited in the the-sis, and that I have identified all text passages quoted in full or by way of approximation, as well as all graphics, tables and analysis programs used. In addition, I hereby assure that the electronic version of the thesis submitted matches the written version, and that no paper in this or in a similar form has yet been submitted and evaluated as a PhD thesis.

Furthermore, I hereby declare that digital images only include the original data or a clear documentation of the type and extent of the image editing performed with regard to content. I also hereby ensure that I have not made use of any commercial mediation or consulting services.

Date Signature

Versicherung an Eides Statt

Ich versichere an Eides statt, dass ich die eingereichte Dissertation selbstständig und ohne unzulässige fremde Hilfe verfasst, andere als die in ihr angegebene Literatur nicht benutzt und dass ich alle ganz oder annähernd übernommenen Textstellen sowie verwendete Grafiken, Tabellen und Auswertungsprogramme kenntlich gemacht habe.

Außerdem versichere ich, dass die vorgelegte elektronische mit der schriftlichen Version der Dissertation übereinstimmt und die Abhandlung in dieser oder ähnlicher Form noch nicht anderweitig als Promotionsleistung vorgelegt und bewertet wurde.

Datum Unterschrift